Learning to count small and clustered objects with application to bacterial colonies

Minghua Zheng

Supervisors: Dr. Na Helian
Dr. Peter Lane
Dr. Yi Sun
Dr. Allen Donald

School of Physics, Engineering and Computer Science
University of Hertfordshire

Submitted to the University of Hertfordshire in partial fulfilment of the requirement of the degree of PhD with Industry Experience

March 2024

To my loving parents

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 59502 words including appendices, bibliography, footnotes, tables and equations and has fewer than 149 figures.

Minghua Zheng
March 2024

Acknowledgements

I must begin by expressing my deepest gratitude towards my supervisors, Dr. Na Helian, Dr. Peter Lane, Dr. Yi Sun, and Dr. Allen Donald. Thanks to your support and guidance, I am able to identify and solve research problems. I would like to especially thank Dr. Peter Lane who helps me distil the essence of my random research ideas through insightful comments, feedback and advice.

I would like to thank my parents for supporting my study in the UK. Thank you for your unconditional love to help me get through tough times during the Covid-19 pandemic. I would also like to thank Yuki for always believing in me hundred times more than I do, encouraging me to get out my comfort zone, and making me happy every day.

Finally, I would like to thank Synoptics Ltd for providing data sets which are essential to my research. I am also thankful to Synoptics Ltd, the European Regional Development Fund and Hertfordshire LEP that partially fund my research.

Abstract

Counting small and clustered objects is a challenging Computer Vision task with many real-world applications. Many researchers have attempted to apply prevalent machine learning algorithms to count objects. However, feature engineering which is a notoriously difficult part of machine learning algorithm development has yet to address the following difficulties of this task collectively: 1) small object size, 2) clustered objects, 3) expensive cost to collect and annotate data, and 4) various domain or category adaptations.

This research solves these four difficulties collectively with an example application to bacterial colonies. It starts with a thorough investigation into MicrobiaNet, which is the best-performing cardinality classification method for bacterial colony counting to the best of my knowledge. Experimental results empirically prove that high image similarity across different classes is the main issue for this method to count clustered colonies accurately. Additionally, it is empirically identified that the class imbalance has a very limited impact on the counting performance. These two findings shine new light on the direction of future improvement for other researchers.

Because of the limitations of the best-performing cardinality classification method for colony counting, this thesis then poses the counting task as a few-shot regression task. I adapt FamNet to particularly count small colonies and propose a new model called ACFamNet to count small and clustered colonies. ACFamNet addresses the first three aforementioned difficulties by tackling region of interest misalignment and optimising feature extraction during the feature engineering process. A real-world data set is collected for developing and evaluating ACFamNet.

To address all aforementioned difficulties together, I propose ACFamNet Pro which is an advanced ACFamNet with additional multi-head attention mechanism and residual connection to count small and clustered objects. The synergy of these additional components supports the model to achieve a better counting performance and become readily generalisable to objects of a different category by dynamically weighting objects of interest, optimising gradient flow and tackling region of interest misalignment. Extensive experiments are conducted to prove ACFamNet Pro is able to tackle the aforementioned difficulties collectively.

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

b	Batch size
$\boldsymbol{h}$	Hidden states in a neural network
$\boldsymbol{X}$	A set of single vector inputs (matrix) to a neural network
$\boldsymbol{x}$	A single vector input to a neural network
$\boldsymbol{Y}$	A set of single vector outputs (matrix) from a neural network
$\boldsymbol{y}$	A single vector output from a neural network
C	Number of channels (dimensions) of an image
F	Height and width of a (square) kernel
f	The learned mapping function for input data and output data
H	Height of an image
K	Kernel (filter) of a convolutional layer
L	Neural network depth
m	Number of examples in a data set
P	Probability
p	Padding size used in padding operation
s	Stride in a convolution operation
t	Epoch number

W Width of an image

y A single scalar output from a neural network

Greek Symbols

$\boldsymbol{\theta}$ Parameters for a model such as neural network

ε Noise (a constant)

η Learning rate

λ Hyper-parameter in a cost function

σ Non-linear activation function

θ Parameters for a probability mass/density distribution or likelihood function

Superscripts

i Superscript index

j Superscript index

l Superscript index

Subscripts

0 Subscript index

Other Symbols

$\boldsymbol{f}_{PQ}$ Projected query feature

$\boldsymbol{f}_{PS}$ Projected support feature

$\boldsymbol{f}_{\boldsymbol{Q}}$ Query feature

$\boldsymbol{f}'_{\boldsymbol{Q}}$ Final enhanced feature

$\boldsymbol{f}_{\boldsymbol{S}}$ Support feature

$\boldsymbol{b}^{(l)}$ A bias vector for the lth layer in a neural network

$\boldsymbol{R}_{\mathbf{0}}$ Score map

$\boldsymbol{R}_{EN}$ Score map after exemplar normalisation

$\boldsymbol{R}_{SN}$ Score map after spatial normalisation

$\hat{\boldsymbol{\theta}}$ Optimal (or learned) parameters for a model such as neural network

$\mathcal{C}_b$ Base classes in few-shot object counting

$\mathcal{C}_n$ Novel classes in few-shot object counting

D_{KL} Kullback-Leibler (KL) divergence

D_{query} Query data set

D_{support} Support data set

D_{test} Test data set

D_{train} Training data set

$\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\hat{p}_{\text{data}}(\boldsymbol{x},\boldsymbol{y})}$ Expectation of $f(\boldsymbol{x},\boldsymbol{y})$ with respect to $\hat{p}_{\text{data}}(\boldsymbol{x},\boldsymbol{y})$

F^0 Ground truth density function

f^* The real mapping function for input data and output data

$\boldsymbol{H}_i^{(l)}$ Convolution output (tensor) from the ith kernel in $\boldsymbol{K}^{(l)}$ for the lth layer in a neural network

H_Q Query image height

H_S Support image height

$\mathcal{J}$ Cost function

$\boldsymbol{K}_i^{(l)}$ ith kernel in $\boldsymbol{K}^{(l)}$

$\boldsymbol{K}^{(l)}$ A set of kernels (tensor) for the lth layer in a neural network

$k^{(l)}$ A kernel for the lth layer in a neural network

$\mathcal{L}$ Loss function

$\mathcal{N}$ Gaussian distribution

p_{data} True data generating distribution

$\hat{p}_{\text{data}}$ Empirical distribution

p_{model} Data generating distribution from a model

σ_{LReLU} Leaky ReLU activation function

σ_{ReLU} ReLU activation function

σ_{sigmoid} sigmoid activation function

σ_{tanh} tanh activation function

$\mathcal{T}$ A task in meta-learning

$\mathcal{T}_{\text{meta-test}}$ A collection of test tasks in meta-learning

$\mathcal{T}_{\text{meta-train}}$ A collection of training tasks in meta-learning

σ^2 Variance

$\boldsymbol{W}^{(l)}$ A weight matrix for the lth layer in a neural network

$\boldsymbol{\omega}$ Parameters specify a meta learner "how to learn"

$\hat{\boldsymbol{\omega}}$ Optimal (or learned) parameters specify a meta learner "how to learn"

W_Q Query image width

W_S Support image width

$\hat{\boldsymbol{y}}$ A predicted single vector output from a neural network

Acronyms / Abbreviations

ACFamNet Aligned Custom Few-shot Adaptation and Matching Network

Adam Adaptive Moment Estimation

CAM Class Activation Map

CNN Convolutional Neural Network

Conv Convolutional layer

ENorm Exemplar normalisation

FamNet Few-shot Adaptation and Matching Network

GAN Generative Adversarial Network

KL Kullback-Leibler

LH Likelihood

LN Layer normalisation

MAE Mean Absolute Error

MAPE Mean Absolute Percentage Error

MAP Maximum a-posterior

MicrobiaNet Microbia Network

MNAE Mean Normalised Absolute Error

NLL Negative log-likelihood

PCA Principal Component Analysis

ReLU Rectified Linear Units

RMSE Root Mean Square Error

RoI Region of Interest

SAFECount Similarity-Aware Feature Enhancement block for object Counting

SNorm Spatial normalisation

SVD Singular Value Decomposition

t-SNE t-distributed Stochastic Neighbour Embedding

Chapter 1

Introduction

Object counting is defined as a task of counting objects of interest from a digital image or video. This task is easy for human beings but challenging for computer systems because the latter has yet to tackle 1) large variations in object size, 2) large variations in object density, 3) the lack of data and its ground truth label, and 4) various domain/category adaptations. These four challenges are difficult hurdles to clear in the process of feature engineering for machine learning algorithm development. This task has many real-life applications in different domains. For example, crowd monitoring, flying insects monitoring in wildlife, crop monitoring in agriculture, bacterial colonies monitoring in healthcare, etc.

Counting small and clustered objects is a special case of generic object counting with some specialised real-world applications, such as bacterial colony counting for healthcare, bee counting for ecological census, corn counting for agricultural monitoring, etc. These applications not only inherit all difficulties of generic object counting, but also require a careful design to address their small and clustered objects.

Previous studies focus on only some aspects of the aforementioned challenges and have yet to address all these challenges together. For example, MicrobiaNet [38], which is the best-performing cardinality classification method for colony counting to the best of my knowledge, only considers small and clustered colonies but not the lack of labelled data and various domain/category adaptations.[1] FamNet [112] and SAFECount [156] tend to address clustered objects, lack of labelled data and domain/category adaptations effectively but overlook small objects. Therefore, the primary goal of this thesis is to investigate and develop an algorithm that is capable of learning from limited labelled data to count small and clustered objects, as well as being readily able to generalise to a different domain/category.

[1]The best-performing cardinality classification method for colony counting to the best of my knowledge is referred to as MicrobiaNet in this thesis for simplicity. Its authors did not name this method.

Bacterial colony counting is used as an example application of the proposed algorithms. This is because this research collaborates with an industrial partner Synoptics Ltd where I spend my first placement year to study their technology with an aim to revolutionise their existing bacterial colony counting method. A *bacterial colony* is a clonal group of cells grown on the surface or embedded within a substance to support the growth [63]. *Bacterial colony counting* is a process of identifying and enumerating viable bacteria in a captured image. Bacterial colony counting is widely used in biological laboratories to estimate the number of viable bacteria present in a test sample. The counting result is an important indicator of the cleanliness of a surface, the sterility of a product, the presence of a bacterial infection, etc. Conventionally, cells are grown on a circular, transparent and lidded petri dish from which a digital camera takes photos. The grown bacterial colonies are often small and clustered which are difficult to count by human beings and computer systems. Additionally, bacterial colonies have many species that are different in colour, shape, opacity, and density, which increases the counting difficulty.

Another reason to use bacterial colonies as an example of *small and clustered objects* is that they meet two conditions: they are either small in real-life or an individual small object only occupies a small portion (less than 1%) of the image; they are very likely to overlap with other small objects when capturing the image. These two conditions are derived from various definitions of small objects. For instance, an object can be called small if it appears in a high resolution image [81]. An object only occupies a small portion (less than 1%) of the image [73]. It is a small object in real-life [17]. An object only occupies less or equal to 32×32 pixels in MS-COCO data set, such as baseball, tennis, traffic sign, etc [139].

This thesis starts by investigating if MicrobiaNet is adaptable to address the counting challenges of small colony size, clustered colonies and limited labelled data. The generalisation to a different domain/category problem is tentatively neglected here because the outcome of this investigation can help Synoptics Ltd decide if this method is integrable to their in-house colony counting method. This investigation and its consequential outcome lead to the first contribution to knowledge listed in § 1.1.

This thesis also takes a detour from the colony-cardinality classification method which treats counting as a classification task, and pose counting as a few-shot object counting task which treats counting as a regression task based on meta-learning. The present thesis studies the adaptation of few-shot object counting methods, namely FamNet and SAFECount, to address small objects with an example application to bacterial colonies.[2] Specifically, the adaptation focuses on *feature engineering* which is a process of formulating the most

[2]SAFECount was not included in this research originally because it was published two years after FamNet. However, due to its superior performance, it was included at a much later stage of this research.

appropriate features based on data, task and machine learning algorithm, where the *feature* is a numeric representation of raw data [164] to support training a downstream statistical model. In the context of neural networks, any operation performed on features before fully connected layers (or the final output layer if the fully connected layers do not exist) is part of feature engineering. The focus on feature engineering in this thesis is motivated by the dramatic difference between generic objects and small and clustered objects. The adaptation of few-shot object counting methods to address small objects leads to the second and third contributions to knowledge listed in § 1.1.

1.1 Contributions and outline

During the research journey of developing an algorithm to count small and clustered objects with application to bacterial colonies, that is also readily generalisable to a different domain/category based on limited labelled data, this thesis makes several contributions to knowledge:

1. Class imbalance is not the key issue for improving colony counting by cardinality classification. Instead, the image similarity across classes is the main issue for improving this counting method, which shines new light on the direction of future improvement for other researchers.

2. The proposal of ACFamNet which is an adaptation of FamNet with bespoke feature engineering to count small and clustered colonies based on limited data by solving region of interest misalignment and improving feature extraction. The bespoke feature engineering includes end-to-end trainable model, aligned region of interest pooling and optimised feature extraction method.

3. The proposal of ACFamNet Pro which is advanced ACFamNet with improved feature engineering to count small and clustered colonies based on limited data, as well as being readily available to count colonies of a different species, by dynamically weighting objects of interest, optimising gradient flow and solving region of interest misalignment. This improved feature engineering includes additional multi-head attention mechanism, residual connection and aligned region of interest pooling.

This thesis has the following outline.

In Chapter 2, I review existing studies related to object counting.

In Chapter 3, I provide relevant technical background on deep learning, few-shot learning, object counting, and a model called Famnet. Two data sets are also described in this chapter.

In Chapter 4, I investigate if MicrobiaNet is adaptable to address the counting challenges of small colony size, clustered colonies and limited labelled data. The investigation concludes two findings that image similarity across classes is the main issue of counting by cardinality classification, and class imbalance has a very limited impact on the counting performance.

In Chapter 5, I propose ACFamNet and ACFamNet Pro with an aim to optimise feature engineering in FamNet so that it can learn from limited labelled data to count small and clustered colonies, as well as being readily generalisable to colonies of a different domain/category.

In Chapter 6, I conduct a series of experiments to evaluate ACFamNet and ACFamNet Pro. Some components of these two algorithms are proven effective at addressing certain problems of feature engineering.

Finally, I conclude this thesis, identify research limitations, discuss the path forward, and provide some take-home messages for readers in Chapter 7.

Chapter 2

Literature Review

Previous object counting approaches can be broadly categorised into detection based approaches, regression based approaches, density map estimation based approaches, and few-shot learning based approaches. These approaches are reviewed against a question: can the counting algorithm address 1) small object size, 2) clustered objects, 3) limited labelled data, and 4) domain/category adaptation collectively. This chapter considers applications in generic object counting and bacterial colony counting with more focus on the latter. This chapter also reveals a research gap based on the review of existing object counting approaches. Finally, some research questions are defined based on the research gap and some potential solutions to bridge the research gap will be reviewed.

2.1 Detection based counting

A very common way to count objects is the detect-then-count approach. This has two main steps. The first step is to detect all instances of the target object from the image. Then, the second step is to add up the number of detections to obtain the final object count. The first step is a challenging task in itself that has been studied in Computer Vision for many years. All object detection approaches can be further categorised into traditional image processing approaches, machine learning approaches and the combination of them. Some specific problems of small object detection are introduced next before reviewing detection based counting approaches.

2.1.1 Difficulties of small object detection

The difficulties of small object detection can be categorised into two groups based on data and detection algorithms. The discussion of these difficulties assumes the process of capturing

images is already completed. The methods based on hardware design, which is before image capturing, to address small object detection are not considered because they are outside the scope of this project.

The data related difficulties come from the poor-quality appearance and the less pixel information [81]. The less discriminative feature representations [73], high interclass similarity [90] and more possible locations of small objects are also difficulties related to data [17]. Additionally, shadows of small objects may be mis-recognised as the object [160]. Moreover, there is no large public data set for small objects [20].

The detection algorithm related difficulties mainly come from neural networks. For example, activation of smaller objects becomes smaller with the use of neural network's pooling layers [73]. Similarly, a large stride in the convolutional and pooling layers may miss small objects [154]. The imbalance between low-level feature (semantically weak) and high-level feature (semantically strong) increases the detection difficulty [41]. Additionally, detectors often suffer from a low tolerance of bounding box perturbation [20] and the intractable ambiguity caused by overlapping bounding boxes [138]. Moreover, there is limited prior knowledge and experience of developing an algorithm to tackle small objects since most of previous studies focus on generic/medium/large size objects [17, 139].

2.1.2 Counting by traditional image processing approaches

Traditional image processing approaches detect objects by detecting boundaries or regions. The detection of boundaries or regions is often followed by using fixed parameters or separability to recognise clustered objects and reject outliers. These fixed parameters are pre-defined or input by users in real time. Boundary detection can be achieved by using edge detection [8, 89, 24], contour detection [102, 147] and Hough Transform [30, 40, 45, 96]. Compared to the boundary detection, region detection is carried out by using connected components labelling [45], thresholding [159, 6, 19, 134, 22, 69] and template matching [67]. A special region detection approach which is a combination of distance transform and watershed algorithm was used by Clarke et al. [26] and Wong et al. [148] to determine if a colony is separable before splitting for counting.

The main drawback of using these traditional image processing approaches to detect objects is that they are parametric, becoming difficult to automatically adapt to a different object size, object density and domain/category. For example, a colony cluster is kept for counting if its area is greater than 30% of median of all colonies [69], which may not work for a different colony species. Similarly, the criteria used to identify clustered colonies is either based on human intervention or inflexible parameters. These parametric approaches are also difficult to automate. For instance, users are asked to define the minimal radius, area and

height before detecting clustered colonies [45]. Additionally, region detection approaches are less accurate if the watershed algorithm is involved due to over-segmentation [111]. Therefore, these parametric approaches have a limited effect of counting small and clustered objects.

Despite the aforementioned disadvantages, traditional image processing approaches avoid the difficulty of collecting and annotating data. This is because these approaches are not required to learn from data and their ground truth. Another benefit of these approaches is that they can correctly adapt to a different domain/category because users can manually change parameters for the adaptation.

2.1.3 Counting by machine learning approaches

Machine learning algorithms have been prevalent in Computer Vision since they are able to identify patterns in images to perform some specific tasks without being explicitly programmed. They can overcome the inflexibility encountered in traditional image processing methods. A major obstacle to the development of machine learning algorithms is the requirement of a large amount of data collection and data annotation. Domain/category adaptation is also challenging since most machine learning algorithms are dependent on training data. This means a machine learning algorithm needs to be retrained on a different data set in order to adapt to a different domain/category. Moreover, very little is currently known how well machine learning algorithms address small and clustered objects.

Many studies that use machine learning algorithms to count objects have overlooked clustered objects, which leads to an inaccurate count. K-means clustering was applied by Chen and Zhang [19] to group each pixel in an image into one of three categories: background, colonies and artefacts. Among classified colonies, users have to select colonies of the target strain to train a non-linear support vector machine (SVM) classifier to identify colonies of different species. This study overlooked clustered colonies and introduced a hurdle for full automation. Similarly, each pixel in an image is classified into either foreground or background by a convolutional neural network (CNN) designed by Andreini et al. [4]. But in their work, the counting of identified colony segments and the splitting of clustered colonies were not taken into account. The handling of clustered objects was also neglected in the research conducted by Hilsenbeck et al. [57], Liu and Yang [85] and Sadanandan et al. [123], even though machine learning algorithms were used to address image segmentation. A recent research [101] transferred Mask R-CNN [55] from object detection to both colony detection and colony species classification. But their examples did not contain many clustered colonies and only two species of colonies were considered. Beznik et al. [10] developed a neural network to classify each pixel in an image into either the border

of clustered colonies, background, virulent colony or avirulent colony. But the outcome of classification from this work can only reveal the area of colonies rather than the number of colonies.

Due to the lack of handling of clustered colonies, Ferrari et al. [38] attempted to use a CNN to individually classify each colony cluster into a pre-defined cardinality class. The number of colonies in a plate image is obtained by adding up the cardinality of each individual colony cluster. In other words, bacterial colony counting is cast into an image classification task with a goal to assign each colony cluster to a specific class. Nevertheless, their method, which is referred to as MicrobiaNet in this thesis for simplicity, is dependent on the detection of each colony cluster. Moreover, class imbalance is not taken into account. The impact of high visual similarity of colony clusters on the counting performance also remains unknown. Furthermore, no studies have been found that explored the interpretability of MicrobiaNet. Although this study has yet to address class imbalance, it is the only study that directly addresses clustered colonies with an accuracy of 92.1% and F1 score of 0.81. This accuracy is the best so far to the best of my knowledge if objects are counted by cardinality classification. MicrobiaNet might become more reliable if the concerns of class imbalance and high visual similarity and its interpretability are further investigated.

Many generic object detection algorithms have been proposed to predict a rectangular bounding box around the object of interest and an objectness score. These detectors use neural networks to classify the presence of an object within different regions in an image. One of the disadvantages of these detectors is the time-consuming bounding box annotation. It is also computationally expensive to select a significant number of regions. To solve the problem of selecting many regions, Girshick et al. [48] proposed a method called R-CNN where only 2000 regions from an image are selected by the selective search algorithm [140], followed by a CNN to extract features which are fed into a SVM to classify the presence of the object in the proposed region. R-CNN also predicts four offset values to increase the precision of the bounding box. To improve R-CNN, Girshick [47] proposed a method called Fast R-CNN where a CNN is used to generate a feature map followed by selective search to propose regions. Fast R-CNN has been proven to be faster than R-CNN because the convolution operation is carried out once per image. Both R-CNN and Fast R-CNN use selective search, so they are slow and time-consuming [118]. Ren et al. [118] proposed a method called Faster R-CNN which uses a CNN to extract features that are fed into a Region Proposal Network (RPN) to propose regions. Faster R-CNN is faster than R-CNN and Fast R-CNN because the selective search algorithm is avoided.

Because R-CNN, Fast R-CNN and Faster R-CNN locate objects based on individual small regions in the image, these methods do not look at the image as one piece. This means

these methods may miss spatial information across different regions in the image. Redmon et al. [115] proposed a method called You Only Look Once (YOLO). This method splits an image into $S \times S$ grid where m bounding boxes will be selected from each grid cell. The network predicts a class probability and offset value for the bounding box, selected based on a threshold to locate the object. YOLO can achieve a fast real-time prediction for general-purpose object detection because its detection pipeline is a single network that is optimised end-to-end directly on detection performance [115]. But YOLO fails to detect small objects that appear in groups [116]. Hence, a second version of YOLO called YOLO9000 was proposed by Redmon and Farhadi [116] to overcome this issue by using pre-defined anchor boxes to improve bounding box proposal, as well as dividing an image into 13×13 grid cells which is better for small object detection. The third version of YOLO called YOLOv3 proposed by Redmon and Farhadi [117] further enables multi-label classification by using logistic classifiers for each class instead of softmax. Many researchers keep improving YOLO, creating YOLOv4 [11], YOLOv5 [65], YOLOv6 [80] and YOLOv7 [143]. However, like other generic object detectors, none of these methods addresses clustered objects.

2.1.4 Counting by hybrid approaches

Because of the inflexibility of pre-defined parameters, user intervention, and the lack of handling of clustered objects, some researchers combine traditional image processing methods and machine learning algorithms to count bacterial colonies. Ferrari et al. [37] used thresholding to detect segments in an image and designed a CNN to classify colony cardinality. Despite the classification outcome, the overall performance depended on the result of thresholding that might not be applicable to a different bacterial colony species. Similarly, thresholding combined with Bayes classification was used by Brugger et al. [13] to detect colonies. But colonies were assumed to be circular in their work, which could be problematic for colonies of a different shape.

Similar to machine learning based approaches, the combination of traditional image processing approaches and machine learning approaches has a limited effect of counting small and clustered colonies. These combined approaches also require a large amount of labelled data. Moreover, this type of approach cannot adapt to a different domain/category with ease because they are heavily dependent on training data.

2.2 Regression based counting

Regression based object counting approaches avoid the hard task of object detection, by directly predicting the object count from an image. These approaches have been used in crowd counting since the spatial information and the distribution of people are not needed for the final count. Davies et al. [32] used hand-coded features to build a regression model for estimating the crowd density as a supplement for crowd monitoring. Similarly, Chan and Vasconcelos [14] input 29 perspective-normalised features into a regression model to predict crowd count. Nevertheless, regression based counting approaches are only suitable when the spatial information of objects and the distribution of objects are not essential for the final count. This imposes a strong limitation on domain/category adaptations because the spatial information of objects is often essential to applications on a different domain/category. It is also unknown what is counted since the final output is merely a number. Additionally, regression based object counting approaches still require a large amount of labelled data since they are learning based algorithms.

2.3 Density map estimation based counting

Density map estimation based counting approaches avoid human intervention and the hard task of object detection. Approaches in this category learn to map an input image to a density map in which the sum of density values represents the object count. Compared with regression based approaches, the predicted density map can represent a general location of objects. The main drawback of density map estimation based approaches is the requirement of density map annotations, even though the density map annotations are significantly easier than bounding box annotations to create. Additionally, very little is currently known if this type of approach can address small objects. Similar to regression approaches, a different domain/category adaptation by density map estimation based counting requires retraining the algorithms. Despite that, these approaches are able to tackle clustered objects because the predicted density value can range from 0 to any positive number. Furthermore, these approaches can employ Generative Adversarial Networks (GANs) [52] to work with limited labelled data.

Crowd counting is one of the most common applications of this type of approach. Based on the survey conducted by Fan et al. [36], density map estimation based approaches for crowd counting can be categorised into several groups: multi-scale model, context-aware model, auxiliary-task model, dealing with the lack of labelled data model, domain adaptation model, perspective map model, attentional mechanism model, and network search model.

The design of multi-scale models for crowd counting typically involves the extraction of information on different scales. This is achieved by using different branches with different receptive fields in the network [12, 163], using feature pyramid structure [68] or using multiple scales of the input image [151]. Although multi-scale models can improve performance when counting crowds of different sizes, they become computationally expensive due to the increase in number of features. Moreover, the activation of smaller crowd diminishes with neural network's pooling layer.

The design of context-aware models takes advantage of local and global context information to enrich features. Amirgholipour et al. [3] proposed a network architecture that processes each part of an input image to predict local density. This method's main advantage is the positive effect of handling large-scale variations in crowd's size. However, this method is heavily influenced by the patch size and Gaussian function, producing a degraded performance when the size of the same object in an image varies dramatically. Liu et al. [88] designed a network to adaptively combine multi-scale contextual information to improve density prediction. Nevertheless, it is unknown if this method can address various sizes of the same object across the whole data set.

Auxiliary-task models simultaneously address one or more tasks related to crowd counting. According to Marsden et al. [94], their model is able to address crowd counting, violent behaviour detection and crowd density level classification simultaneously, even though additional ground truth labels are required. Similarly, Huang et al. [61] added the detection of pedestrian body parts as an auxiliary task to improve crowd counting. However, auxiliary-task models may fail to tackle objects with less contextual information. For example, a colony image, which only has colonies, cannot provide any additional information to support other auxiliary tasks.

Models that deal with the lack of labelled data and domain adaptations are often based on GANs. This is because GANs are capable of producing synthetic images or features that are from the same distribution as the real data [52]. For example, Olmschenk et al. [105] modified the traditional discriminator used in GANs to produce a value of expected crowd count and a flag to indicate if the input is real. The modified discriminator is more effective to capture features from data because it is a result of combining supervised regression and unsupervised classification. Similarly, Wang et al. [145] proposed a model called SSIM Embedding (SE) Cycle GAN to adapt to a different domain by translating synthetic crowd scenes to real scenes. Nevertheless, the pitfall of models that involve GANs is that GANs are notoriously difficult and unstable to train [126].

The design of the perspective map model considers the density map generation with additional perspective information. According to Shi et al. [130], such a design can address

perspective distortions when capturing crowd photos. This is because the perspective information can provide additional knowledge of the person scale change in an image. Similarly, Yan et al. [153] proposed a novel perspective-guided neural network to address intra-scene scale variations of pedestrians caused by perspective distortion. Despite the improved crowd counting performance, the use of a perspective map for other applications, such as microbiology where images are captured without perspective distortion, could be redundant since the camera is normally mounted in a fixed position.

Due to the success of transformers [141] in handling natural language processing tasks, many researchers [133, 162, 59, 86, 84, 53, 158, 18, 157, 44, 106, 72, 132] have attempted to employ an attention mechanism to improve object counting performance with some promising results. Sindagi and Patel [133] designed a spatial attention module that guides the model to focus more on relevant regions and foreground regions, as well as a global attention module that computes attention along the channel dimension. Pan et al. [106] introduced an attention map, which is the output of the first ten layers of VGG16 [131] followed by a convolutional layer and sigmoid activation, to guide the model to focus more on dense areas by multiplying the attention map by other network layer's output. Zhang et al. [157] proposed a self-attention mechanism to address noisy and inconsistent density prediction at pixel level by attending features that are spatially close to a human head, as well as pixels that are at a distance from a human head before fusing them together. Similarly, the attention mechanism implemented by Hossain et al. [59] focuses on global and local scales to address the scale variation in images. Despite the promising crowd counting performance from these attention based approaches, it is still unclear if they can adapt to a different domain/category.

Network architecture search models attempt to address the difficulty of designing a hand-crafted network architecture for crowd counting. The first attempt to automate the design of crowd counting models was made by Hu et al. [60]. They used neural architecture search to automatically search the optimal architecture from a multi-path encoder-decoder architecture. However, a drawback of their method is the expensive computation cost which is a day for training the model based on their results. Additionally, their method has yet to consider the more promising attention mechanism.

2.4 Few-shot learning based counting

Some researchers [112, 156] have attempted to address the lack of labelled data and domain/category adaptation for the counting task with few-shot object counting. Few-shot object counting aims to count the number of exemplar objects presented in a query image

where exemplar objects are described in only a few labelled support images.[1] Object classes are divided into base classes and novel classes which are used in training phase and test phase respectively. The base classes have no intersection with novel classes. In training phase, the model learns from the query image and a few support images with ground truth density map. In test phase, the model predicts a density map for a given query image with only a few support images by leveraging the knowledge gained from base classes. These few-shot counting approaches have a promising counting performance on generic objects. Nevertheless, they have yet to be applied to small objects.

Two few-shot counting models have shown great potential to address clustered objects, limited labelled data and domain/category adaptation. FamNet (Few-shot Adaptation and Matching Network), proposed by Ranjan et al. [112], uses pre-trained ResNet-50 [56] to extract features from query and support images, followed by a feature correlation layer and regression module to predict the final density map. Their domain/category adaptation is achieved by using provided exemplars and proposed combined adaptation loss to fine tune the regression module in FamNet. Despite its promising experimental results, FamNet has yet to consider small objects. Additionally, it is not end-to-end trainable, meaning the model is not fully optimised for the counting task. Moreover, the domain adaptation requires test time training which may not satisfy a busy laboratory. Two years after the publication of FamNet, SAFECount (Similarity-Aware Feature Enhancement block for object Counting) proposed by You et al. [156] integrates a transformer's attention mechanism to produce a clearer boundary between objects in the predicted density map. Another benefit of SAFECount is that it no longer requires test time training to adapt to a different domain due to the attention mechanism. Similar to FamNet, SAFECount has yet to be applied to small objects. Despite that, SAFECount has shown a great potential to address clustered objects, lack of labelled data and domain/category adaptation based on its experimental results.

2.5 Research gap and research questions

This chapter has reviewed a wide range of object counting methods based on their effect of addressing small object, clustered objects, limited labelled data, and domain/category adaptation. These counting methods are broadly categorised into detection based, regression based, density map estimation based, and few-shot learning based. The detection based approaches are further categorised into traditional image processing approaches, machine learning approaches and their hybrid approaches.

[1]The statement that few-shot object counting approaches address the lack of labelled data means these approaches only use **a few labelled support images.**

2.5.1 Research gap

Table 2.1 Comparison between different types of counting method.

Method category		Small object	Clustered objects	Limited labelled data	Domain/category adaptation
Detection based	Traditional image processing	limited [a]	limited	✓	✓
	Machine learning	limited	limited	×	×
	Hybrid approaches	limited	limited	×	×
	Regression based	unknown [b]	unknown	×	×
	Density map estimation based	unknown	✓	limited	×
	Few-shot learning based	unknown	✓	✓	✓

[a] Limited means the method has a limited effect of addressing the specified problem.
[b] Unknown means the method has not been applied.

An overview of all discussed counting methods in this chapter in relation to their effect of learning from limited labelled data to count small and clustered objects, and being readily generalisable to a different domain/category is presented in Table 2.1. In this table, a tick sign ✓ indicates the method is able to address the specified challenge; a cross sign × indicates the method is not able to address the specified challenge; a word "limited" denotes that the method has a limited effect of tackling the specified challenge; a word "unknown" means the method has not been applied to tackle the specified challenge.

As summarised in Table 2.1, **none of the existing counting methods is able to address small object, clustered objects, limited labelled data, and domain/category adaptation collectively.** Among these methods, detection based methods have a limited effect of detecting small and clustered objects. The output of regression based methods is merely a number that is extremely unintuitive, let alone the positive effect of dealing with limited labelled data and domain/category adaptation. Density map estimation based methods have the potential to address limited labelled data and domain/category adaptation if the design of GANs is sophisticated enough. However, it is unknown if this type of methods can address small object. Similar to density map estimation based methods, few-shot learning based methods are able to count clustered objects effectively because of the use density map. Few-shot learning based methods are the best option among the reviewed methods to tackle

clustered objects, limited labelled data and domain/category adaptation, even though it is unclear if this type of methods can count small objects.

2.5.2 Research questions

Aspects of cardinality classification

Before the investigation of few-shot learning, this thesis starts by investigating if MicrobiaNet can be integrated into Synoptics's in-house colony counting method. There are two reasons for choosing this algorithm. One is that it is the best-performing cardinality classification algorithm for colony counting to the best of my knowledge. The other is that it is the most compatible one to Synoptics's in-house colony counting method.

Motivated by Synoptics's business needs, this investigation does not aim to directly address the research gap identified in § 2.5.1. As review in § 2.1.3, this investigation attempts to address a more specific research gap instead. That is **no research has been found that explored MicrobiaNet's interpretability and the impact of class imbalance and high visual similarity between clustered objects on the counting performance. In addition, research to date has yet to investigate MicrobiaNet's ability to learn to count small and clustered objects from limited labelled data.** This investigation is presented in the first part (§ I) of this thesis in Chapter 4.

Although MicrobiaNet was evaluated by Ferrari et al. [38], their experimental results may not be reliable. This is because their experiment was only based on a one-time hold-out evaluation and they did not include detailed training and evaluation data for reproducibility. The investigation on MicrobiaNet focuses on the impact of class imbalance and high visual similarity between clustered colonies on the counting performance. This is because the former is a common problem for classification tasks and the latter is a well-known characteristic of colonies. Meanwhile, very little is known about MicrobiaNet's interpretability. Additionally, there has been no detailed investigation of the impact of class imbalance and high visual similarity across classes on the counting performance. Therefore, this research addresses the following research questions.

1. To what extent does MicrobiaNet address small and clustered colonies?
2. What insights into MicrobiaNet can be gained by studying its interpretability?
3. What has been the impact of class imbalance and high visual similarity on MicrobiaNet?
4. To what extent does MicrobiaNet perform with limited labelled data?

Aspects of density estimation

FamNet and SAFECount, which are based on few-shot learning, are thoroughly studied in this thesis because of their great potential to address the research gap identified in § 2.5.1. In this thesis, the research on few-shot learning takes place with FamNet which was published two years before SAFECount. The research on few-shot learning based counting is presented in the second part of this thesis in § II, including Chapter 5 to introduce proposed algorithms and Chapter 6 for experiments. Based on the research gap identified in § 2.5.1, this research addresses the following research questions.

1. To what extent does FamNet address small bacterial colonies?
2. How can the feature engineering in FamNet be modified to learn from limited labelled data to count small and clustered colonies, and be readily generalisable to a different domain or category? And what has been the effect of the modified feature engineering on addressing these problems?
3. To what extent does SAFECount address small bacterial colonies?
4. How can the feature engineering of SAFECount be transferred to FamNet or the modified FamNet to better adapt it to address small and clustered bacterial colonies, limited labelled data and cross domain/category generalisation since SAFECount is newer and superior to FamNet? And what has been the effect of the modified feature engineering on addressing these problems?

Chapter 3

Technical background and data description

Chapters 1 and 2 introduced the background and related work of object counting. In particular, the problems encountered while tackling small object size, clustered objects, limited labelled data, and domain/category adaptation. This chapter introduces the necessary technical background on deep learning, few-shot learning and object counting which are important prerequisites to understand the investigation in Chapter 4 and proposed solutions in Chapter 5. Additionally, this chapter discusses a neural network called Few-shot Adaptation and Matching Network (FamNet) [112] from which the proposed solutions in Chapter 5 are derived. Moreover, two data sets are described for their use in proposed solutions in Chapters 4 and 6.

3.1 Deep Learning

3.1.1 Neural networks

Neural networks are computing systems inspired by the human brain and designed to mimic the signal processing behaviour of biological neurons. They have been applied to recognise hidden patterns in data to address complex tasks in computer vision [77, 74, 131], natural language processing [27, 28], drug design [9, 23], bioinformatics [161, 5], etc. As illustrated in Fig. 3.1, the structure of a neural network consists of multiple layers of nodes with an input layer, one or more hidden layers and an output layer. Each node in a neural network layer connects to other nodes in the next network layer with a weight and threshold. The data, which mimics the signal in human brain, is computed with the weight and sent to the next network layer if the output of the node is above a specified threshold.

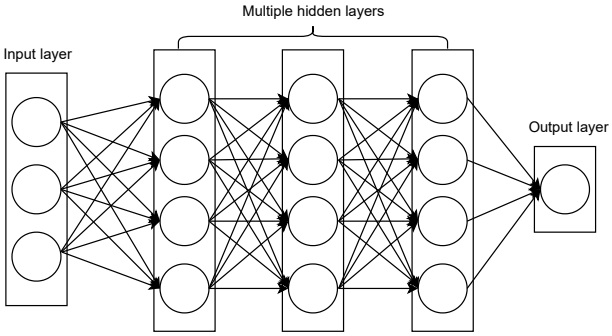


Fig. 3.1 Illustration of a neural network.

A neural network computes a map between input variables and output variables. The map is computed by layers of nodes, each being a non-linear function of its weighted inputs. A neural network is trained to approximate a mapping f^* from its input to output based on examples [51]. For a supervised learning task, such as classification where the input $\boldsymbol{x}$ and true output $\boldsymbol{y}$ are known prior to training, the real mapping between them is defined as:[1]

$$\boldsymbol{y} = f^*(\boldsymbol{x}) \tag{3.1}$$

Then, a neural network aims to learn a mapping:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta}) \approx \boldsymbol{y} \tag{3.2}$$

where $\hat{\boldsymbol{y}}$ is the predicted value of the output variable $\boldsymbol{y}$ based on the input variable $\boldsymbol{x}$ shown in Equation 3.1, and f is a function parameterised by parameters $\boldsymbol{\theta}$. The function f is a directed acyclic graph composed of L non-linear functions $f_{1:L}$. Equation 3.2 is thus expressed as:

$$\boldsymbol{h}^{(1)} = f_1(\boldsymbol{x}; \boldsymbol{\theta}^{(1)}) \tag{3.3}$$

$$\boldsymbol{h}^{(2)} = f_2(\boldsymbol{h}^{(1)}; \boldsymbol{\theta}^{(2)}) \tag{3.4}$$

$$\dots$$

$$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(L)} = f_L(\boldsymbol{h}^{(L\text{-}1)}; \boldsymbol{\theta}^{(L)}) \tag{3.5}$$

[1]The bold symbol $\boldsymbol{x}$ and $\boldsymbol{y}$ indicate the input and output are vectors. In the context of classification, the output vector $\boldsymbol{y}$ for a single input $\boldsymbol{x}$ is a one-hot vector transformed from a single value y. More details are given in § 3.1.3

where $\boldsymbol{\theta} = [\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \cdots, \boldsymbol{\theta}^{(L)}]$, and $\boldsymbol{h}^{(1)}$, $\boldsymbol{h}^{(2)}$, $\cdots$, $\boldsymbol{h}^{(L\text{-}1)}$ are intermediate representations called *hidden states*. Each function in the L non-linear functions constitutes a *layer* of the neural network of *depth L*. These layers in which hidden states are generated are called *hidden layers*. The final layer of the neural network is known as the *output layer*. The form and number of non-linear functions in a neural network is called its *architecture*. The name *Deep Learning* arises where the neural network has a depth L that is greater than 2. In contrast to the aforementioned acyclic connections, some neural networks, such as recurrent neural networks [122], have cyclic connections. However, they are not covered in this thesis.

The process of optimising $f(\boldsymbol{x};\boldsymbol{\theta})$ in Equation 3.2 to approximate $f^*(\boldsymbol{x})$ in Equation 3.1, by using a set of $\boldsymbol{x}$ and $\boldsymbol{y}$ pairs is called *training* or *learning*. The set of $\boldsymbol{x}$ and $\boldsymbol{y}$ forms the *training data set* $D_{\text{train}} = \{(\boldsymbol{X},\boldsymbol{Y})\} = \{(\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)})\}_i^m$ where m is the number of examples and i is an index of the ith example. The closeness of the approximation of $\boldsymbol{\theta}$ is measured by the difference between $\hat{\boldsymbol{Y}}$ and $\boldsymbol{Y}$ using a *cost function* $\mathcal{J}(\boldsymbol{\theta})$:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{\boldsymbol{y}}^{(i)},\boldsymbol{y}^{(i)}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\boldsymbol{y}^{(i)}) \tag{3.6}$$

where $\{(\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)})\}_i^m = D_{\text{train}}$, and the $\mathcal{L}(\hat{\boldsymbol{y}},\boldsymbol{y})$ is the *loss function*.[2] The cost function sometimes has some model complexity penalty added for regularisation, which is omitted here and discussed in § 3.1.4. The average loss over all examples, i.e. $\frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{\boldsymbol{y}}^{(i)},\boldsymbol{y}^{(i)})$, is called *empirical risk*.

After training, the process of producing the value of $\hat{\boldsymbol{y}}$ based on input variable $\boldsymbol{x}$ from the function f, parameterised by learned parameters $\hat{\boldsymbol{\theta}}$ when the real value of $\boldsymbol{y}$ is unknown, is called *inference*. To quantify the generalisation performance of a trained neural network model, it is evaluated on previously unseen data during training, known as the *test data set*, with a constraint that the unseen data is from the same distribution as the training data.

3.1.2 Neural network architectures

Different types of neural network architecture have been developed to tackle different tasks: feed-forward neural networks [97, 120], recurrent neural networks [122], convolutional neural networks [77], generative adversarial networks [52], transformer neural networks [141], etc. This section reviews feed-forward and convolutional networks that are relevant to this thesis. Feed-forward networks map an arbitrary input to an arbitrary output. Convolutional neural

[2]Loss function used to represent the quality of $\boldsymbol{\theta}$ calculated from a single pair of $\boldsymbol{x}$ and $\boldsymbol{y}$. The term loss function and cost function have become interchangeable nowadays. If not specified, loss function and cost function are also interchangeable in this thesis.

networks are similar to feed-forward networks except that they contain some additional convolutional layers to tackle images.

Feed-forward networks

Feed-forward networks, also often called *multi-layer perceptrons (MLPs)*, map an arbitrary input vector to an arbitrary output vector. A feed-forward network model can be defined by Equation 3.2 which is further expressed by Equation 3.3, 3.4 and 3.5.

The simplest form of a feed-forward network layer is the *fully connected layer* which is an affine transformation of the input followed by a non-linear activation function $\sigma(\boldsymbol{z})$. Suppose the feed-forward network input, output and hidden states are vector $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{h}^{(1:L)}$ respectively, the fully connected layer is defined as:

$$\boldsymbol{h}^{(l)} = \sigma({\boldsymbol{W}^{(l)}}^{\top}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}) \tag{3.7}$$

where $\boldsymbol{W}^{(l)}$ is a weight matrix and $\boldsymbol{b}^{(l)}$ is a bias vector. The weight matrix and bias vector constitute the fully connected layer parameters $\boldsymbol{\theta}^{(l)} = \{\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}\}$. The dimensions of $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ are dependent on $\boldsymbol{h}^{(l-1)}$ from the previous layer and the number of neurons specified by the designer. The purpose of adding the bias is to shift the decision boundary to maximise activations. Fig. 3.2 illustrates a simple feed-forward neural network with two fully connected layers as hidden layers when the input, hidden states and output are vectors. The fully connected layer is further detailed in Fig. 3.3.

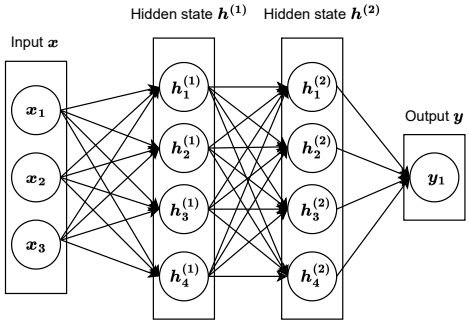


Fig. 3.2 Illustration of a feed-forward neural network mapping a length-3 input vector to a length-1 output vector with two hidden layers of size 4.

Example activation functions include sigmoid, tanh, Rectified Linear Unit (ReLU) [2], and Leaky Rectified Linear Unit (LReLU) [91]. They are respectively defined as:

$$\sigma_{\text{sigmoid}}(\boldsymbol{z}) = \frac{1}{1+e^{-\boldsymbol{z}}} \tag{3.8}$$

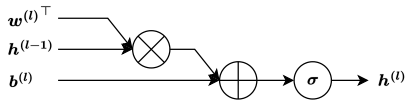


Fig. 3.3 Illustration of a fully connected layer, i.e. the hidden layers shown in Fig. 3.2.

$$\sigma_{\text{tanh}}(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3.9}$$

$$\sigma_{\text{ReLU}}(z) = \max(0, z) \tag{3.10}$$

$$\sigma_{\text{LReLU}}(z) = \max(\alpha z, z) \quad \text{where } 0 < \alpha < 1 \tag{3.11}$$

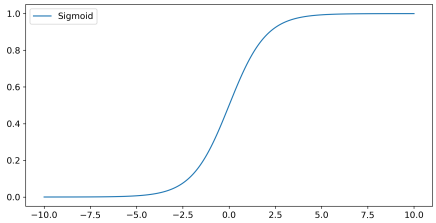
As illustrated in Fig. 3.4, the output of sigmoid (Fig. 3.4a) and tanh (Fig. 3.4b) becomes flat when the input is at extreme values. Because the network parameters $\boldsymbol{\theta}$ are learned using gradient descent (discussed in § 3.1.4), the saturation of output value when the input is at extreme values leads to *vanishing gradient*. In other words, the derivative of the activated value *w.r.t* the input value is close to 0 when the input value is large. This thus hinders gradient descent optimisation (discussed in § 3.1.4) and jeopardises the closeness of the approximation of $\boldsymbol{\theta}$. The vanishing gradient in sigmoid and tanh is tackled by ReLU (Fig. 3.4c) by increasing positive activation value linearly. LReLU (Fig. 3.4d) extends ReLU to protect a negative activation value by multiplying a constant value that is between 0 and 1. ReLU is used in this thesis for traditional neural networks as suggested by [51] because of its dominance in modern neural network research. LReLU is used for transformer based neural networks because it can better tackle long sequential data.

Convolutional neural networks

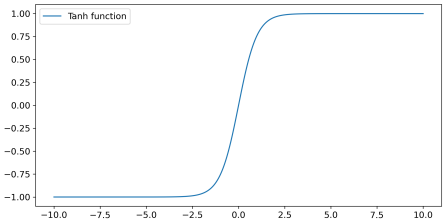
Convolutional neural networks (CNNs) are similar to traditional feed-forward neural networks except that a CNN contains additional convolutional layers. In particular, the first layer in a CNN is a convolutional layer designed to tackle digital colour images. A digital colour image of three channels is made of pixels which are further made of a combination of red, green and blue colours represented by numbers.[3] A channel in a digital colour image is a greyscale image of the same size as the colour image, made of only one of the red, green or blue colours, meaning the value of each pixel is a number used to represent the amount of light.

The convolution is performed by sliding a *kernel* K, also known as *filter*, on an image of dimension $C \times H \times W$ (channel $\times$ height $\times$ width) to produce a new output tensor in which

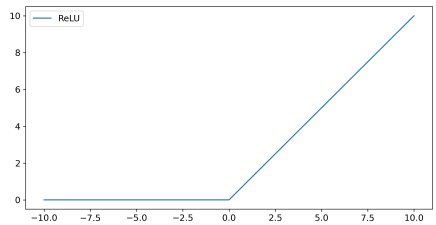
[3] A colour image of three channels is also called a three-dimensional image. Channel and dimension for a colour image are used interchangeably in this thesis because they have the same meaning.



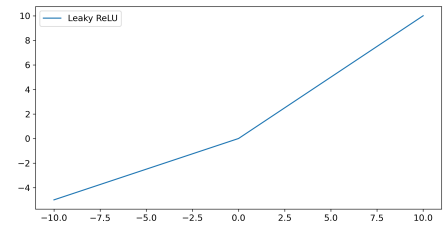
(a) Sigmoid.



(b) Tanh.



(c) ReLU.



(d) LReLU ($\alpha = 0.5$).

Fig. 3.4 Example activation functions where the input value ranges from -10 to 10.

each value is the sum of the element-wise multiplication. The convolution is useful to detect features, such as lines, curves, circles, rectangles, etc, in images. This is because a large convolution output value can indicate whenever the feature in the image matches the feature specified in the kernel, assuming that values in the image are positive numbers. Conversely, a small convolutional output value implies that the feature in the image cannot match the feature in the kernel.

The dimension of the kernel K is denoted as $C \times F \times F$ because it is often a square kernel and its channel number must match the input image channel number. The amount of movement of a kernel over the image is called *stride* which affects the output size. The amount of pixels added to the input image boarder to ensure the output size is identical to the input size after convolution is called *padding*. The padded pixel value could be 0 or 1 to represent black or white colour.[4] As recommended by Goodfellow et al. [51], this thesis uses 0 as the padded pixel value so that no additional information is added to the input data. The output tensor becomes a two-dimensional tensor of height $\frac{H-F+2p}{s} + 1$ and width $\frac{W-F+2p}{s} + 1$, where p is the padding and s is the stride. This output tensor followed by a non-linear function is often known as *feature map* or *activation map*.

[4]Modern image processing software uses 0 and 255 to represent black and white colours. However, multiplying large pixel values is computationally expensive when developing CNNs. Therefore, 0 and 1 are commonly used to represent black and white colours to reduce computation in the field of deep learning.

Fig. 3.5 illustrates a convolution of a $1 \times 3 \times 3$ kernel on a $1 \times 5 \times 5$ image with stride 1 without paddings for simplicity. In contrast, Fig. 3.6 shows a convolution with 0 padding added to preserve the original input size. Fig. 3.7 demonstrates an example of feature detection by convolution. The feature presented in the input image is a vertical edge, a sharp change in brightness across certain locations in the image vertically. Higher pixel values indicate brighter image areas, whereas lower pixel values indicate darker image areas. The visualised output after convolution presents a white area in the middle of the image which corresponds to the vertical edge in the visual input image.

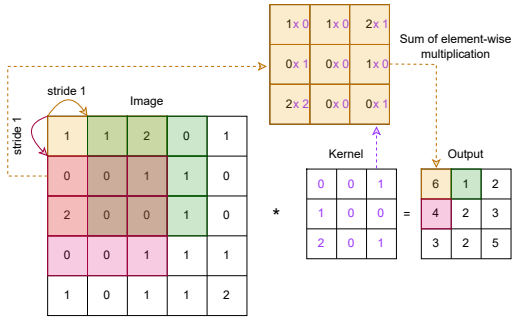


Fig. 3.5 Illustration of a convolution of a $1 \times 3 \times 3$ kernel on a $1 \times 5 \times 5$ image with stride 1 without paddings.

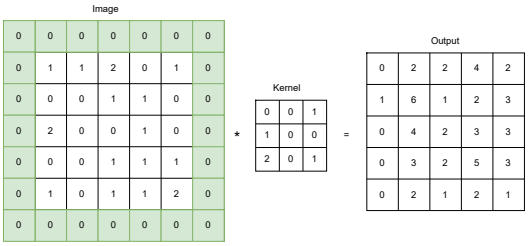


Fig. 3.6 Illustration of a convolution that is identical to Fig. 3.5 except the $1 \times 5 \times 5$ image is padded with zeros (green-coloured area) to ensure the output size is identical to the input size.

A convolutional layer often has many kernels applied on the input data so that different kernels can detect different patterns individually. Their outputs are stacked together to form a three-dimensional tensor whose additional dimension represents the number of kernels. Following the same style in Equation 3.7, the convolutional layer is defined as:

$$\boldsymbol{H}_i^{(l)} = \sigma\Big(\sum_{j=1}^{k^{(l-1)}} \boldsymbol{K}_i^{(l)} \boldsymbol{H}_j^{(l-1)} + \boldsymbol{b}_i^{(l)}\Big) \quad \forall i \in k^{(l)} \tag{3.12}$$

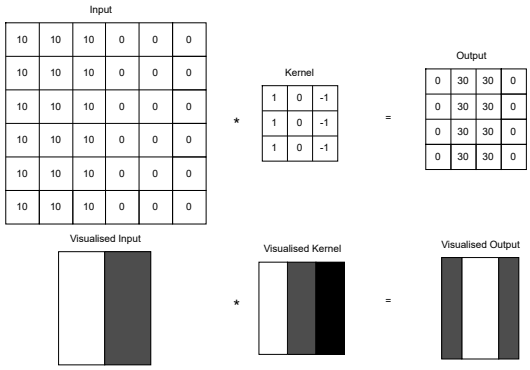


Fig. 3.7 Illustration of an edge detection by convolution.

where $\boldsymbol{K}^{(l)}$ is a set of $k^{(l)}$ kernels in the lth layer, $\boldsymbol{K}_i^{(l)}$ is the ith kernel in $\boldsymbol{K}^{(l)}$, $\boldsymbol{H}_i^{(l)}$ is a two-dimensional tensor which will be stacked with other two-dimensional tensors to form a three-dimensional tensor $\boldsymbol{H}^{(l)}$, and $\boldsymbol{b}^{(l)}$ is the bias added after convolution to shift the non-linear activation function to maximise activation.[5]

3.1.3 Training criteria

This section introduces an empirical risk minimisation approach called *Maximum Likelihood Estimation* to define the training criteria of classification and regression tasks. The *training criteria* are concrete forms of the loss function for classification and regression tasks. The prerequisite to understand this approach, namely the probability mass function, probability density function and likelihood function, are discussed before introducing the Maximum Likelihood Estimation.

Probability mass and density functions

The probability mass function describes the probability of a set of possible values of a discrete random variable occurring. Similarly, the probability density function describes the probability of a set of possible values of a continuous random variables occurring. The discrete and continuous variables are two types of random variable, where a random variable is a variable whose possible values are numerical outcomes of a random phenomenon. Discrete values are a countable number of distinct values and the continuous values are from an infinite number of possible values. Both the probability mass and probability density

[5]The bias can change the range of input value to a non-linear activation function to maximise activation. Suppose the non-linear activation is Sigmoid shown in Fig. 3.4a, with a bias of 10, the input value ranges from -10 to -7.5 is changed to the range of 0 to 2.5 to increase activation. This change is also known as the shift of a non-linear activation function.

functions approximately indicate the probability of a sample being a particular value. They are denoted as $P(y \mid \theta)$, where y is the value of a particular sample and θ is an already known variable describing a particular probability distribution.[6] When m samples $y_1, y_2, \ldots, y_m$ are drawn independently from the same distribution, the probability mass/density function of the sample values is the product of the value of the probability mass/density function for each individual sample value:[7]

$$P(y_1, y_2, \ldots, y_m \mid \theta) = \prod_{i=1}^{m} P(y_i \mid \theta) \tag{3.13}$$

Likelihood function

In contrast to the known variable θ describing the probability distribution in a probability mass or density function, the likelihood function considers the sample values $y_1, y_2, \ldots, y_m$ as fixed and the θ as an independent variable. It is denoted as $LH(y_1, y_2, \ldots, y_m \mid \theta)$. It is used when the sample values are known, because they are collected by users in the context of supervised learning, and the parameter θ is unknown. The likelihood function is statistically the same as the probability mass/density function:[8]

$$\underbrace{LH(y_1, y_2, \ldots, y_m \mid \theta)}_{\text{likelihood, function of } \theta} = \underbrace{P(y_1, y_2, \ldots, y_m \mid \theta)}_{\text{probability mass/density, function of } y_1, y_2, \ldots, y_m} \tag{3.14}$$

Maximum likelihood estimation

The maximum likelihood estimation is a method to estimate the best parameters in the chosen probability model that maximise the sample likelihood [29]. This method, also known as the *maximum likelihood estimator*, is based on the principle that the likelihood of a set of data belonging to the true data distribution is equivalent to the probability of obtaining that specific set of data given the chosen probability model. It is used in this thesis because of its two properties: consistency [70] and statistic efficiency [51]. The former ensures the bias induced by the maximum likelihood estimator diminishes as the number of examples increases. In other words, the maximum likelihood estimate of neural network parameters is asymptotically best as the number of examples approaches infinity. The latter allows the consistent maximum likelihood estimator to obtain a fixed level of generalisation error based

[6] This notation indicates y and θ are a particular example's value of their corresponding random variable. y and θ are not bold since they are two values. This notion will be adjusted to represent the parameter $\boldsymbol{\theta}$ (bold symbol) when explaining neural network training.

[7] Probability mass/density function is interpreted as a function of $y_1, y_2, \ldots, y_m$ with a fixed/known parameter θ.

[8] Likelihood function is interpreted as a function of θ with fixed/known $y_1, y_2, \ldots, y_m$.

on fewer examples, compared to other consistent estimators, such as the maximum spacing (MSP) method [21].

For a supervised learning task, consider m examples in a finite training data set $D_{\text{train}} = \{(\boldsymbol{X}, \boldsymbol{Y})\} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_i^m$ are sampled independently from the same true data generating distribution $p_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$.[9] In other words, examples in D_{train} are independent and identically distributed (i.i.d.). Meanwhile, consider a family of probability distributions over the same space as $p_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$ parameterised by $\boldsymbol{\theta}$ with $p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$,[10] such as probability distributions generated by neural networks. The training process aims to find the parameters in the neural network model that maximise the true probability $p_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$. The optimal parameters $\hat{\boldsymbol{\theta}}$ that maximise the likelihood of the observed data are thus defined as:

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\boldsymbol{Y} \mid \boldsymbol{X}; \boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \end{aligned} \tag{3.15}$$

Because the product of many probabilities can yield numerically unstable behaviour in floating-point numbers, Equation 3.15 is transformed into a sum of log-probabilities by taking the logarithm of the likelihood, which is shown in Equation 3.16. Equation 3.16 is further expressed as Equation 3.17, showing the expectation *w.r.t*. the *empirical distribution* $\hat{p}_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$ defined by the training data set D_{train} by dividing by m because the optimal parameters $\hat{\boldsymbol{\theta}}$ remain unchanged.

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{3.16}$$

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}) \sim \hat{p}_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})} \left[\log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) \right] \tag{3.17}$$

The maximum likelihood estimation is also viewed as a method to minimise the dissimilarity between the empirical distribution $\hat{p}_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$ defined by the training data set D_{train} and the model distribution $p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$. Based on the Kullback-Leibler (KL) divergence [75], which is a measure of how one probability distribution is different from a second probability distribution, the dissimilarity between the empirical distribution and model distribution is thus expressed as:

$$D_{KL}(\hat{p}_{\text{data}} \,||\, p_{\text{model}}) = \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}) \sim \hat{p}_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})} \left[\log \hat{p}_{\text{data}}(\boldsymbol{x}, \boldsymbol{y}) - \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) \right] \tag{3.18}$$

[9] $\boldsymbol{x}$ and $\boldsymbol{y}$ are paired because it is a supervised learning task.

[10] This notation indicates $\boldsymbol{x}$ and $\boldsymbol{y}$ are variables, whereas $\boldsymbol{\theta}$ are parameters.

where the term on the left, i.e. $\log \hat{p}_{\text{data}}(\boldsymbol{x}, \boldsymbol{y})$, is from the underlying data distribution rather than the neural network model. Hence, the training process to minimise the KL divergence is simplified as only minimising:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\hat{p}_{\text{data}}(\boldsymbol{x},\boldsymbol{y})} \left[-\log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})\right] \tag{3.19}$$

which is equivalent to the maximisation in Equation 3.17. Equation 3.19 is often called *negative log-likelihood (NLL)*. Conventionally, maximum likelihood estimation is implemented as Equation 3.19 because it is easier to minimise a function in practice.

Training models for classification

The maximum likelihood estimation is generalisable to classification tasks. The expected output $\boldsymbol{y}$ discussed in § 3.1.3 is a categorical variable representing which of the Q finite classes the input variable $\boldsymbol{x}$ belongs to. In practice, $\boldsymbol{y}$ is a length-Q vector where the yth value is 1 and other values are 0, which is known as a *one-hot vector*. For example, an output representing red among red, green and blue is represented as $[1,0,0]$. Similarly, a neural network $f(\boldsymbol{x};\boldsymbol{\theta})$ produces an output length-Q vector where each element represents the score for the corresponding category. For instance, suppose the neural network output is $[2.1,-1.5,0]$ for a sample, then 2.1 is the score for red class, -1.5 is the score for green class, and 0 is the score for blue class.

The raw output of a neural network $f(\boldsymbol{x};\boldsymbol{\theta})$ is followed by *softmax function* to produce a normalised probability distribution. The softmax function is defined as:

$$\sigma_{\text{softmax}}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{Q} e^{z_j}} \quad \text{for } i = 1,\ldots,Q \text{ and } \boldsymbol{z} = (z_1,\ldots,z_Q) \in \mathbb{R}^Q \tag{3.20}$$

where $\boldsymbol{z}$ is a length-Q raw vector output of the neural network $f(\boldsymbol{x};\boldsymbol{\theta})$, Q is the total number of categories in the classification task, and the sum of $\sigma_{\text{softmax}}(\boldsymbol{z})$ is 1. The maximum likelihood estimation shown in Equation 3.19 is thus transformed for a classification task into:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\hat{p}_{\text{data}}(\boldsymbol{x},\boldsymbol{y})} \left[-\boldsymbol{y}^T \log(\sigma_{\text{softmax}}(f(\boldsymbol{x};\boldsymbol{\theta})))\right] \tag{3.21}$$

where $\boldsymbol{y}$ is a one-hot encoded vector. The loss consisting of a negative log-likelihood in Equation 3.21 is known as *cross-entropy* between the empirical distribution defined by the training data set D_{train} and the classification model distribution $p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x};\boldsymbol{\theta})$.

Training models for regression

The maximum likelihood estimation is also readily generalised to regression tasks in which the goal is to predict an output value $\hat{y}$ that is as close to the expected output y as possible. The main difference between regression and classification is that regression output variables are continuous and the classification output variables are discrete. The maximum likelihood estimation for regression tasks is equivalent to minimising the *mean squared error (MSE)*:

$$\text{MSE}_{\text{train}} = \frac{1}{m}\sum_{i=1}^{m} || \hat{y}^{(i)} - y^{(i)} ||^2 \tag{3.22}$$

where m is the number of samples in the training data set D_{train}, $\hat{y}$ is the predicted output, and y is the expected output. This is because they both produce the same optimal parameters. The detailed explanation is given in the rest of this section.

Consider m examples in the finite training data set $D_{\text{train}} = \{(\boldsymbol{X}, Y)\} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_i^m$ independently drawn from the same true data distribution, meaning examples in D_{train} are i.i.d.. A regression model $f(\boldsymbol{x};\boldsymbol{\theta})$, such as a neural network, aims to fit y with some noise ε:

$$y = f(\boldsymbol{x};\boldsymbol{\theta}) + \varepsilon \tag{3.23}$$

$$\varepsilon = y - f(\boldsymbol{x};\boldsymbol{\theta}) \sim \mathcal{N}(0, \sigma^2) \tag{3.24}$$

where the noise variable ε is normally distributed with zero mean and constant variance σ^2 as shown in Equation 3.24. Based on Equation 3.23 and 3.24, y also follows the normal distribution centred on the neural network output, i.e. the Gaussian's mean is the neural network output, with the same constant variance σ^2:

$$y \sim \mathcal{N}(f(\boldsymbol{x};\boldsymbol{\theta}), \sigma^2) \tag{3.25}$$

The probability density function of y is thus defined as:

$$p(y \mid \boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\left(y - f(\boldsymbol{x};\boldsymbol{\theta})\right)^2\right) \tag{3.26}$$

Because training examples are i.i.d., the negative log-likelihood gives:

$$\begin{aligned} \text{NLL}(Y \mid \boldsymbol{X}; \boldsymbol{\theta}) &= -\sum_{i=1}^{m} \log p(y \mid \boldsymbol{x}; \boldsymbol{\theta}) \\ &= -\sum_{i=1}^{m} \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2} \left(y - f(\boldsymbol{x}; \boldsymbol{\theta})\right)^2 \right) \right) \\ &= \frac{m}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y - f(\boldsymbol{x}; \boldsymbol{\theta}))^2 \end{aligned} \tag{3.27}$$

Because only the last term in Equation 3.27 includes neural network parameters $\boldsymbol{\theta}$, minimising the negative log-likelihood is similar to minimising the mean square error in Equation 3.22.

3.1.4 Optimisation

This section introduces optimisation algorithms to meet training criteria discussed in § 3.1.3. Although there are no closed-form solutions to Equations 3.21 and 3.22, they are differentiable. In other words, they can be optimised by iterative gradient descent. This section discusses three variants of gradient descent, as well as the Adam optimiser. Additionally, topics of parameter initialisation, normalisation and regularisation are discussed.

Gradient descent optimisation

Gradient descent aims to minimise the cost function $\mathcal{J}(\boldsymbol{\theta})$ parametrised by neural network parameters $\boldsymbol{\theta}$ by updating the parameters in the opposite direction of the gradient of the cost function $\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$ *w.r.t.* the parameters obtained by the back-propagation algorithm [122]. The pace of updating the parameters is determined by the *learning rate* η, which multiplies the gradient, and the amount of data used to compute the gradient [121]. Gradient descent requires users to pre-determine a termination condition called *epoch*. A full pass over the entire training data set for computing the gradient and updating parameters is known as one epoch. Batch gradient descent, stochastic gradient descent and mini-batch gradient are three variants of gradient descent based on the amount of data used to compute the gradient.

Batch gradient descent, also known as *vanilla gradient descent*, calculates the gradient of the cost function $\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$ *w.r.t.* the parameters based on the entire training data set. Batch gradient descent training can be slow because the gradient, which is based on the entire training data set, is only used to perform one update. Likewise, it may cause the gradient to be intractable if the entire training data set is too large to fit in computer memory. Despite that, batch gradient descent is guaranteed to reach the global optimum for convex error surfaces

and a local optimum for non-convex surfaces. Algorithm 1 shows how batch gradient descent searches for the optimal parameters, where the loss function $\mathcal{L}$ is used to replace the cost function to emphasise data rather than parameters.

Algorithm 1: Batch gradient descent.

Input : Learning rate η
Epoch number t
Training data set $D_{\text{train}} = \{(\boldsymbol{X}, \boldsymbol{Y})\} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_i^m$
Neural network model f
Loss function $\mathcal{L}$

Output : Optimal parameters $\boldsymbol{\theta}$

$\boldsymbol{\theta} \leftarrow$ initialise;
while $t \neq 0$ **do**
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \times \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} \mathcal{L}(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$;
  $t \leftarrow t - 1$;
end

Stochastic gradient descent (SGD), in contrast, computes the gradient based on each training example. However, the objective function can fluctuate severely because the update is performed based on each training example. Compared to the batch gradient descent, SGD can converge faster and potentially escape local optima. SGD is detailed in Algorithm 2.

Algorithm 2: Stochastic gradient descent.

Input : Learning rate η
Epoch number t
Training data set $D_{\text{train}} = \{(\boldsymbol{X}, \boldsymbol{Y})\} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_i^m$
Neural network model f
Loss function $\mathcal{L}$

Output : Optimal parameters $\boldsymbol{\theta}$

$\boldsymbol{\theta} \leftarrow$ initialise;
while $t \neq 0$ **do**
  while $m \neq 0$ **do**
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \times \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{x}^{(m)}; \boldsymbol{\theta}), \boldsymbol{y}^{(m)})$;
    $m \leftarrow m - 1$;
  end
  $t \leftarrow t - 1$;
end

Mini-batch gradient descent combines the advantages of gradient descent and SGD by updating parameters based on one mini-batch at a time.[11] It often has a more stable convergence because the update frequency is reduced compared to SGD. The mini-batch is often known as *batch size* which is typically between 16 and 256 or larger depending on the data set size.[12] However, this method may introduce unnecessary bias if similar samples are always in a batch. Therefore, the training data set is shuffled beforehand. The data shuffling also introduces regularisation because the noise added to the gradient can avoid the training data over-fitting the model and getting stuck in a local minimum. However, a proper choice of learning rate is required because it may lead to slow convergence or a heavy fluctuation of cost function. Additionally, mini-batch gradient descent struggles to escape saddle points surrounded by a plateau of the same error [31]. Algorithm 3 details the mini-batch gradient descent. In practice, the remaining data can be treated as a full batch if the training set size is not divisible by the batch size.

Algorithm 3: Mini-batch gradient descent.

Input : Learning rate η
Epoch number t
Batch size b
Training data set $D_{\text{train}} = \{(\boldsymbol{X}, \boldsymbol{Y})\} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_i^m$
Neural network model f
Loss function $\mathcal{L}$

Output : Optimal parameters $\boldsymbol{\theta}$

$\boldsymbol{\theta} \leftarrow$ initialise;
while $t \neq 0$ **do**
  $D_{\text{train}} \leftarrow \texttt{shuffle}(D_{\text{train}})$;
  $m \leftarrow \texttt{getSize}(D_{\text{train}})$;
  while $m > 0$ **do**
    if $m - b + 1 \leq 0$ **then**
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \times \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} \mathcal{L}(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$;
    else
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \times \frac{1}{b} \nabla_{\boldsymbol{\theta}} \sum_{i=m-b+1}^{m} \mathcal{L}(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$;
    end
    $m \leftarrow m - b$;
  end
  $t \leftarrow t - 1$;
end

[11] In practice, mini-batch gradient descent is often referred to as SGD with a batch size that is more than 1.
[12] The batch size is often set as powers of 2 because of data partitions on model GPUs.

To escape the saddle points surrounded by a plateau of the same error, mini-batch gradient can be accelerated by *momentum* which adds a fraction of the previous gradient to the current gradient [109]. The mini-batch gradient with momentum is further improved by *Adaptive Moment Estimation (Adam)* which adaptively adjusts the first and second moment of the gradient and learning rate for updating individual parameters [71]. More specifically, Adam stores an exponential weighting of previous gradients as an estimate of the *first moment* of the gradient, and another exponential decaying of previous squared gradients as an estimate of the *second moment* of the gradient. The *n*th-order moment is defined as the expected value of the variable, i.e. gradient, to the power of n. In other words, the first moment is the mean of the gradient, and the second moment is the uncentred variance of the gradient. They are defined in Equation 3.28 and 3.29 respectively.

$$s_t = \beta_1 s_{t-1} + (1-\beta_1)\boldsymbol{g}_t \tag{3.28}$$

$$r_t = \beta_2 r_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2 \tag{3.29}$$

where t indicates current training step, $t-1$ indicates the previous training step, β is the decay rate, $\boldsymbol{g}$ is the gradient, and $\boldsymbol{g}^2$ indicates the element-wise square $\boldsymbol{g} \odot \boldsymbol{g}$. However, the first moment and second moment are biased towards zero as observed by Adam's authors. Therefore, they are corrected before being incorporated into the parameter update using Equation 3.30 and 3.31. Finally, the Adam update is produced with Equation 3.32 where η is the learning rate, and ε is a small number added to the denominator to prevent the denominator from being zero for numerical stability. The detailed implementation of Adam is shown in Algorithm 4. Adam is used in this thesis due to its state-of-the-art efficiency and popularity in practice [121].

$$\hat{s_t} = \frac{s_t}{1-\beta_1^t} \tag{3.30}$$

$$\hat{r_t} = \frac{r_t}{1-\beta_2^t} \tag{3.31}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \times \frac{\hat{s}}{\sqrt{\hat{r}}+\varepsilon} \tag{3.32}$$

Parameter initialisation

It is necessary to assign initial values to the parameters to be optimised by any gradient descent method. These initial values should be asymmetric because gradient descent requires some asymmetry in the gradient to begin search effectively. Without the asymmetry, hidden

Algorithm 4: Adam.

Input : Learning rate η
Epoch number t
Batch size b
Constant ε ▷`suggest` 10^{-8} `for numerical stability`
Decay rate β_1 and β_2 ▷`suggest 0.9 and 0.999`
Training data set $D_{\text{train}} = \{(\boldsymbol{X}, \boldsymbol{Y})\} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_i^m$
Neural network model f
Loss function $\mathcal{L}$

Output : Optimal parameters $\boldsymbol{\theta}$

$\boldsymbol{\theta} \leftarrow$ initialise;
while $t \neq 0$ **do**
  $D_{\text{train}} \leftarrow \texttt{shuffle}(D_{\text{train}})$;
  $m \leftarrow \texttt{getSize}(D_{\text{train}})$;
  $s \leftarrow 0$;
  $r \leftarrow 0$;
  $l \leftarrow 0$;
  while $m > 0$ **do**
    if $m - b + 1 \leq 0$ **then**
      $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} \mathcal{L}(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$;
    else
      $\boldsymbol{g} \leftarrow \frac{1}{b} \nabla_{\boldsymbol{\theta}} \sum_{i=m-b+1}^{m} \mathcal{L}(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$;
    end
    $s \leftarrow \beta_1 \times s + (1 - \beta_1) \times \boldsymbol{g}$;
    $r \leftarrow \beta_2 \times r + (1 - \beta_2) \times \boldsymbol{g}^2$;
    $\hat{s} \leftarrow \frac{s}{1 - \beta_1^l}$;
    $\hat{r} \leftarrow \frac{r}{1 - \beta_2^l}$;
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \times \frac{\hat{s}}{\sqrt{\hat{r}} + \varepsilon}$;
    $l \leftarrow l + 1$;
    $m \leftarrow m - b$;
  end
  $t \leftarrow t - 1$;
end

layers tend to learn the same function which jeopardises the efficiency of neural networks. These initial values also need to prevent the gradient of the cost function $\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$ *w.r.t.* the parameters obtained by the back-propagation algorithm from being too small or too large. This is because a small or large gradient will hinder gradient descent methods. Glorot and Bengio [50] proposed a properly scaled uniform distribution for initial parameter values.

However, according to He et al. [56], this method is based on the assumption that the activation function is only linear. A normalised version proposed by Glorot and Bengio [50] still inherits the limited theoretical assumption. He et al. [56] investigated the variance of response signals in each neural network layer and defined a condition to stop reducing or magnifying the magnitudes of input signal exponentially:

$$\frac{1}{2} n_l \mathrm{Var}[w_l] = 1 \tag{3.33}$$

where n_l is the number of connections of a response in the lth layer, and w_l is the weights of parameters in the lth layer. Equation 3.33 leads to the initialisation:

$$w_l \sim \mathscr{N}(0, \frac{2}{n_l}) \tag{3.34}$$

where the values of weights are drawn from a zero-mean Gaussian distribution with standard deviation of $\sqrt{\frac{2}{n_l}}$. Meanwhile, the values of biases are initialised at 0. The initialisation method proposed by He et al. [56], which is often known as *He initialisation*, is used in this thesis because of its state-of-the-art efficiency and the design choice of ReLU.

Another way to initialise parameters is to reuse trained parameters from a similar network on a related task. This is often known as *transfer learning*. This method allows rapid training progress or an improved performance because the knowledge, i.e. parameters, learned from another task is exploited to improve generalisation in another task [104]. This method is also used in this thesis to improve training performance for some tasks.

Normalisation

Neural network training may be unstable because when updating the parameters of a layer changes the distribution of inputs to subsequent layers. Consequentially, the change of data distribution in deeper layers slows down the training process. This is known as *internal covariate shift* according to Ioffe and Szegedy [62]. This problem is mitigated by Ioffe and Szegedy [62] by adding a *batch normalisation* layer before non-linear activation. Batch normalisation ensures all activation has a zero mean and unit variance for each mini-batch so that the training process is stabilised. This is achieved by computing the mean and variance of a mini-batch of input data:

$$\boldsymbol{Y} = \frac{\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}]}{\sqrt{\mathrm{Var}[\boldsymbol{X}] + \varepsilon}} \times \gamma + \beta \tag{3.35}$$

where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are a mini-batch of input data and output data, ε is 10^{-5} for numerical stability, γ and β are learnable parameters initialised at 1 and 0. During inference, the mean and variance are calculated from each mini-batch of the training data:

$$\begin{aligned} \boldsymbol{y} &= \frac{\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}]}{\sqrt{\mathrm{Var}[\boldsymbol{x}] + \varepsilon}} \times \gamma + \beta \\ &= \frac{\gamma}{\sqrt{\mathrm{Var}[\boldsymbol{x}] + \varepsilon}} \times \boldsymbol{x} + (\beta - \frac{\gamma \mathbb{E}[\boldsymbol{x}]}{\sqrt{\mathrm{Var}[\boldsymbol{x}] + \varepsilon}}) \end{aligned} \tag{3.36}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are a single input and output data, γ and β are learned parameters from training, $\mathbb{E}[\boldsymbol{x}]$ and $\mathrm{Var}[\boldsymbol{x}]$ is the mean of m mini-batch training data's mean and variance calculated by Equation 3.37 and 3.38 respectively.

$$\mathbb{E}[\boldsymbol{x}] = \mathbb{E}[\mathbb{E}[\boldsymbol{X}]] \tag{3.37}$$

$$\mathrm{Var}[\boldsymbol{x}] = \frac{m}{m-1} \mathbb{E}[\mathrm{Var}[\boldsymbol{X}]] \tag{3.38}$$

Since the publication of batch normalisation [62], many methods derived from it have been proposed to further stabilise training. For example, weight normalisation [125], layer normalisation [7], group normalisation [149], and weight standardisation [110]. Among these normalisation methods, layer normalisation normalises input values in all neurons in the same layer for each data sample. This design makes it a default choice for transformer based neural network because it can better tackle long sequence data, be used with any batch number and be used in parallel. Additionally, batch normalisation is used in this thesis for traditional neural networks due to its dominance and popularity in practice.

Regularisation

A common problem encountered when developing neural network models is *over-fitting*, meaning the trained network performs well on its training data set but fails to generalise to an unseen test data set. This problem can be alleviated by regularisation which discourages the neural network from becoming too complex or having large parameter values. Meanwhile, regularisation helps in controlling the neural network's ability to fit noise in the training data. Common regularisation methods are weight decay, early stopping, dropout [135], pooling and data augmentation.

Weight decay is a widely used method to regularise neural networks by adding a L_p norm penalty to parameters. It is added to the cost function shown in Equation 3.6 and thus

expressed as:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{\boldsymbol{y}}^{(i)},\boldsymbol{y}^{(i)}) + \lambda \mid\mid \boldsymbol{\theta} \mid\mid_p = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\boldsymbol{y}^{(i)}) + \lambda \mid\mid \boldsymbol{\theta} \mid\mid_p \tag{3.39}$$

where λ is a hyper-parameter, and p is the L_p norm.[13] Adding weight decay regularisation can be seen as the *Maximum a-posterior (MAP)* estimation which is similar to maximum likelihood estimation but with prior knowledge added. Similar to Equation 3.15, MAP is defined as Equation 3.40 based on Bayesian framework.

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\text{MAP}} &= \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\boldsymbol{Y} \mid \boldsymbol{X};\boldsymbol{\theta}) p_{\text{parameter}}(\boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)};\boldsymbol{\theta}) p_{\text{parameter}}(\boldsymbol{\theta})\end{aligned} \tag{3.40}$$

Following the same transform in maximum likelihood estimation shown in Equation 3.19, Equation 3.40 is transformed into:

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\text{MAP}} &= \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\hat{p}_{\text{data}}(\boldsymbol{x},\boldsymbol{y})}\left[-\log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x};\boldsymbol{\theta}) p_{\text{parameter}}(\boldsymbol{\theta})\right] \\ &= \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\hat{p}_{\text{data}}(\boldsymbol{x},\boldsymbol{y})}\left[-(\log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x};\boldsymbol{\theta}) + \log p_{\text{parameter}}(\boldsymbol{\theta}))\right]\end{aligned} \tag{3.41}$$

When L_2 norm is used in Equation 3.39, $p_{\text{parameter}}(\boldsymbol{\theta})$ in Equation 3.41 can be treated as a zero-mean normal distribution. Hence:

$$\log p_{\text{parameter}}(\boldsymbol{\theta}) \propto \mid\mid \boldsymbol{\theta} \mid\mid_2^2 \tag{3.42}$$

L_2 norm weight decay regularisation is used in this thesis because of the assumption that the parameters $\boldsymbol{\theta}$ are zero-mean normally distributed.

Early stopping is another popular regularisation method. It simply terminates training before the neural network has over-fit training data. This method requires an additional validation data set, assuming the training, validation and test data set are sampled from the same distribution. During training, the performance on the validation data set is monitored so that the training can be terminated early if the generalisation error on the validation data set starts increasing. Because of the assumption, performance on the validation data set can be an estimate for the performance on the test data set. Therefore, over-fitting can be prevented.

[13] Hyper-parameters of a neural network include variables that determine the network's architecture and variables that determine how the network is trained. They are set by users before training. On the contrary, parameters of a network are required for the network to make predictions. They are learned from data during training.

Dropout is another way to regularise networks by randomly resetting neural network activation to zero [135]. It works differently in the training and inference phases. During training phase, the activation of every hidden layer are randomly reset to 0. This is achieved by multiplying the activation with a binary mask vector whose element is either 0 or 1. The exact value of an element is sampled from a Bernoulli distribution.[14] The value of an element being 1 is sampled with a probability of p from the Bernoulli distribution:

$$\boldsymbol{h}^{(L)} = f_L(\boldsymbol{h}^{(L\text{-}1)}; \boldsymbol{\theta}^{(L)})\boldsymbol{b}, \quad \boldsymbol{b} \sim \text{Bernoulli}(p) \tag{3.43}$$

where $f_L(\boldsymbol{h}^{(L\text{-}1)})$ is the activation in the Lth layer shown in Equation 3.5 before applying the dropout regularisation. In other words, the expected value of an element x in the $\boldsymbol{h}$ becomes $px + (1-p)0$ with dropout during training. During inference, x is multiplied by p to keep the same expected output so that all activation is preserved:

$$\boldsymbol{h}^{(L)} = pf_L(\boldsymbol{h}^{(L\text{-}1)}; \boldsymbol{\theta}^{(L)}) \tag{3.44}$$

In practice, the product of p during inference is moved to training phase with an inverted dropout $\frac{1}{p}$ so that no changes need to be made during inference. The completed dropout method for training and inference is shown in Equation 3.45 and Equation 3.46 respectively.

$$\boldsymbol{h}^{(L)} = \frac{1}{p} f_L(\boldsymbol{h}^{(L\text{-}1)}; \boldsymbol{\theta}^{(L)})\boldsymbol{b}, \quad \boldsymbol{b} \sim \text{Bernoulli}(p) \tag{3.45}$$

$$\boldsymbol{h}^{(L)} = f_L(\boldsymbol{h}^{(L\text{-}1)}; \boldsymbol{\theta}^{(L)}) \tag{3.46}$$

Similar to dropout, the activation in convolutional layers can be replaced with the maximum or average value in a filter. They are known as *max pooling* and *average pooling* respectively. This pooling method can also reduce the spatial size of the representation to reduce the amount of parameters and computation in neural networks. Therefore, over-fitting is alleviated. Pooling is achieved by sliding a filter of size $F \times F$ with a stride of s over the depth slice of output tensor from the convolutional layer along height and width dimensions. Figure 3.8 illustrates a max pooling with a filter size of 2×2 and stride 2 on a 4×4 single depth slice to produce a 2×2 output. This thesis uses max pooling to exploit the strongest activation inside the neural network.

Data augmentation is a simple regularisation method, aiming to increase data set size to improve generalisation performance. It is also equivalent to adding prior knowledge. For example, it may be assumed that adding rotated images to the model will not change the class

[14] Bernoulli distribution is a discrete distribution with only two possible outcomes for a random variable.

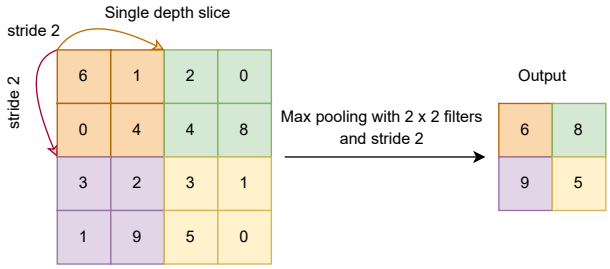


Fig. 3.8 Illustration of a max pooling with a filter size of 2×2 and stride 2 on a 4×4 single depth slice to produce a 2×2 output.

label. However, data augmentation should be carefully designed because it may completely change output. For example, rotating a digit 6 image clockwise 180 degrees will change its class label to digit 9. Data augmentation may also increase the difficulty of neural network design due to the increase of data set size. To reduce the difficulty of neural network design, data augmentation is not used in this thesis.

3.1.5 Summary of design decisions

This section has introduced various neural networks, neural network architectures, training criteria and optimisation methods. A summary of design decisions in relation to the network and optimisation is presented in Table 3.1. The detailed reasons for these decisions are provided when introducing the proposed algorithms in § 4.1.1, § 5.1, and § 5.2 respectively.

Table 3.1 Summary of design decisions.

	Method category	Detailed method	Use case
Network	Activation function	ReLU	§ 4.1.1, § 5.1
		LReLU	§ 5.2
	Conv layer	0 padding	§ 4.1.1, § 5.1, § 5.2
		1 stride	§ 4.1.1, § 5.1, § 5.2
Optimisation	Training criteria	Softmax	§ 4.1.1
		MSE	§ 5.1,§ 5.2
	Optimisation algorithm	Adam	§ 4.1.1, § 5.1, § 5.2
	Weight initialisation	He initialization	§ 4.1.1, § 5.2
		Transfer learning	§ 5.1
	Normalisation	Batch normalisation	§ 4.1.1, § 5.1, § 5.2
		Layer normalisation	§ 5.2
	Regularisation	Weight decay	§ 4.1.1
		Early stopping	§ 4.1.1, § 5.1, § 5.2
		Dropout	§ 4.1.1
		Max pooling	§ 4.1.1

3.2 Few-shot learning

Few-shot learning [39] is a variant of *meta-learning* [127], which is about learning to learn [137] and often interpreted as the process of improving a learning algorithm over many tasks, when the number of examples is very limited. The idea of few-shot learning is to find ways to build models to make accurate predictions given a limited number of examples. It is used in this thesis because of the lack of examples in some problems. Few-shot learning is inspired by the fact that humans can quickly learn to solve a problem from just a few examples. For example, given only a few photos of a stranger, humans can easily identify the same stranger in other photos. In other words, few-shot learning aims to build models that can learn to learn from very few examples. This section also covers meta-learning which is a prerequisite to understand few-shot learning.

3.2.1 Formalising meta-learning

Meta-learning [127] is a sub-field of machine learning with a focus on learning from previous experience of learning tasks to improve future learning performance. An important characteristic of conventional deep learning is that the 'how to learn', such as the choice of the initial parameters [39], optimisation strategy [114], or entire learning model [98], is problem or task dependent specified by its data. In other words, the learning must be performed from scratch for every problem or task. On the contrary, meta-learning aims to improve algorithm performance by learning the learning algorithm itself rather than assuming the 'how to learn' is pre-specified and fixed [58]. From a perspective of task distribution, meta-learning aims to learn a general purpose learning algorithm that is generalisable across different tasks so that every new task is learned better than the last.

In conventional supervised learning, given a task $\mathcal{T}$ with a data set $D = \{(\boldsymbol{X}, \boldsymbol{Y})\} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_i^m$ which is split into training data set and test data set $D = \{D_{\text{train}}, D_{\text{test}}\}$, a predictive model $f(\boldsymbol{x}; \boldsymbol{\theta})$ parameterised at $\boldsymbol{\theta}$ aims to approximate the true $\boldsymbol{y}$:

$$\boldsymbol{y} \approx f(\boldsymbol{x}; \hat{\boldsymbol{\theta}}), \quad (\boldsymbol{x}, \boldsymbol{y}) \in D_{\text{test}} \tag{3.47}$$

where

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}), \quad \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_i^k = D_{\text{train}} \tag{3.48}$$

In meta-learning, consider a task distribution $p(\mathcal{T})$, a meta-learner learns from a set of training tasks drawn independently from the same task distribution $p(\mathcal{T})$ and is evaluated on a set of test tasks that are also i.i.d. and unseen during the training process. These

two processes are called *meta-training* and *meta-test* to distinguish them from conventional deep learning. Each task $p(\mathcal{T})$ has its own data set $D_{\text{meta-task}}$ which is also split into *support data set* and *query data set*: $D_{\text{meta-task}} = \{D_{\text{support}}, D_{\text{query}}\} \rightleftharpoons \mathcal{T}$.[15] The meta-training task set and meta-test task set are denoted as $\mathcal{T}_{\text{meta-train}} = \{\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(k)}\}$ and $\mathcal{T}_{\text{meta-test}} = \{\mathcal{T}^{(m-k)}, \ldots, \mathcal{T}^{(m)}\}$. The meta-learner is defined as:

$$\hat{\boldsymbol{y}} = g(D_{\text{support}}, \boldsymbol{x}; \boldsymbol{\omega}) \tag{3.49}$$

where $\{D_{\text{support}}, D_{\text{query}}\} \rightleftharpoons \mathcal{T}$, $\hat{\boldsymbol{y}}$ is the predicted value for $\boldsymbol{y}$ given $(\boldsymbol{x}, \boldsymbol{y}) \in D_{\text{query}}$, and $\boldsymbol{\omega}$ specifies 'how to learn'. Equation 3.49 can be interpreted as the meta-learner g uses the previous experience of 'how to learn' $\boldsymbol{\omega}$ to learn a new task $\mathcal{T}$ that has its own training data set D_{support} and test data set D_{query}. The meta-learner is thus optimised as:

$$\boldsymbol{y} \approx g(D^{(i)}_{\text{support}}, \boldsymbol{x}; \hat{\boldsymbol{\omega}}) \tag{3.50}$$

where $(\boldsymbol{x}, \boldsymbol{y}) \in D^{(i)}_{\text{query}}$, $\{D^{(i)}_{\text{support}}, D^{(i)}_{\text{query}}\} \rightleftharpoons \mathcal{T}^{(i)} \in \mathcal{T}_{\text{meta-test}}$, and $\hat{\boldsymbol{\omega}}$ is obtained from Equation 3.51.

$$\hat{\boldsymbol{\omega}} = \arg\min_{\boldsymbol{\omega}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(g(D^{(i)}_{\text{support}}, \boldsymbol{x}; \boldsymbol{\omega}), \boldsymbol{y}) \tag{3.51}$$

where $(\boldsymbol{x}, \boldsymbol{y}) \in D^{(i)}_{\text{query}}$, and $\{D^{(i)}_{\text{support}}, D^{(i)}_{\text{query}}\} \rightleftharpoons \mathcal{T}^{(i)} \in \mathcal{T}_{\text{meta-train}}$. Fig. 3.9 illustrates how meta-learning could tackle binary classification tasks in different domains.

3.2.2 Few-shot learning and few-shot object counting

Few-shot learning is a special case of meta-learning when the number of examples in the support and query sets are very limited, ranging from 0 to 3. It has become an active research field because the knowledge of 'how to learn' from other tasks is particularly useful when the number of examples in the support and query sets is very limited [39]. This thesis uses few-shot learning because the number of sample images available to solve some problems is also very limited.

Few-shot object counting is a term to indicate that few-shot learning is used for counting tasks. It aims to count the number of exemplar objects presented in a query image where the exemplar objects are described in only a few support images. As illustrated in Figure 3.10, object classes are divided into base classes $\mathcal{C}_b$ and novel classes $\mathcal{C}_n$ which are used in the

[15] Support data set is the equivalent training data set in a traditional supervised learning. Query data set is the equivalent test data set in a traditional supervised learning. The sign $\rightleftharpoons$ indicates the data set $D_{\text{meta-task}}$ and task $\mathcal{T}$ are mapped to each other.

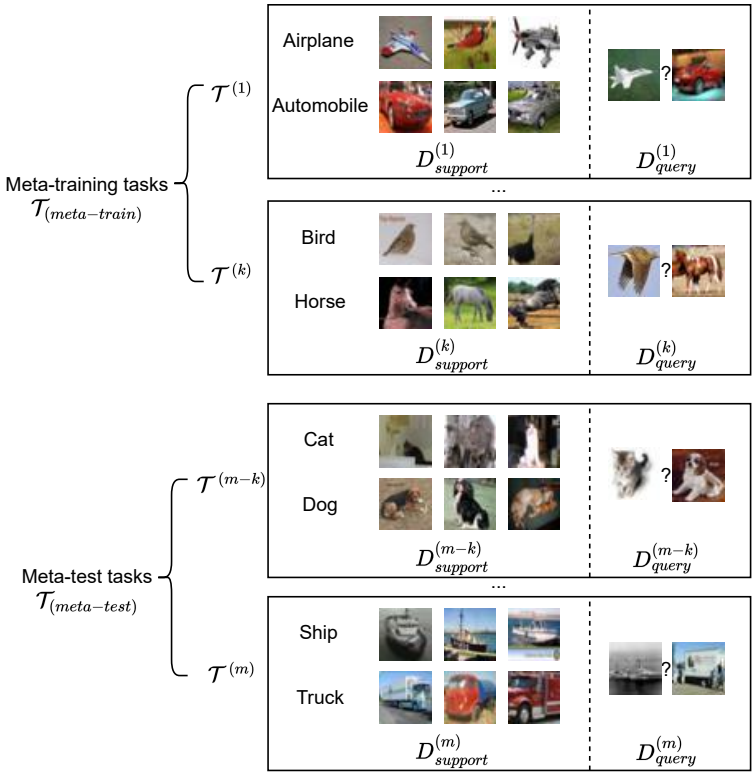


Fig. 3.9 **Meta-learning example setup.** Each task $\mathcal{T}$ is a binary classification task with a support set D_{support} and query set D_{query}. During meta-training, samples in D_{query} is known and the meta-learner aims to gain the optimal 'how to learn' $\hat{\boldsymbol{\omega}}$ from meta-training tasks. During meta-test, the meta-learner utilises $\hat{\boldsymbol{\omega}}$ to tackle unseen tasks from meta-test tasks and predict labels.

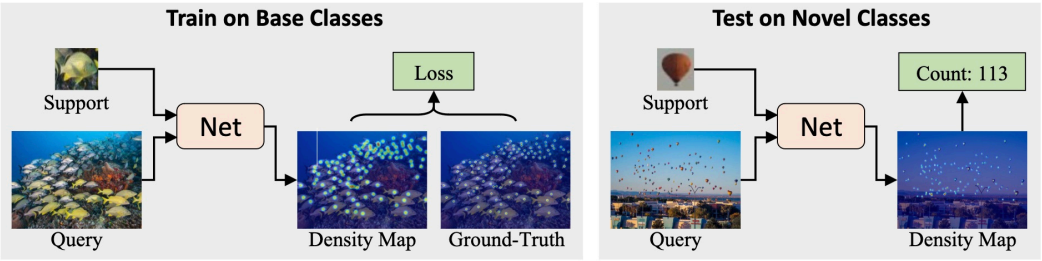


Fig. 3.10 **Illustration of few-shot object counting** [156]. This task aims to count the number of exemplar objects occur in the query image where the exemplar objects are described in only *a few* support images. It is assumed that the object classes in training phase have no intersection with the object classes in test phase.

training phase and test phase respectively, where $\mathcal{C}_b$ are completely different to $\mathcal{C}_n$. In the training phase, the model learns from the query image and a few support images with ground

truth density map provided.[16] In the test phase, the model predicts a density map for a given query image with only a few support images provided by leveraging the knowledge gained from $\mathcal{C}_b$. The detailed explanation of density map is given in § 3.3.1.

3.3 Object counting

3.3.1 Counting by density estimation

Counting objects by density estimation is a supervised approach originally proposed by Lempitsky and Zisserman [78]. It avoids difficult localisation and detection of individual object instances by estimating a density map from the input image. The *density* is defined as the degree to which an area is filled with people or things. The *density map* is presented as a one-dimensional image whose height and width match that of the three-dimensional (red, green, blue) input image. The sum of values over any region in the density map indicates the number of objects within that region in the input image. This approach is also interpreted as a regression task which aims to predict a density value for each pixel in the input image.

Compared with detection based counting methods, the ground truth for density estimation based counting methods is significantly less laborious to create. This is because detection based methods traditionally require bounding box annotations, whereas density estimation based methods only require dotted annotations that are easier to create: i.e. dotted annotations only require marking the centre of each object rather than drawing a bounding box for each object. Additionally, the hard task of predicting coordinates and objectness (a score to indicate how likely the object exists) related to the bounding box in object detection is replaced by predicting the density map. Moreover, large density values can represent objects that are occluded or heavily overlapped given the fact that a density value can range from zero to any positive number. Thus, it is potentially useful when images possess occlusion or a background cluttered with extremely dense objects [43]. Finally, a density map contains spatial information about the input image based on the position of dots.

As shown in Fig. 3.11, a ground truth density map is derived from dotted annotations on which a Gaussian kernel convolves. The motivation of applying a Gaussian kernel convolution on dotted annotations is to improve the robustness of counting algorithms to image noise, i.e. unwanted high-frequency signals a camera receives during image capturing. As illustrated in Fig. 3.12, a normal counting workflow involves the use of Gaussian convolution to remove noise so that the counting algorithm can product a high quality dot map (a dot map without

[16] Few-shot object counting still uses traditional terms for training and test phases by convention. This is slightly different to the meta-training and meta-test phases used in meta-learning.

	280	281	282	283	284	285	286	287	288	289
708	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
709	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
710	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
711	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
712	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
713	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
714	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
715	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
716	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
717	00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000
718	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
719	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
720	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
721	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
722	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
723	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
724	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
725	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
726	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
727	00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

(a) Pixel values in a dot map, representing position of a single object.

	280	281	282	283	284	285	286	287	288	289
708	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
709	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
710	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
711	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
712	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
713	0.00076	0.00183	0.00342	0.00498	0.00564	0.00498	0.00342	0.00183	0.00076	0.00000
714	0.00183	0.00439	0.00821	0.01194	0.01353	0.01194	0.00821	0.00439	0.00183	0.00000
715	0.00342	0.00821	0.01533	0.02231	0.02528	0.02231	0.01533	0.00821	0.00342	0.00000
716	0.00498	0.01194	0.02231	0.03246	0.03678	0.03246	0.02231	0.01194	0.00498	0.00000
717	0.00564	0.01353	0.02528	0.03678	0.04168	0.03678	0.02528	0.01353	0.00564	0.00000
718	0.00498	0.01194	0.02231	0.03246	0.03678	0.03246	0.02231	0.01194	0.00498	0.00000
719	0.00342	0.00821	0.01533	0.02231	0.02528	0.02231	0.01533	0.00821	0.00342	0.00000
720	0.00183	0.00439	0.00821	0.01194	0.01353	0.01194	0.00821	0.00439	0.00183	0.00000
721	0.00076	0.00183	0.00342	0.00498	0.00564	0.00498	0.00342	0.00183	0.00076	0.00000
722	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
723	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
724	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
725	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
726	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
727	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

(b) Pixel values in a density map for a single object.

Fig. 3.11 A comparison of pixel values in a dot map and density map. The range of pixel values is from 0 to 1 where 0 means it is a non-object background and 1 means it is an object centre. After applying the Gaussian convolution, the single dot shown in Fig. 3.11a expands to a broader region where the sum of all pixel values is still 1 as shown in Fig. 3.11b.

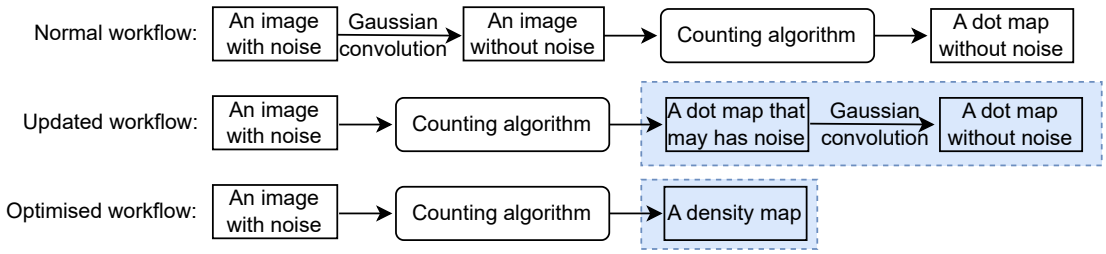


Fig. 3.12 Illustration of the optimised counting workflow with density map.

noise). The noise removal step in the normal workflow can be shifted to the end to save computational resources because a dot map that may have noise has less pixels than an image with noise. The updated workflow can be further optimised by using a density map to replace the noise removal on the dot map that may have noise (the two blue blocks are interchangeable).

Lempitsky and Zisserman [78] also claim that it does not matter how exactly the ground truth density is defined locally, as long as the sum of the ground truth density over the region reflects the object counts correctly. Therefore, the ground truth density for a dotted annotation where each dot has a pixel value of 1 is defined as the sum of normalised Gaussians centred at the dot, where the sum is still 1. Another benefit of applying the Gaussian kernel on a dotted annotation is the representation of a general object shape. This is because the sparse density resulted from Gaussian convolution spread over a larger area can represent a general object shape.

Assume that a set of N images (pixel grids) I_1, I_2, ..., I_N is given. Each image I_i is annotated with a set of points $\mathbf{P}_i = \{P_{i1}, P_{i2}, \ldots, P_{iC(i)}\}$, where $C(i)$ is the total number of objects annotated by users, and a point P_{ij} indicates an object centre in the image I_i. The density function is a real-valued function over pixel grids, whose integrals over image regions represent the object counts. Following [78], for an image I_i, the ground truth density function F^0 is defined as a kernel density estimate based on the provided points:

$$F_i^0(p) = \sum_{P_{ij} \in \mathbf{P}_i} \mathcal{N}(p; P_{ij}, \sigma^2 \mathbf{1}_{k \times k}) \quad \forall p \in I_i \tag{3.52}$$

Where p denotes a pixel, $\mathcal{N}(p; P_{ij}, \sigma^2 \mathbf{1}_{k \times k})$ denotes a normalised Gaussian kernel whose size is $k \times k$ and sum is 1 evaluated at p, with the mean at the user-placed dot P_{ij}, and the Gaussian standard deviation (an isotropic covariance matrix) σ. The choice of kernel size $k \times k$ and Gaussian standard deviation σ are data dependent, and specified in the late experimental sections.

3.3.2 Region of interest pooling and align operations

Region of interest (RoI) pooling and region of interest align are commonly used in object detection algorithms [17, 87, 47, 118, 83, 55] to produce a fixed-size feature map from a non-uniform input because fully connected layers require a fixed-size input. The non-uniform input to an object detection algorithm is often encountered because different objects have different sizes which produce different sizes of feature map. In other words, these two operations aim to map RoI from the original input image onto the non-uniform feature map to produce a fixed-size feature map.

Region of interest pooling

Region of interest pooling (RoI pooling) [47] maps RoI in an input image to the region in a feature map followed by a max or average pooling to produce a fixed-size feature map.

If a feature map is reduced k times from its original input image, RoI's coordinate in the feature map is computed by dividing its original coordinate by k with quantisation which is a process of rounding down a real number to an integer. This mapping process is illustrated in Fig. 3.13. After that, RoI's data in the feature map is sampled by max or average pooling to produce a fixed-size feature map. An example of max pooling process is demonstrated in Fig. 3.14.

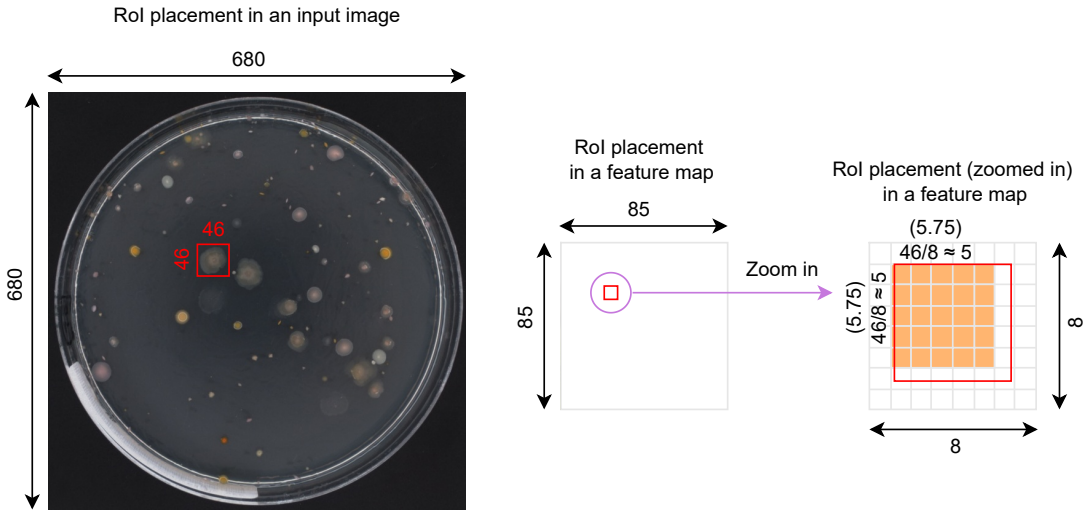


Fig. 3.13 Illustration of RoI (max) pooling - mapping process.

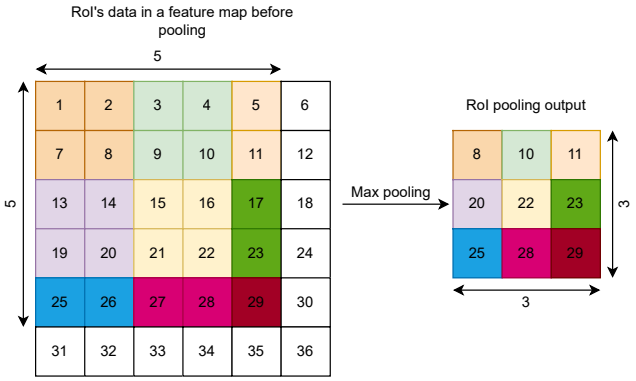


Fig. 3.14 Illustration of RoI (max) pooling - pooling process (The data in feature map is made up).

According to He et al. [55], the quantisation in RoI pooling leads to RoI misalignment and information loss. As illustrated in Fig. 3.13, a 46×46 RoI (the colony highlighted by a red bounding box) is proportionally projected from the $3 \times 680 \times 680$ input image onto the 5×5 RoI in the $1 \times 85 \times 85$ future map which is 8 times smaller. The quantisation causes the

shift of the actual RoI in the red bounding box to orange areas because the original 46×46 RoI is not divisible by 8. Additionally, RoI misalignment leads to information loss because of the reduced information for RoI to sample. The RoI misalignment and information loss become more severe when the object is small in the input image.

Region of interest align

Region of interest align (RoI align) is proposed by He et al. [55] to tackle RoI misalignment and information loss in RoI pooling. RoI align requires users to predetermine the target output size, as also required in RoI pooling. It will then equally divide the original RoI into boxes based on the target output size. In each box, four sampling points are used to sample data from the feature map. The value of each sampling point is calculated from its four nearest neighbouring pixels by using the bilinear interpolation algorithm[15]. The average or max value of these four sampling points forms the value for its corresponding box in the output feature map. According to Lin et al. [83], the position of these four sampling points has little impact on their experimental results, as long as no quantisation is performed. This thesis uses 1/3 as the position implemented in Pytorch [107], meaning the position of these four sampling points equally divides the box into three partitions vertically and horizontally.

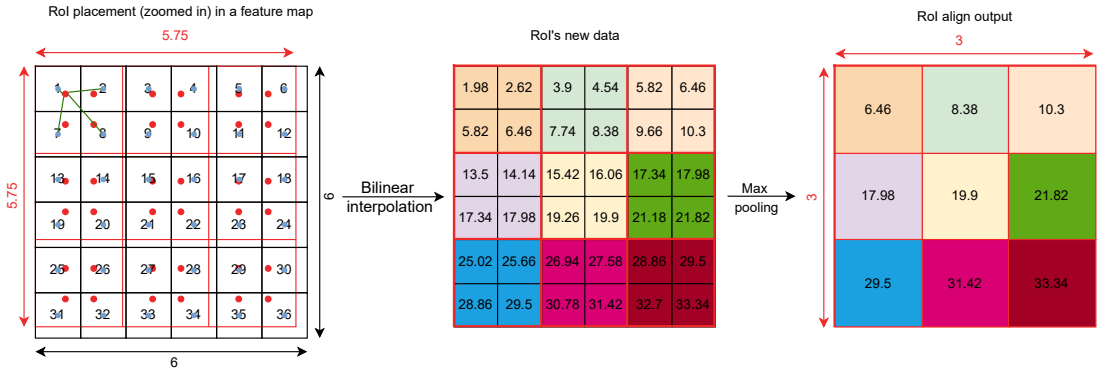
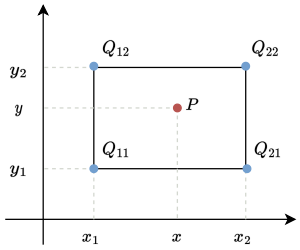


Fig. 3.15 Illustration of RoI (max) align.

Following the RoI max pooling example in Fig. 3.13 and 3.14, RoI align is demonstrated in Fig. 3.15 with the same made-up data. The original 5.75×5.75 RoI (red grid) is equally divided into boxes based on the target output size (3×3 in this example). Four sampling points (red dots) inside each box are used to sample data from the feature map where the value of each sampling point is interpolated from its closest four neighbouring pixels (four blue dots connected by green lines) by using the bilinear interpolation algorithm shown in Fig 3.16. The maximum value of these four sampling points becomes the value for its



$$P(x,y) = \frac{y_2-y}{y_2-y_1}\left(\frac{x_2-x}{x_2-x_1}Q_{11} + \frac{x-x_1}{x_2-x_1}Q_{21}\right) + \frac{y-y_1}{y_2-y_1}\left(\frac{x_2-x}{x_2-x_1}Q_{12} + \frac{x-x_1}{x_2-x_1}Q_{22}\right)$$

Fig. 3.16 Illustration of bilinear interpolation.

corresponding box in the output feature map. The position of these four sampling points normally divides the box into three partitions vertically and horizontally.

3.4 FamNet

3.4.1 Overview

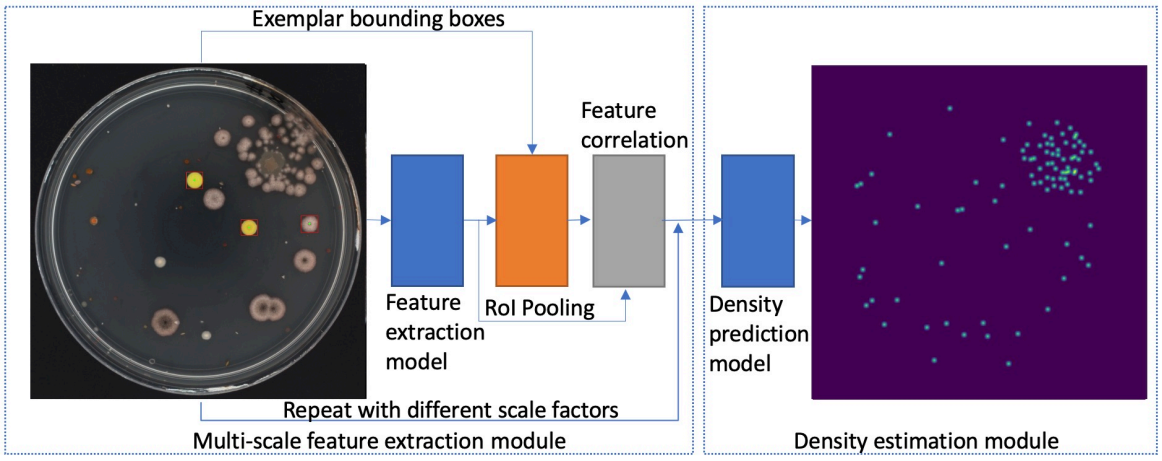


Fig. 3.17 Overview of FamNet

An overview of FamNet applied to bacterial colony counting is illustrated in Fig. 3.17. It takes a plate image as input along with the location of three exemplars (red bounding boxes in the plate image). The plate image is input to a feature extraction model, whose parameters are always frozen during training and inference phases, to produce a feature map. The location of three exemplars are used to extract features in the feature map by RoI pooling. The extracted features are used as kernels to convolve the feature map. The RoI pooling and feature correlation are repeated on different feature maps returned by the

feature extraction model, and further repeated multiple times with the exemplars scaled by different scale factors (Fig. 3.18 shows more details). The outputs are stacked into a single correlated feature map which is input to the density estimation module to predict a density map (Fig. 3.19 shows more details). The colony count is obtained by adding up all density (pixel) values in the predicted density map.

3.4.2 Multi-scale feature extraction module

The multi-scale feature extraction module aims to extract exemplar features from the feature map returned by the feature extraction model, as well as to perform feature correlation using exemplar features on the feature map.

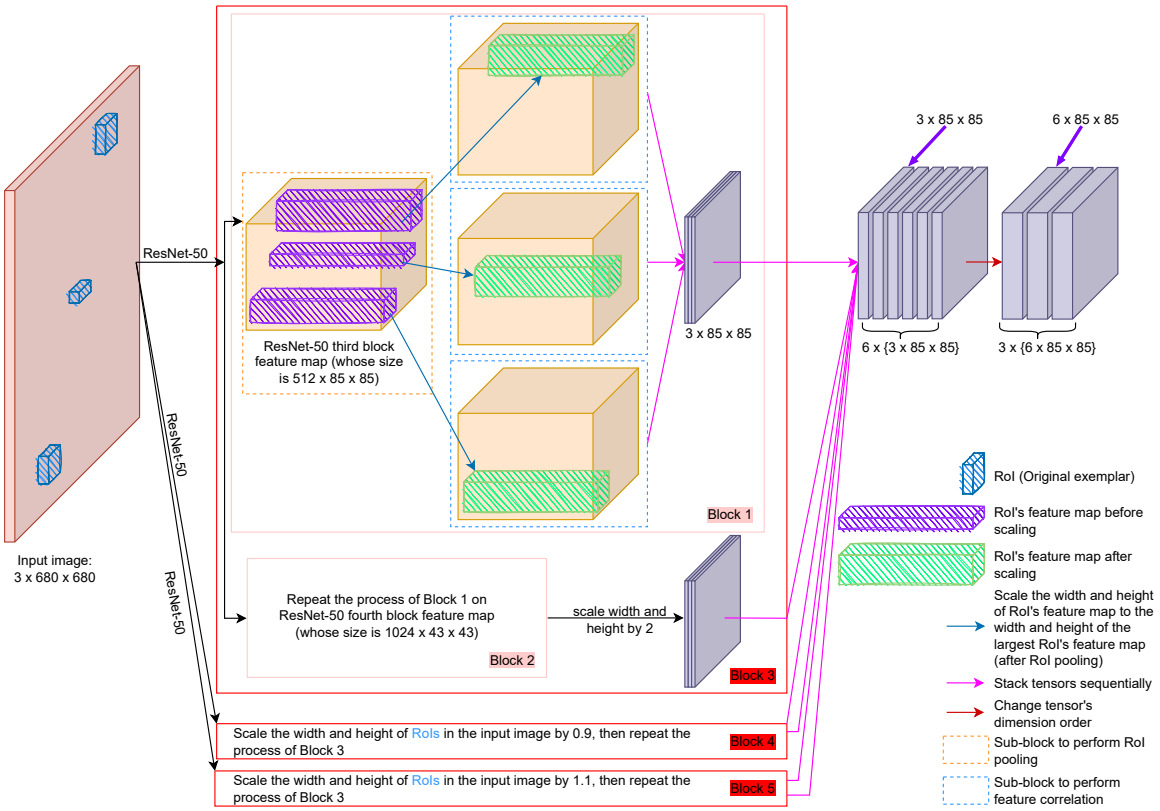


Fig. 3.18 FamNet multi-scale feature extraction module.

A detailed feature extraction process is presented in Fig. 3.18. FamNet feature extraction module takes a plate image and three exemplars specified by the bounding box's top-left and bottom-right corners' coordinates from annotations as input. This module can take a plate image of different resolutions. The $3 \times 680 \times 680$ plate image is only used for illustration purposes. The plate image is input to a feature extraction model whose parameters are always

frozen. In other words, the parameters of the feature extraction model are never updated during training and test phases so that this model only acts as a feature extractor. The feature extraction model is ResNet-50 [56] pre-trained on ImageNet [64]. ResNet-50 stands for residual network with 50 layers which are organised into 5 different blocks. ResNet-50 is widely used to empower computer vision applications.

The output of the third block layer of ResNet-50, as well as the output of the fourth block layer of ResNet-50 are extracted as two feature maps. The ResNet-50 third block feature map and ResNet-50 fourth block feature map have three dimensions. They are depth, height and width. The depth depends on the number of kernels used for convolution, and the height and width depend on the convolutional kernel's height and width, convolution stride and padding method.

The location of exemplars (blue cubes in Fig. 3.18), the RoI, in the input image is proportionally projected to the ResNet-50 third block feature map (orange cube in Fig. 3.18) by dividing the coordinates of exemplars by 8. This is because the height and width of ResNet-50 third block is 8 times smaller than the input image's height and width. Then, the projected location (location of purple cubes in Fig. 3.18) is used to extract features from the ResNet-50 third block feature map to form the RoI's feature map. This process is called *RoI pooling*, and it is performed for each RoI. After that, the height and width of each RoI's feature map are scaled to match the largest RoI's height and width by using the bilinear interpolation algorithm [15]. Each RoI's scaled feature map is used as a kernel (green cube in Fig. 3.18) to convolve ResNet-50 third block feature map with zero padding added to preserve the height and width. The outputs of these three convolutions are stacked sequentially to form a correlated feature map.

The process of RoI pooling, feature correlation, and stacking based on ResNet-50 third block feature map (in Fig. 3.18 Block 1) is repeated on ResNet-50 fourth block feature map after dividing the coordinates of exemplars by 16. This is because the height and width of ResNet-50 fourth block are 16 times smaller than the input image's height and width. The height and width of the correlated feature map is scaled by 2 to match the height and width of ResNet-50 third block feature map so that these two correlated feature maps can be stacked together.

The process of RoI pooling, feature correlation, and stacking based on ResNet-50 third and fourth feature map (in Fig. 3.18 Block 3) is repeated after scaling the height and width of each RoI in the input image by 0.9 and 1.1 separately. This helps FamNet to handle the same object at different scales. In addition to the previous two correlated feature maps, this process produces another four correlated feature maps. Then, these six correlated feature maps are stacked sequentially to form a four-dimensional feature map, which is also referred

to as a tensor (multi-dimensional array). The dimension order of the four-dimensional feature map is reorganised into the number of exemplars $\times$ the number of feature maps $\times$ feature map height $\times$ feature map width. The final feature map from FamNet multi-scale feature extraction is $3 \times 6 \times 85 \times 85$ when the input image is of size $3 \times 680 \times 680$.

Algorithm 5: FamNet multi-scale feature extraction.

Input : An image x
A feature extraction model *ResNet*-50
A list of scale factors S
A list of the bounding boxes B
Output : Extracted feature map y

$y \leftarrow \emptyset$;
$y_{b3} \leftarrow$ `GetThirdBlockFeatureMap(`*ResNet-50, x*`)`;
$y_{b4} \leftarrow$ `GetFourthBlockFeatureMap(`*ResNet-50, x*`)`;
$y_b \leftarrow \{y_{b3}, y_{b4}\}$;
$h_{largest} \leftarrow$ `GetHeightOfTheLargestBoundingBox(`B`)`;
$w_{largest} \leftarrow$ `GetWidthOfTheLargestBoundingBox(`B`)`;
foreach $s \in S$ **do**
  foreach $y_{temp} \in y_b$ **do**
    $k \leftarrow \emptyset$;
    foreach $b \in B$ **do**
      $s_{special} \leftarrow$ `CalculateScaleFactor(`x, y_{temp}`)`;
      $b_{scaled} \leftarrow$ `ScaleBoundingBox(`b, $s_{special} \times s$`)`;
      $k_{temp} \leftarrow$ `RoIPooling(`y_{temp}, b_{scaled}`)`;
      $k_{temp} \leftarrow$ `Scale(`k_{temp}, $(h_{largest} \times s \times s_{special}, w_{largest} \times s \times s_{special})$`)`;
      $k \leftarrow$ `Stack(`k, k_{temp}`)`;
    end
    $y_k \leftarrow$ `Conv(`y_{temp}, k`)`;
    if $y_{temp} == y_{b4}$ **then**
      $y_k \leftarrow$ `Scale(`y_k, y_{b3}`)`;
    end
    $y \leftarrow$ `Stack(`y, y_k`)`;
  end
end
$y \leftarrow$ `ChangeDimensionOrder(`y`)`;

The pseudocode for implementing FamNet feature extraction is presented in Algorithm 5. It also provides an overview of the repetitive RoI pooling and feature correlation. S contains three scale factors 1, 0.9, and 1.1. B contains three exemplar bounding boxes where each bounding box is represented by its top-left and bottom-right corners' coordinates. y_{b3} and y_{b4} indicate ResNet-50 third and fourth block feature maps when ResNet-50 is input an

image x. They are later stored in a list y_b. $CalculateScaleFactor$ calculates the scale factor of the feature map against the input image so that the bounding box's coordinates can be proportionally projected to the feature map. *ScaleBoundingBox* indicates the scaling of height and width of input by the specified scale factor. $Scale$ indicates the scaling of height and width of input to target height and width. *Stack* represents the stacking of multiple inputs to produce an output whose shape has an additional dimension which is determined by the number of inputs stacked together. *Conv*, which is also referred to as feature correlation, represents the convolution of kernel k performed on the feature map by one stride with zero padding added so that the output's height and width are the same as those of the feature map. Finally, $ChangeDimensionOrder$ will reorganise the feature map's dimension order into the number of exemplar bounding boxes, the number of feature maps, the height of ResNet-50 third block feature map, and the width of ResNet-50 third block feature map.

3.4.3 Density map prediction module

The density map prediction module receives the final feature map from the previous module as input and predicts a density map. The density prediction model in this module consists of five convolutional layers and three upsampling layers placed after each of the first, second and third convolutional layers. Each upsampling layer will double the input's height and width using the bilinear interpolation algorithm [15]. The kernel parameters, which are the number of kernels, height and width, for these five convolutional layers are $196 \times 7 \times 7$, $128 \times 5 \times 5$, $64 \times 3 \times 3$, $32 \times 1 \times 1$, and $1 \times 1 \times 1$ respectively. The first, second, and third convolutional layers have zero padding added to preserve the input's height and width. All convolutions are performed by one stride. Additionally, ReLU is used as the activation function to activate the output of each convolutional layer.

The network architecture for the density prediction model is detailed in Fig. 3.19 following the example data shown in Fig. 3.18. The input feature map has four dimensions. They are the number of exemplar bounding boxes, the number of feature maps, the height of ResNet-50 third block feature map and the width of ResNet-50 third block feature map. These five convolutions are performed on the input's the number of exemplar bounding boxes dimension. The outputs are four-dimensional, which are the number of exemplar bounding boxes, the number of kernels, the height of feature map, and the width of feature map. The four-dimensional output of the last ReLU activated convolutional layer is averaged based on the number of exemplar bounding boxes dimension. It results in a three-dimensional output: the number of kernels, the height of the feature map and the width of the feature map. Among them, the number of kernels is 1, and the height and width of the feature map match those of the input plate image in the feature extraction module. The final output is the

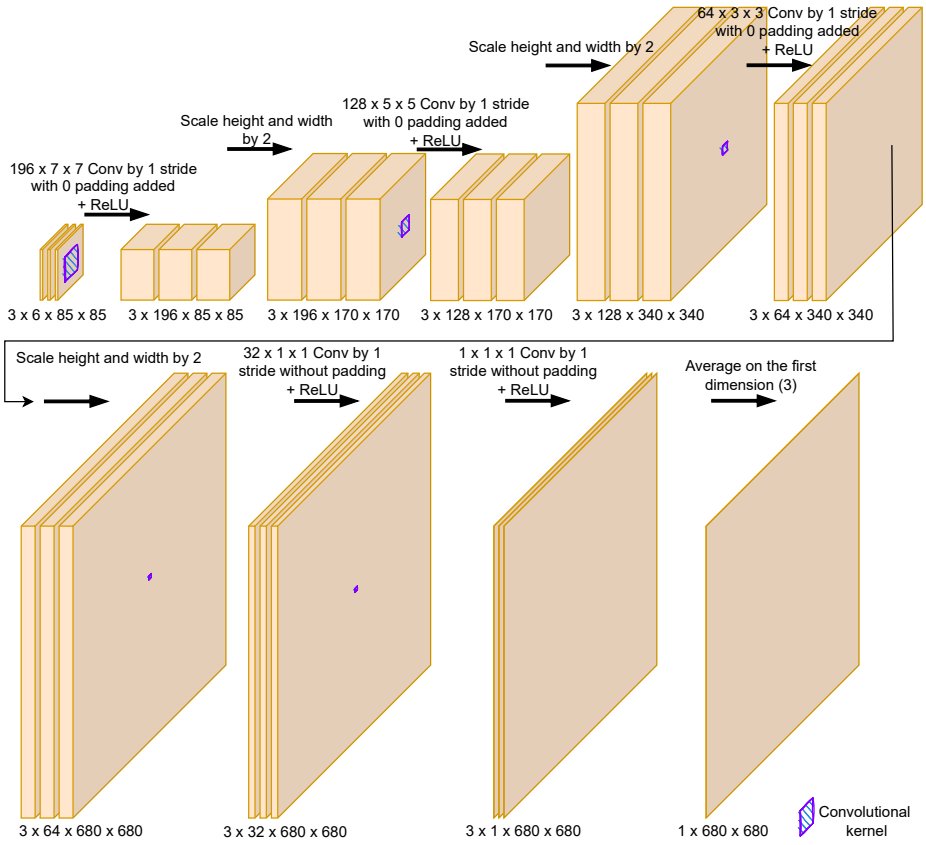


Fig. 3.19 FamNet density estimation model.

predicted density map whose sum of values indicates the number of colonies in the input plate image in the feature extraction module.

3.5 Data description

This section introduces two collections of images on which experiments in this thesis are conducted. They are Microbia data set and Synoptics data set. The former is a public data set, and the latter is provided by Synoptics Ltd who is an industrial partner of this research. Image annotation, data set split, data set statistics, and a derived data set of Synoptics data set are also discussed.

3.5.1 Microbia data set

Microbia data set is a collection of 28418 segments created by Ferrari et al. [38] to investigate colony-cardinality classification.[17] This data set is used in this thesis because it is the only labelled large data set to investigate colony-cardinality classification. A segment represents a standalone region of the colony cluster in the plate image regardless of the colony count. Each segment is manually cropped from the plate image into a square image whose dimension is determined by either the longer height or the longer width of the colony cluster, followed by a border extension of a fixed size and padding. Fig. 3.20 illustrates a plate image from which the segments are cropped.[18]

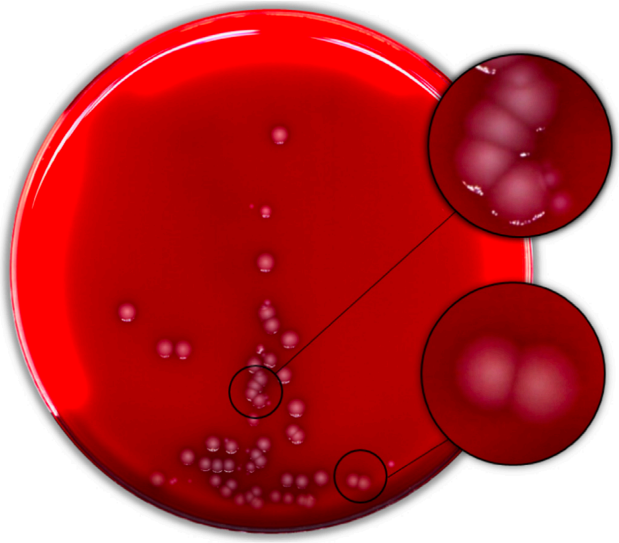


Fig. 3.20 A plate image with clustered bacterial colonies grown on a blood agar [38].

The cropped segment is manually assigned to a label to indicate the number of colonies. Seven labels are provided in this data set. They are Outlier class, One-colony class, Two-colonies class, Three-colonies class, Four-colonies class, Five-colonies class and Six-colonies class. Among them, Outlier class means the segment is bubble, dust, or dirt, which has zero colonies. Some segments of these seven classes are shown in Fig. 3.21. Particularly, the first Five-colonies segment and the second Six-colonies segment contain some neighbouring colonies which may disrupt colony counting algorithms. Moreover, the class distribution of Microbia data set is imbalanced. This is because One-colony class constitutes 50.27% of

[17] Its original download url is no longer available now. But a snapshot is available at https://web.archive.org/web/20220401001815/https://www.microbia.org/index.php/resources. Additionally, data split is not provided in Microbia data set, meaning it does not contain fixed training set, validation set and test set.

[18] Microbia data set does not contain any plate images. This image is from [38].

the whole data set, whereas the rest six classes constitute 49.73% of the whole data set. The detailed class distribution is listed in Table 3.2.

Fig. 3.21 Segments of seven different classes.

Table 3.2 Class distribution in Microbia data set.

Class	Count	Percentage (%)
One-colony	14285	50.27
Two-colonies	5443	19.15
Three-colonies	3634	12.79
Four-colonies	1836	6.46
Five-colonies	953	3.35
Six-colonies	1006	3.54
Outlier	1261	4.44

In addition to the creation of labelled segments, Microbia data set includes a collection of 28418 masks for the cropped segments to allow users to remove disturbing neighbouring colonies. This can be achieved by applying a bitwise and operation on the segment with the mask. In other words, the pixel in the segment is assigned a value of zero if its corresponding pixel in the mask has a value of zero, and the pixel in the segment is unchanged if its corresponding pixel in the mask has a value that is greater than zero. Some masks and the masked segments of these seven classes corresponding to the segments in Fig. 3.21 are demonstrated in Fig. 3.22 and 3.23. The first Five-colonies segment and the second Six-colonies segment become cleaner as shown in Fig. 3.23 because the disturbing neighbouring colonies are removed by using the mask. This thesis uses masked segments to avoid disruptions from the disturbing neighbouring colonies.

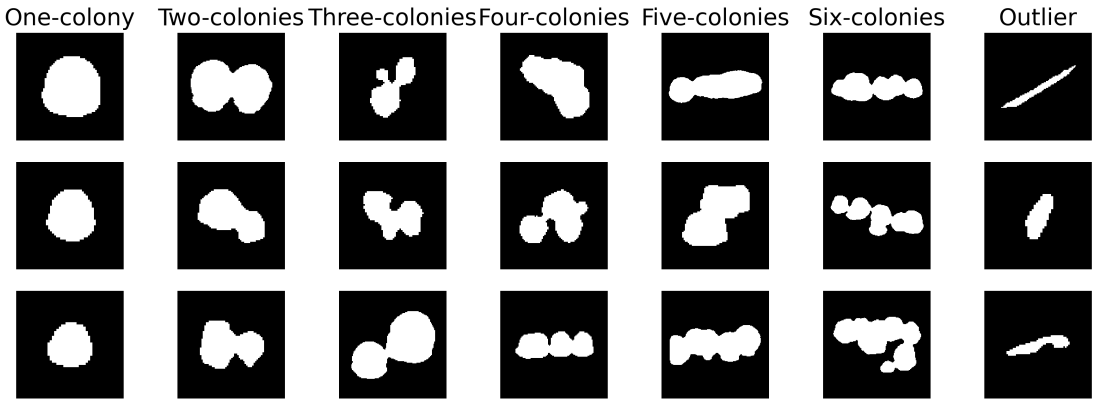


Fig. 3.22 Masks for segments in Figure 3.21.

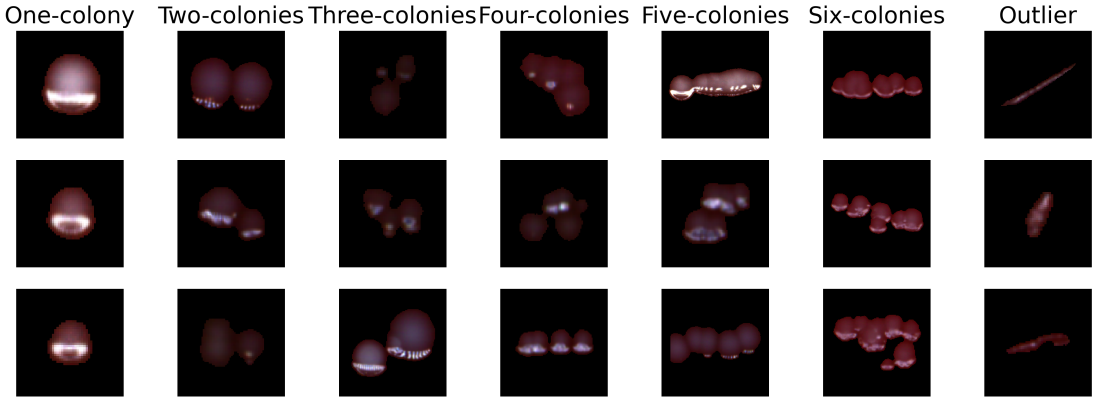


Fig. 3.23 Masked segments generated based on Figure 3.21 and Figure 3.22.

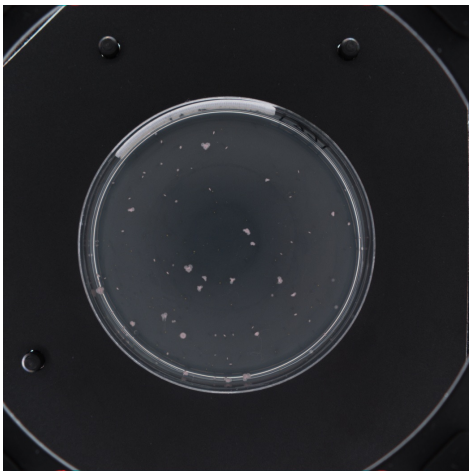
3.5.2 Synoptics data set

Image collection

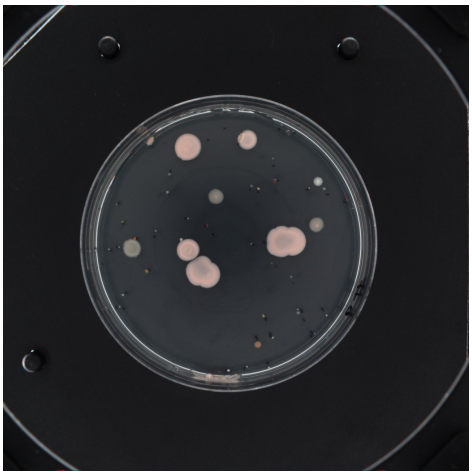
A collection of 572 plate images of colonies was provided by Synoptics Ltd to support this research. These images are organised into different batches, each is different in resolution, background colour, image format, shape of petri dish, existence of sticky labels and hand-writing in petri dish, and colony species. Moreover, many images are either duplicates or from the same petri dish but taken with the petri dish slightly rotated. Furthermore, colony species remain unknown because Synoptics Ltd has little information about the source of these images, meaning different images may contain the same colony species.

To ensure the assumption that samples are i.i.d., a subset of these images are used to build and evaluate deep learning models. This subset consists of 125 images chosen from the same batch which forms Dataset V1. Consequently, images in Dataset V1 have the same $3 \times 1040 \times 1040$ resolution, image format and petri dish shape. Among the chosen

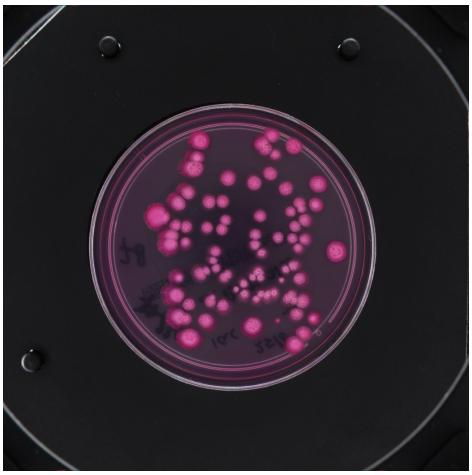
125 images, there are no duplicates or the same petri dish taken with its position rotated. Moreover, different images among these 125 images may contain the same colony species. Four hand-picked example images in Fig. 3.24 illustrate the difference in colony species, shape and colour. For instance, colonies shown in Fig. 3.24b and 3.24c have a larger size than those shown in Fig. 3.24a and Fig. 3.24d. Colonies are shown as grey, pink, red and brown in Fig. 3.24a, 3.24b, 3.24c and 3.24d respectively, indicating colony species are different across these images. But they all have a circular petri dish centred in the image and have a large non-plate area which is the area outside the circular petri dish in the image.



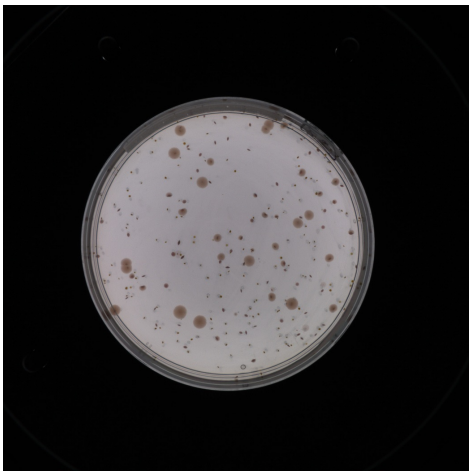
(a) Plate image one.



(b) Plate image two.



(c) Plate image three.



(d) Plate image four.

Fig. 3.24 Plate images with colonies of different species, colour and shape.

Image annotation

The colony centres and bounding boxes are created with the assistance of a proprietary piece of software called ProtoCOL provided by Synoptics Ltd.[19] ProtoCOL requires careful and manual parameter adjustment for single colony and clustered colonies in each image, which is very time-consuming. The created colony centres and bounding boxes are checked manually by myself to ensure the correctness.

Data set split

The Dataset V1 is split into a fixed training set and test set with a ratio of 8 : 2. This ratio ensures that 20% of the whole data is kept as a test set as suggested by Ranjan et al. [112]. Therefore, the training set and test set consist of 100 and 25 images respectively. Because different images in this data set may have the same colony species, it is assumed that images in the training set contain the same colony species in the test set.[20]

Data statistics

The statistics for colony counts in different data sets are shown in Fig. 3.25. In each sub-figure, x axis is the colony count with an interval of 10; y axis is the number of plate images whose colony counts fall in the corresponding range. Fig. 3.25 shows that colony counts between 40 and 50 occur more frequently in the training set and the whole set; colony counts between 70 and 80 occur more frequently in the test set; and overall each plate image contains at least 20 colonies.

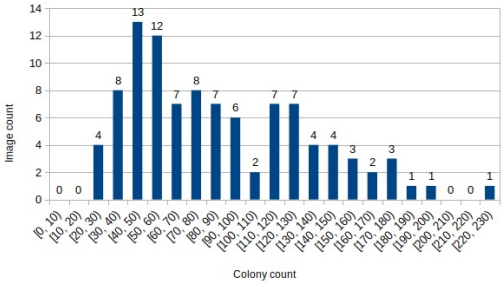
Table 3.3 Mean, standard deviation and variance of training data, test data and the whole data.

Data set	Mean	Standard deviation	Variance
Training	(54.43, 56.19, 60.98)	(30.05, 29.74, 30.45)	(903.25, 884.39, 927.27)
Test	(53.48, 55.44, 60.03)	(28.65, 28.63, 29.35)	(820.91, 819.79, 861.32)
The whole	(54.24, 56.04, 60.79)	(29.78, 29.52, 30.24)	(886.93, 871.56, 914.22)

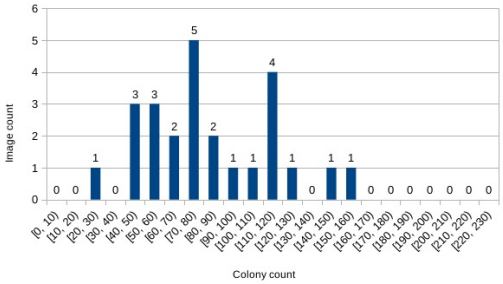
The mean, standard deviation and variance of training data, test data and the whole data are detailed in Table 3.3. Values inside each pair of parenthesis are the value calculated from red channel, green channel and blue channel across all images in the data set, where the pixel

[19] https://www.synbiosis.com/product/automated-colony-counting-zone-measurement-protocol-3

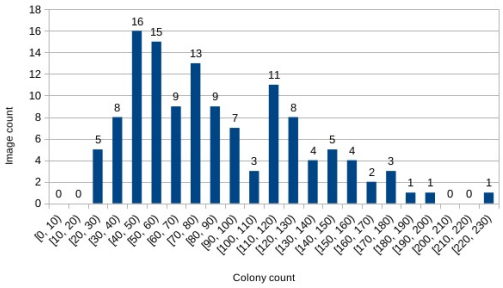
[20] This implies the assumption that base classes do not overlap with novel classes in few-shot object counting is broken when this data set is used to develop few-shot object counting algorithms.



(a) Statistics in training set.



(b) Statistics in test set.



(c) Statistics in the whole set.

Fig. 3.25 Statistics for colony counts in different data sets.

value ranges from 0 to 255.[21] The mean of training data is close to the mean of test data, meaning these two data sets are drawn from the same distribution. However, test data has a smaller standard deviation and a smaller variance than training data.

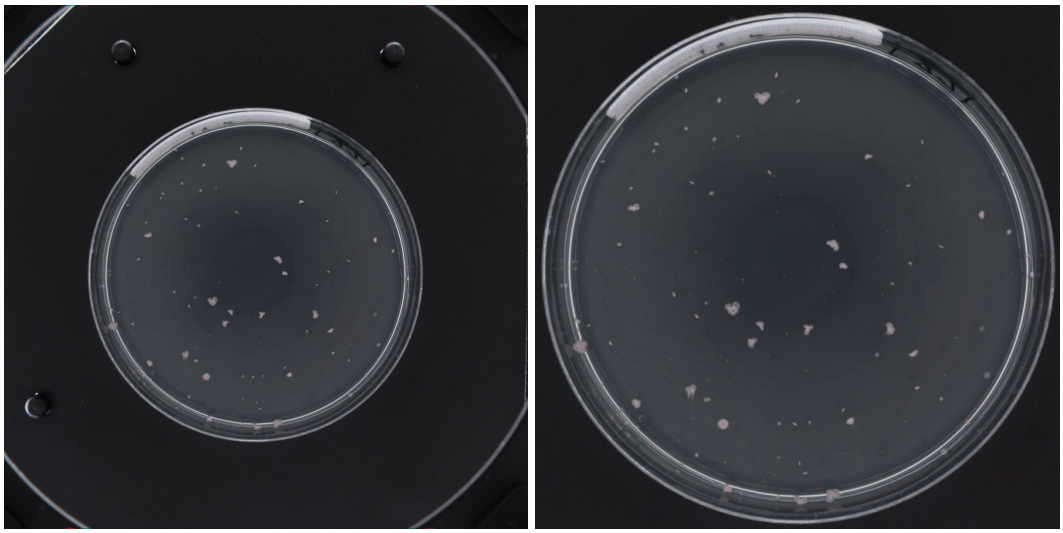
Derived data set

Table 3.4 Synoptics data sets.

Name	Explanation
Dataset V1	Original $3 \times 1040 \times 1040$ image.
Dataset V2	Identical to Dataset V1 except that each image is cropped from the petri dish centre until its edge to reduce the image size from 1040×1040 to 680×680.

In addition to Dataset V1, a derived data set named Dataset V2 is manually created using Photoshop to enable different types of experiment. Dataset V2 consists of all images in Dataset V1 except the plate image is cropped from the petri dish centre until its edge so that the redundant background in each plate image is completely removed. As a result, the image

[21]The pixel value that ranges from 0 to 255 here is only used to demonstrate the difference of mean/standard deviation/variance across different data sets. The pixel value will be rescaled to the range of 0 to 1 when images are used for developing/evaluating neural networks to save computational resources.



(a) Plate image one in Dataset V1. (b) Plate image one in Dataset V2.

Fig. 3.26 A comparison of Plate image one between Dataset V1 (original) and V2 (cropped). The image in (a) is of shape $3 \times 1040 \times 1040$. The image in (b) is of shape $3 \times 680 \times 680$.

size is reduced from 1040×1040 to 680×680. The differences between these two data sets are visualised in Fig. 3.26 and summarised in Table 3.4. Additionally, Dataset V2 is split into a fixed training set and test set with the same ratio of 8:2 and the same seed, meaning images in training/test set from Dataset V2 are identical to those in training/test set from Dataset V1 except the difference in image resolution. All images in Dataset V2 are provided in Appendix A.1.

Part I

Aspects of cardinality classification

Chapter 4

Counting by cardinality classification

This chapter introduces MicrobiaNet, which is the best-performing cardinality classification method for colony counting to the best of my knowledge. A series of case studies of the impact of imbalanced class distribution, visual similarity and limited labelled data on the counting performance will be conducted on the application of bacterial colony counting. Some research questions listed in Chapter 2 from aspects of cardinality classification will be investigated here as well. It will be empirically shown that visual similarity is the key issue to address, and class imbalance has a limited impact on the counting performance when counting small and clustered colonies by cardinality classification.

This chapter aims to address the more specific research gap identified in § 2.5.2. The generalisation problem of applying counting algorithms to a different domain/category is tentatively neglected. This is because the outcome of this investigation will be used to help Synoptics Ltd decide if this method can be integrated into their in-house colony counting technology.

4.1 Colony-cardinality classification baseline performance

Counting objects by cardinality classification is a supervised classification approach originally proposed by Ferrari et al. [37, 38] to count bacterial colonies. The *cardinality* indicates the number of colonies in an image.[1]. The counting algorithm receives an image that may contain a standalone object or clustered objects, and classifies it into a pre-defined cardinality class which represents the object count. The input image contains an unknown number of objects and is often obtained from an upstream object detection algorithm. The final object count is obtained by adding up the cardinality of all detected objects.

[1]Cardinality was originally used in mathematics and physics to indicate the number of elements in a set.

This case study builds a baseline performance of MicrobiaNet, which is the best-performing colony-cardinality classification algorithm to the best of my knowledge, to investigate the research question "To what extent does MicrobiaNet address small and clustered colonies?". Experimental results in this section will be used to explore the impact of class imbalance and image similarity on the counting performance in other case studies later this chapter.

4.1.1 Model

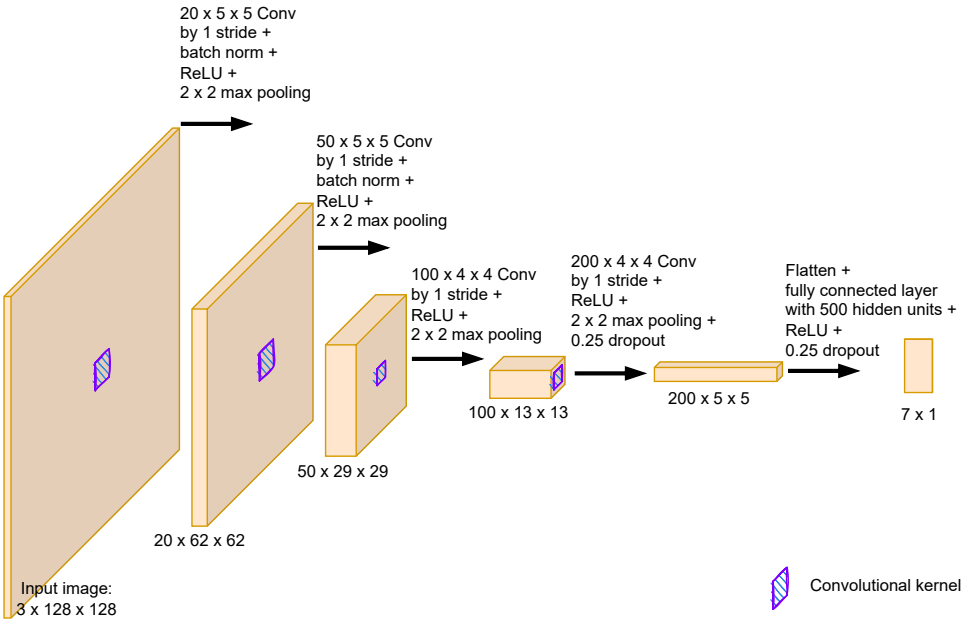


Fig. 4.1 MicrobiaNet architecture.

As illustrated in Fig. 4.1, MicrobiaNet consists of four convolutional layers and a fully connected layer. The input image is of size $3 \times 128 \times 128$, i.e. three-dimensional image whose height and width are both 128. The kernel parameters, which are the number of kernels, height and width, for these four convolutional layers are $20 \times 5 \times 5$, $50 \times 5 \times 5$, $100 \times 4 \times 4$, and $200 \times 4 \times 4$. These convolutional layers and the fully connected layer use a ReLU activation function. Additionally, the output of the first two convolutional layers is followed by batch normalisation and 2×2 max pooling.[2] The output of the last two convolutional layers is followed by 2×2 max pooling without batch normalisation. The last

[2]The original normalisation method is local response normalisation attached to ReLU activation. It is replaced with batch normalisation followed by ReLU activation as it produces a better result based on our experimental results.

convolution output is fattened before being input to a fully connected layer which consists of 500 hidden units.[3] Moreover, a dropout of the weights with 25% in probability is added before and after the fully connected layer.[4] Finally, the model produces 7 scores to indicate how likely the input image belongs to 7 classes.

4.1.2 Experimental setup

Data

In this chapter, the Microbia data set (as described in § 3.5.1) is split into training set, validation set and test set with a ratio of 6:2:2 so that different parts of the data set can be used to meet different needs. As a result, training set, validation set and test set have 17050, 5684 and 5684 images respectively. Other common split ratios such as 8:1:1 and 7:2:1 are not considered here because of the time constraint in this project and the relatively large data set size of 28418 which is more suitable for the ratio of 6:2:2. This split is a stratified split as shown in Table 4.1, meaning the class distribution in training set, validation set and test set remains identical. Additionally, the Microbia data set is randomly shuffled with 5 different seeds before the split to produce a robust estimate of MicrobiaNet's performance. To avoid confusion, Microbia data set split with 5 different seeds are called MicrobiaS1, MicrobiaS2, MicrobiaS3, MicrobiaS4, and MicrobiaS5 data sets where S indicates the seed.

Table 4.1 Class distribution of Microbia training, validation and test sets.

Class	Count			Percentage (%)
	Training	Validation	Test	
One-colony	8571	2857	2857	50.27
Two-colonies	3265	1089	1089	19.15
Three-colonies	2180	727	727	12.79
Four-colonies	1102	367	367	6.46
Five-colonies	571	191	191	3.35
Six-colonies	604	201	201	3.54
Outlier	757	252	252	4.44

[3]Flatten means a multi-dimensional array is changed to a two-dimensional array. For example, a multi-dimensional array of shape $200 \times 5 \times 5$ will be flatten to 200×25.

[4]The original dropout probability/rate is 75% as described in [38]. Based on our experimental results, 25% is the dropout rate that reproduces a similar performance.

Training and evaluation

MicrobiaNet is trained on the training set and evaluated on the validation set. The training and evaluation use MicrobiaS1, MicrbiaS2, MicrobiaS3, MicrobiaS4, and MicrobiaS5 data sets individually. In other words, MicrobiaNet is trained and evaluated five times with different data each time. The validation result is used to determine the baseline performance to facilitate other studies. Therefore, only one of the five trained models will be kept for other case studies. The test set is not used here to prevent the model from being overly optimised.

Before training, images in the training set are normalised by subtracting the mean of RGB pixel values in the training set and dividing by the standard deviation of RGB pixel values in training set to speed up convergence for training. The mean and standard deviation of RGB pixel values in the training set are also applied to normalise images in the validation set during the evaluation process. This data normalisation procedure has been widely used when training neural networks [74, 56].

The parameters of MicrobiaNet are initialised with the method proposed by He et al. [56]. Its benefit has been discussed in § 3.1.4. The network is optimised by Adam with a learning rate of 10^{-2}, a batch size of 64, an epoch number of 500, and a weight decay of 5×10^{-4}. These hyper-parameters are identical to [38] in which MicrobiaNet is introduced. The training will be early terminated to avoid a wasteful use of computational resources if the F1 score has not made 1% improvement for 100 epochs continuously. The F1 score is a weighted average F1 score calculated from validation set during training to mitigate the influence of imbalanced class distribution, which is further explained in the next section.

Evaluation metrics

Accuracy is the most commonly used metric to represent classification performance. However, it is less informative when the class distribution is imbalanced. Therefore, precision, recall and F1 score are also used in this chapter to represent classification performance since they are commonly used in imbalanced classification tasks [136, 66, 33]. Moreover, the performance is visualised in a confusion matrix.

Accuracy is the percentage of the total number of correct predictions over the number of all predictions. It can only partially reflect the performance because it does not provide an insightful view of classification on each individual class.

Precision is the proportion of predicted positives that are correctly classified. Its formula is $\frac{\text{True positive}}{\text{True positive+False positive}}$, where true positive is the number of real positive samples that are correctly classified, and false positive is the number of real negative samples that are incorrectly classified as positive. Each class will produce a precision value in the classification

result. The overall precision is a weighted average of multiple precision values where the weight comes from the class distribution.

Recall is the proportion of actual positives that are correctly classified. Its formula is $\frac{\text{True positive}}{\text{True positive}+\text{False negative}}$, where true positive is the number of real positive samples that are correctly classified, and false negative is the number of real positive samples that are incorrectly classified as negative. Each class will produce a recall value in the classification result. The overall recall is a weighted average of multiple recall values where the weight comes from the class distribution.

F1 score is a combination of the precision and recall by calculating their harmonic mean penalises the performance of classification when either the precision or recall is low. F1 score's formula is $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision}+\text{Recall}}$. Each class will produce a F1 score in the classification result. The overall F1 score is a weighted average of multiple F1 scores where the weight comes from the class distribution.

A confusion matrix is a table layout that visualises the classification performance. Each row represents an instance of the actual class, whereas each column represents an instance of the predicted class.[5] Therefore, the values of the diagonal elements represent the degree of correctly predicted classes.

Implementation details

The implementation of all practical work in this chapter is based on Pytorch [107]. The computer used to conduct experiments has an AMD Ryzen Threadripper 3990X 64-Core CPU, 2 Nvidia RTX 3090 GPU and 128Gb RAM.

4.1.3 Results

F1 score is the main focus of discussion in this chapter because it can better reflect the classification performance when the class distribution is imbalanced. If not specified in this chapter, the precision, recall and F1 score in the overall classification results are the weighted average ones.

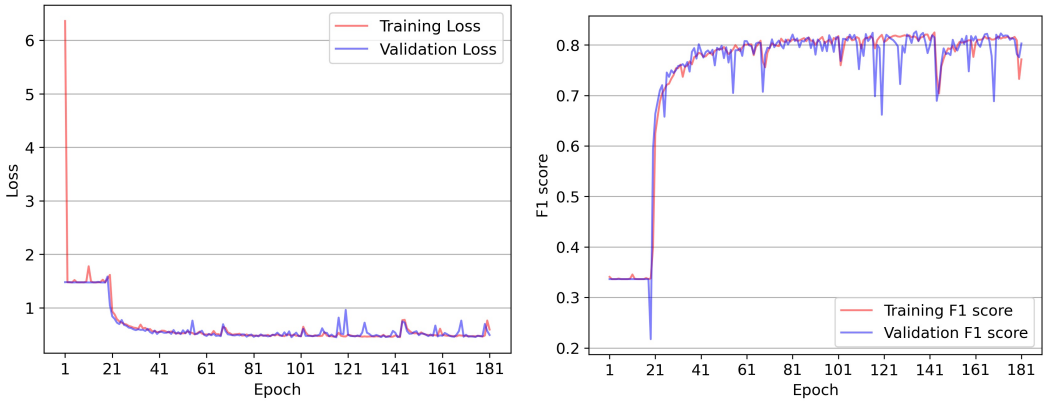
The overall classification performance of MicrobiaNet on MicrobiaS1, MicrobiaS2, MicrobiaS3, MicrobiaS4 and MicrobiaS5 data sets is detailed in Table 4.2. It is observed that MicrobiaNet can achieve an average training F1 score of 0.85 and an average validation F1 score of 0.82 across five different data splits. Among these training and validation F1 scores, they have a standard deviation of 0.0030 in the training F1 scores and a standard deviation

[5] A transposed variant of confusion matrix is often used in literature. The confusion matrix used in this thesis chooses the same form as introduced in [108].

Table 4.2 Overall evaluation results on MicrobiaS1, MicrobiaS2, MicrobiaS3, MicrobiaS4, and MicrobiaS5 data sets.

	Metric	Microbia Data Set Split with Different Seeds					Mean	Std
		S1	S2	S3	S4	S5		
Training	Precision	0.85	0.85	0.84	0.85	0.85	0.85	0.0010
	Recall	0.86	0.85	0.85	0.85	0.86	0.85	0.0039
	F1 score	**0.85**	**0.85**	**0.84**	**0.85**	**0.85**	**0.85**	**0.0030**
	Accuracy (%)	85.56	85.18	84.63	84.82	85.62	85.16	0.3900
Validation	Precision	0.82	0.83	0.83	0.83	0.81	0.82	0.0080
	Recall	0.83	0.84	0.83	0.83	0.82	0.83	0.0055
	F1 score	**0.82**	**0.83**	**0.83**	**0.83**	**0.81**	**0.82**	**0.0063**
	Accuracy (%)	82.79	83.52	83.27	83.39	82.00	82.99	0.5500

of 0.0063 in the validation F1 scores. This indicates that the estimate of MicrobiaNet's performance is robust regardless of the data on which it is trained. Additionally, the gap between each pair of training F1 score and validation F1 score never exceeds 0.03. This implies that the trained MicrobiaNet is able to generalise well on unseen data.



(a) Loss curves.

(b) F1 score curves.

Fig. 4.2 Loss value and F1 score throughout the training process obtained from MicrobiaS1 data set.

The performance of MicrobiaNet evaluated on MicrobiaS1 data set is chosen as the baseline performance. This is because its performance measured in F1 score is the closest to the average performance, as well as due to the importance of F1 score. As shown in Fig. 4.2, the gap between training loss and validation loss remains small throughout the training process obtained from MicrobiaS1 data set. The gap between training and validation F1 scores is also small in despite of the abrupt changes after 100 epochs. This suggests that the model is neither over-fitting nor under-fitting, which could be early terminated if

the validation result stops improving. The curves obtained from MicrobiaS2, MicrobiaS3, MicrobiaS4, and MicrobiaS5 data sets show a similar trend as illustrated in Fig. B.1, B.2, B.3, and B.4 in Appendix B.1.

Table 4.3 Classification results evaluated on MicrobiaS1 validation set.

Class Name	Precision	Recall	F1 score
One-colony	0.94	0.98	0.96
Two-colonies	0.82	0.81	0.82
Three-colonies	0.65	0.68	0.66
Four-colonies	0.48	0.38	0.43
Five-colonies	0.42	0.26	0.32
Six-colonies	0.64	0.61	0.63
Outlier	0.86	0.80	0.83

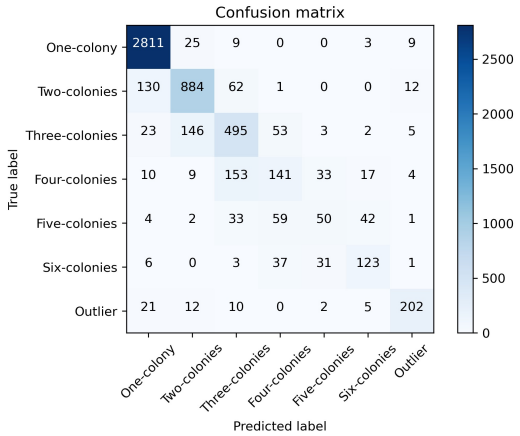


Fig. 4.3 Confusion matrix from MicrobiaS1 validation results.

The classification results of individual classes from the MicrobiaS1 validation set are presented in Table 4.3, calculated from the confusion matrix shown in Fig. 4.3. Table 4.3 shows that the minority classes, such as Four-colonies and Five-colonies as demonstrated in Table 3.2, have a F1 score that is dramatically lower than half of the F1 score from the majority One-colony class. This implies that the model is biased towards the majority class. However, Outlier class, which is another minority class, has a F1 score of 0.83 that is only 0.13 lower than the F1 score from the majority One-colony class. Additionally, only a small portion of the wrong classifications from minority classes (all classes except the One-colony class) are predicted as the majority One-colony class. For example, the majority of errors for Six-colonies are in Four-colonies and Five-colonies rather than One-colony; the majority of errors for Five-colonies are in Four-colonies and Six-colonies rather than One-colony. This

contradicts the implication that the model is biased towards the majority One-colony class observed from the comparison among Five-colonies class, Six-colonies class and One-colony class. This thus suggests that the model may suffer from more problems than imbalanced class distribution. This finding can also be observed from the classification results evaluated on MicrobiaS2, MicrobiaS3, MicrobiaS4 and MicrobiaS5 validation sets, which are detailed in Appendix B.1.

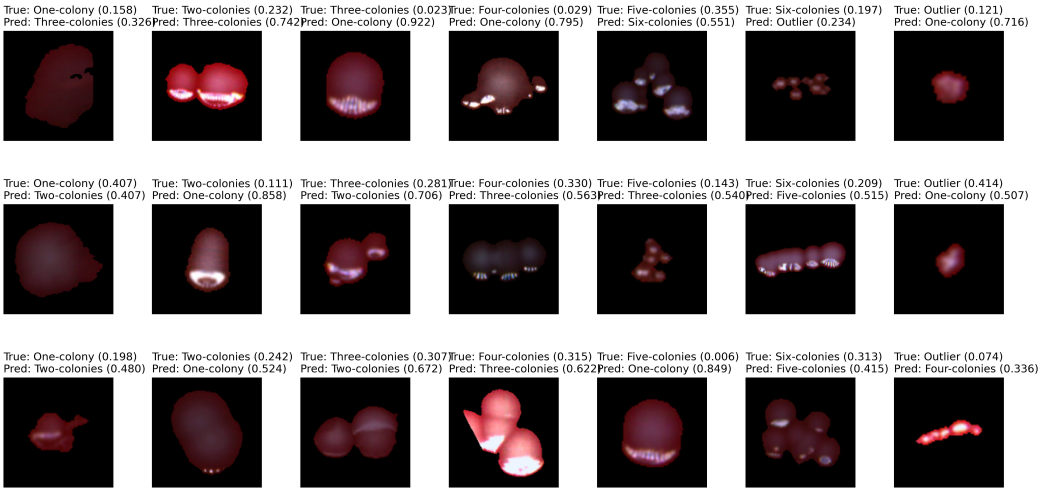


Fig. 4.4 Examples of incorrect predictions from MicrobiaS1 training set.

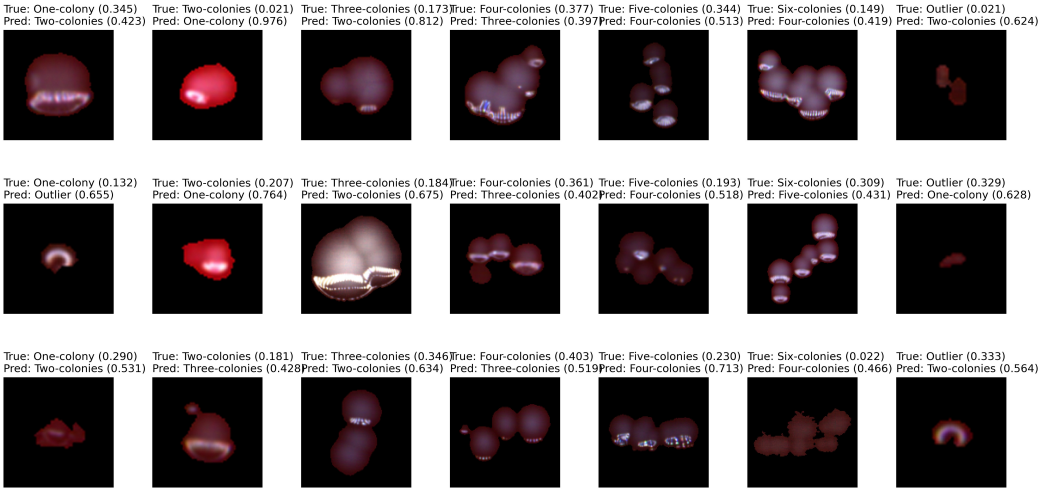


Fig. 4.5 Examples of incorrect predictions from MicrobiaS1 validation set.

Some misclassified examples from MicrobiaS1 training set and validation set are illustrated in Fig. 4.4 and 4.5. In these two figures, the probability of true label and the probability of predicted label are displayed above the colony image. It can be seen that many colony

images are predicted as their neighbouring classes. For example, the Two-colonies images in Fig. 4.4 are predicted as either Three-colonies or One-colonies; the Three-colonies images in Fig. 4.5 are all predicted as Two-colonies; the Four-colonies images in Fig. 4.5 are all predicted as Three-colonies; The Five-colonies images in Fig. 4.5 are all predicted as Four-colonies; This may be due to the high visual similarity across classes. Additionally, it can be observed that the colony image in the third row and fifth column in Fig. 4.4 is actually a One-colony image in spite of its true label of Five-colonies, suggesting that Microbia data set has mislabelled data.

4.1.4 Case study summary

This case study has empirically identified that MicrobiaNet is able to predict colony class with an average validation F1 score of 0.82 and a standard deviation of 0.0063 from five different data splits of the Microbia data set. This answers the research question "To what extent does MicrobiaNet address small and clustered colonies?". Additionally, it is uncovered that some minority classes have a significantly worse F1 score than the majority One-colony class, whereas the minority Outlier class's F1 score is only 0.13 lower than the majority One-colony class. It is further concluded that the trained MicrobiaNet may suffer from more problems than imbalanced class distribution. These problems may include the mislabelled data and high visual similarity across classes. Because the Microbia data set has 28418 images, it is time-consuming and expensive to identify/count/relabel mislabelled images. Compared with the mislabelled data, the high visual similarity across classes attributes more to the prediction error as shown in the confusion matrix. With the selection of baseline performance, case studies in the rest of this chapter will focus on the influence from imbalanced class distribution and other problems on the classification performance.

4.2 Interpretability of MicrobiaNet

This section aims to investigate the research question "What insights into MicrobiaNet can be gained by studying its interpretability?". Because no studies have investigated the interpretability of MicrobiaNet so far, some experiments are conducted to interpret it based on its baseline performance. The interpretability aims to provide an understanding of the feature space used by the network to make decisions, explain what features attribute to neuron activation in the network, and which part of the input image attributes to neuron activation in the network. If not specified, the trained MicrobiaNet in this section refers to the one

obtained from the baseline performance, i.e. MicrobiaS1 data set. This model is interpreted via the visualisation of network layer outputs, features and class activation maps.

4.2.1 Network layer output visualisation

Network layers often produce high-dimensional outputs which are difficult to interpret. In order to plot these outputs in a plane for visualisation, two dimensionality reduction techniques are used to reduce the high-dimensional data to two-dimensional data.

Principal Component Analysis

Principal Component Analysis (PCA) is a popular technique to analyse data sets that have many features. It allows users to find a set of vectors, i.e. principal components, that best match the description of the spread and direction of data across multiple dimensions, and thus select the top-n best-describing principal components to reduce the dimensionality of feature space in the data set. PCA consists of four steps:

1. Calculate orthonormal unit vectors to express the spread of data, where these vectors are orthogonal to each other.
2. Sort these vectors based on the importance which explains the variance of data along with the new feature dimension.
3. Reduce the number of dimensions to the most important ones (the top-n).
4. Project the original data set onto these new feature dimensions, i.e. principal components, so that the number of original dimensions is reduced without losing too much information.

The first step is achieved by Singular Value Decomposition (SVD). The top-2 important principal components are used here so that they can be plotted in a plane. Meanwhile, this case study focuses on the outputs from the last two layers because they are close to the final output. These two layers' outputs are produced by the MicrobiaNet trained on MicrobiaS1 training set. The second to last layer is the linear layer which consists of 500 hidden units activated by ReLU, and the last layer is the final linear layer which consists of 7 hidden units.

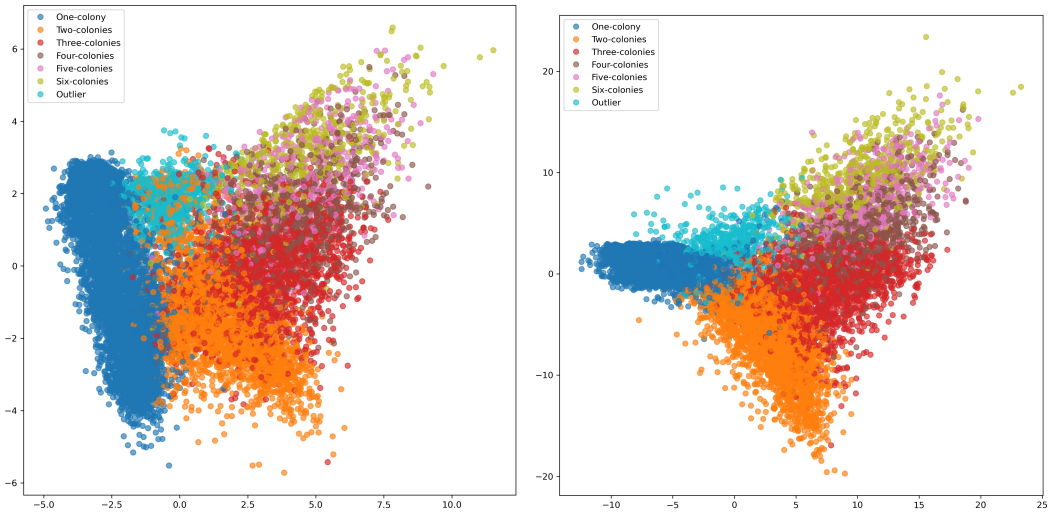
t-distributed Stochastic Neighbour Embedding

In addition to the PCA visualisation, t-distributed Stochastic Neighbour Embedding (t-SNE) is another popular technique to visualise high-dimensional neural network outputs. It remaps

each data point in the high-dimensional space to a low-dimensional space, where each point's nearby points are similar and more distant points are dissimilar [92]. This is achieved by converting similarities between data points to joint probabilities, followed by minimising the KL divergence between the joint probabilities of the mapped low-dimensional data and the high dimensional data. It is particularly useful when the data is non-linearly separable. Similar to PCA, t-SNE visualises high-dimensional outputs from the last two layers of the trained MicrobiaNet in a two-dimensional space.

Perplexity is a hyper-parameter in t-SNE to determine the number of close neighbours each point has, aiming to balance the attention between regional and global data. According to t-SNE's authors, typical values for perplexity are between 5 and 50. As suggested by Wattenberg et al. [146], this case study uses 2, 5, 30, 50, and 100 as different perplexity values, as well as 5000 iterations, to produce multiple t-SNE plots. Additionally, this case study chooses a learning rate that is the maximum value between 50 and the data set size divided by 48 as suggested by Pedregosa et al. [108].

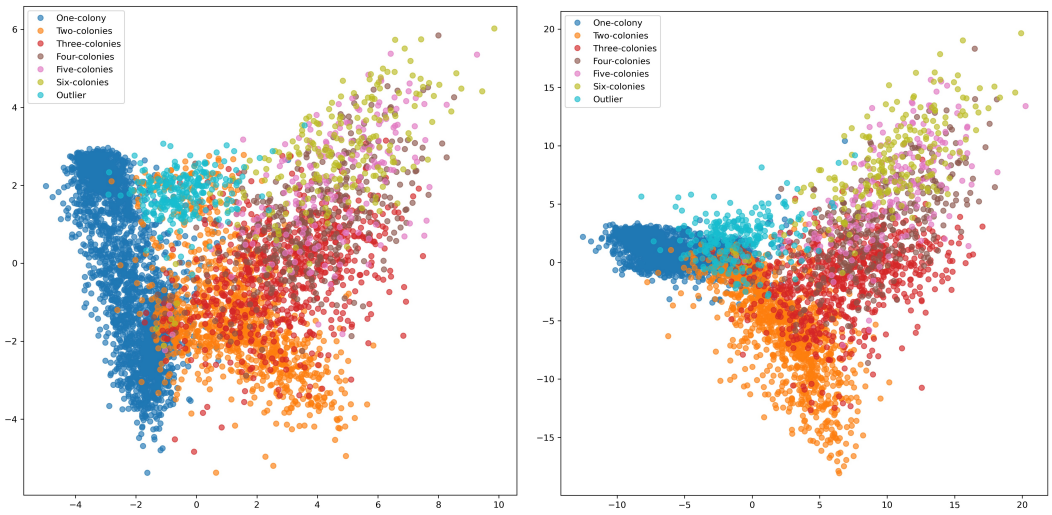
Results



(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.6 Visualisation of the last two network layer outputs from the baseline model evaluated on MicrobiaS1 training set with dimensionality reduced by PCA.

The last two network layer outputs from the baseline model evaluated on the MicrobiaS1 training set and validation set with PCA dimensionality reduction are depicted in Fig. 4.6 and 4.7 respectively. They both illustrate the imbalanced class distribution in which the One-colony class (the largest blue cluster) dominates. Meanwhile, it can be observed that

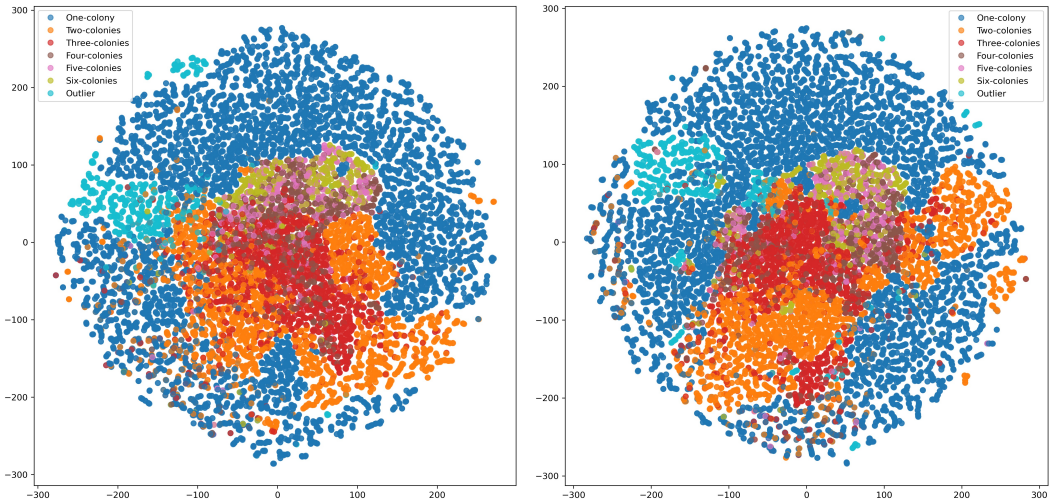


(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.7 Visualisation of the last two network layer outputs from the baseline model evaluated on MicrobiaS1 validation set with dimensionality reduced by PCA.

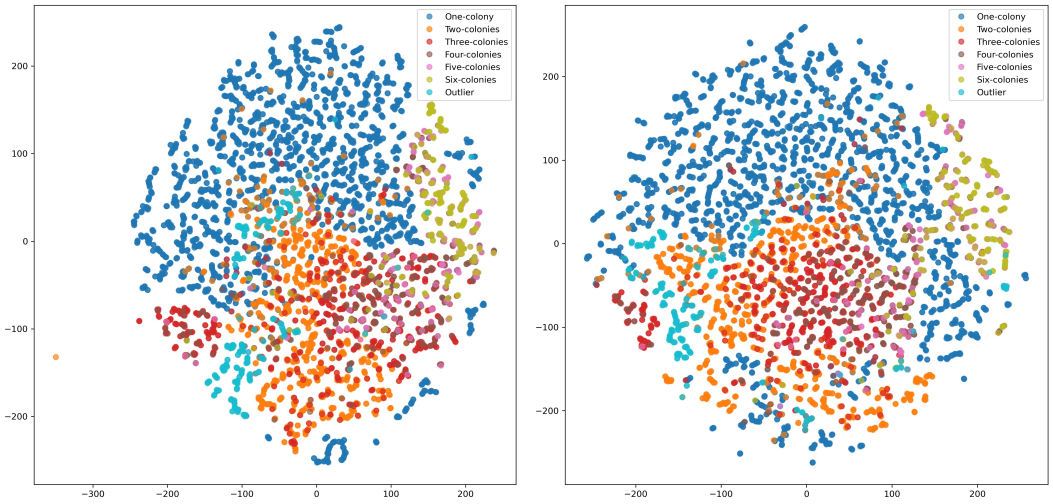
One-colony, Two-colonies and Outlier classes are distinct with their own clusters, which explains why they are the most three performant classes in Table 4.3. However, it is difficult to identify the distribution of Three-colonies, Four-colonies, Five-colonies, and Six-colonies classes as they are entangled and overlapped together in the feature space, leading to an inferior performance as shown in Table 4.3. Additionally, the least minority Five-colonies class scatters over the Three-colonies, Four-colonies and Six-colonies classes, suggesting it is the least separable class. However, it is still unknown if the class distribution in feature space is caused by class imbalance or high visual similarity across classes.

The visualisation of the last two network layer outputs from MicrobiaS1 training and validation sets with t-SNE dimensionality reduction of multiple perplexity values are presented in Fig. 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, and 4.17. It can be observed that the clusters found by t-SNE have a tendency towards clearer shapes with the increase of the perplexity value. This is because the larger perplexity value will retain more non-local information in the dimensionality reduction result, producing a denser cluster structure. More importantly, these figures show that One-colony, Two-colonies and Outlier classes have distinct clusters, whereas Three-colonies, Four-colonies, Five-colonies and Six-colonies classes are entangled and overlapped together. These findings are similar to these identified from PCA dimensionality reduction results, suggesting it is necessary to investigate class imbalance and high visual similarity separately.



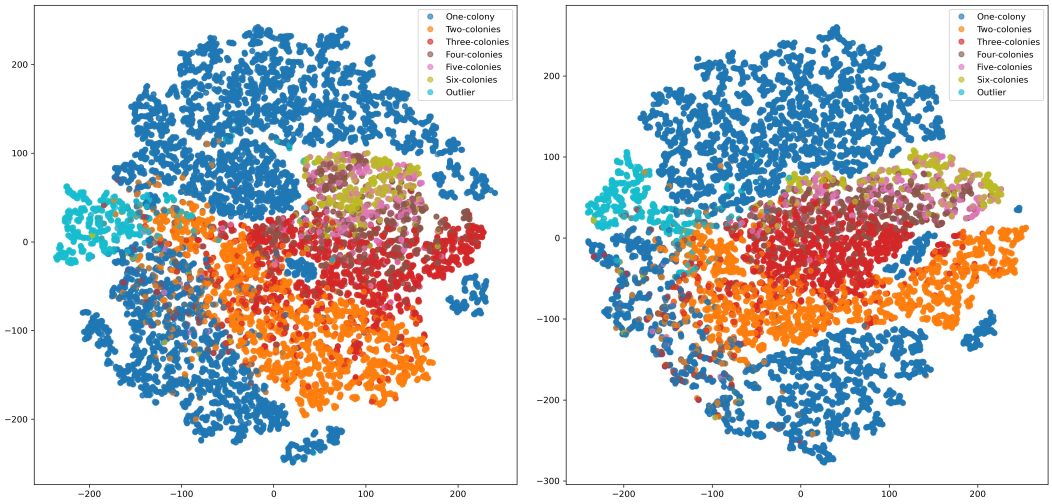
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.8 Visualisation of the last two network layer outputs from the baseline model evaluated on MicrobiaS1 training set with dimensionality reduced by t-SNE of 2 perplexity.



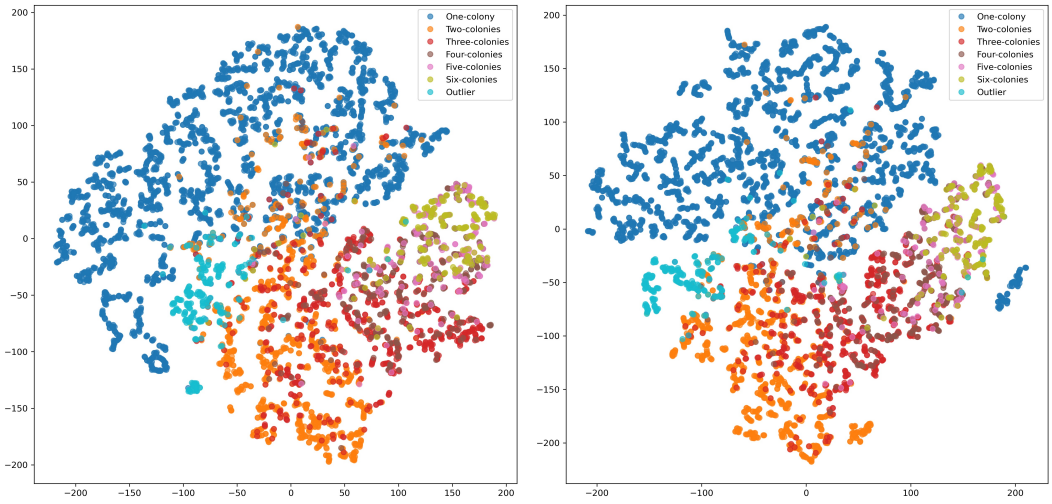
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.9 Visualisation of the last two network layer outputs from the baseline model evaluated on MicrobiaS1 validation set with dimensionality reduced by t-SNE of 2 perplexity.



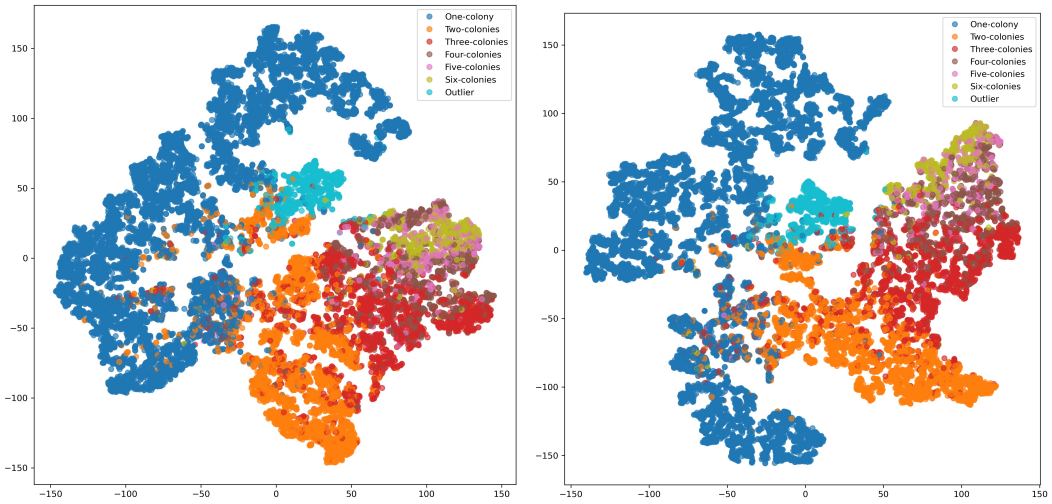
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.10 Visualisation of the last two network layer outputs from the baseline model evaluated on MicrobiaS1 training set with dimensionality reduced by t-SNE of 5 perplexity.



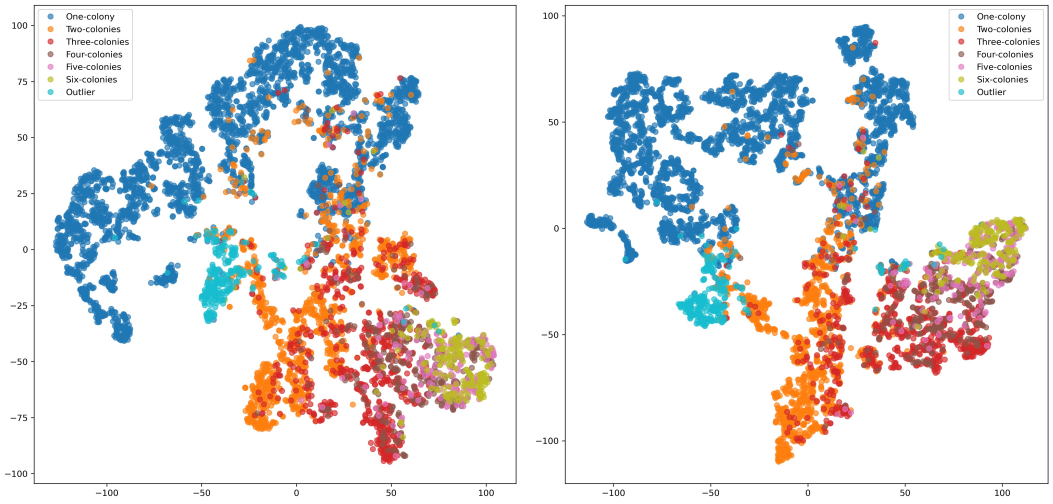
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.11 Visualisation of the last two network layer outputs from the baseline model evaluated on MicrobiaS1 validation set with dimensionality reduced by t-SNE of 5 perplexity.



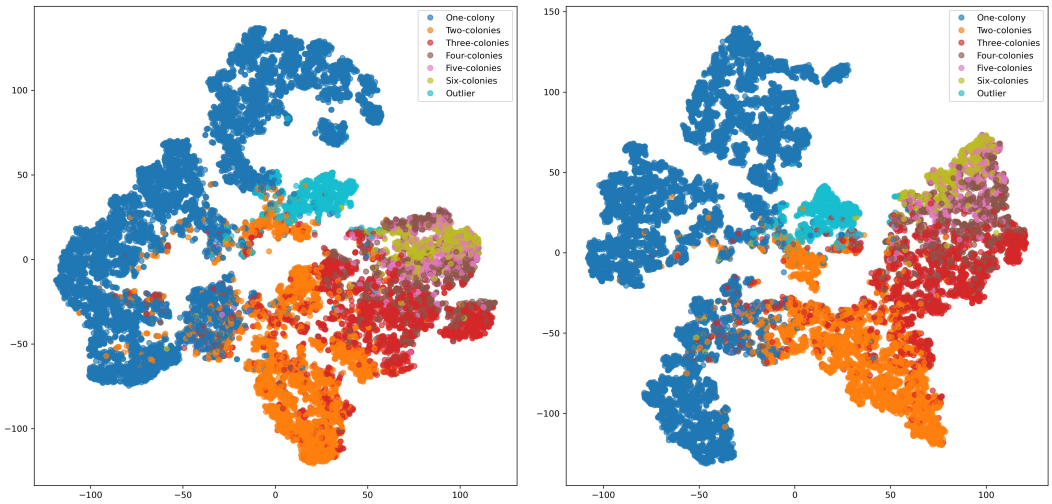
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.12 Visualisation of the last two network layer outputs from the baseline model evaluated on MicrobiaS1 training set with dimensionality reduced by t-SNE of 30 perplexity.



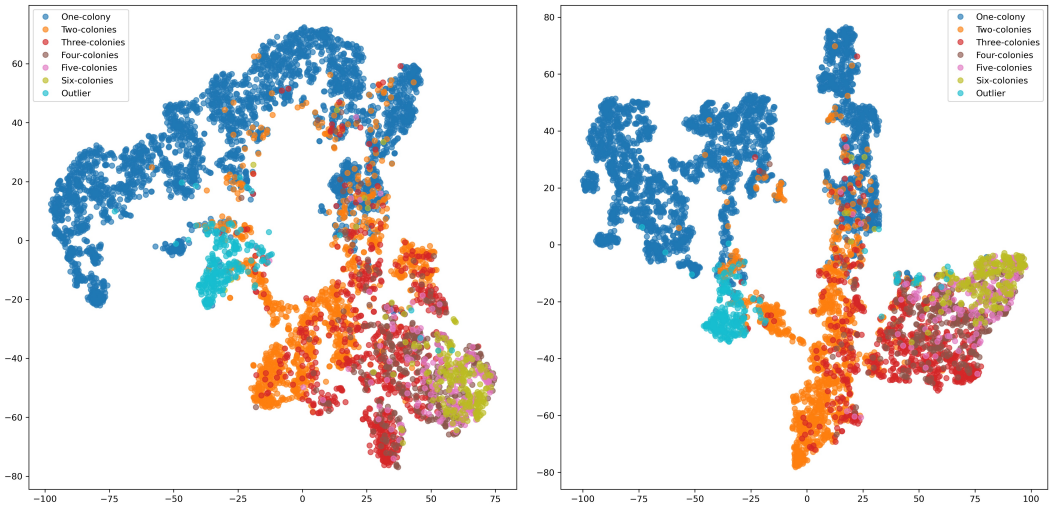
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.13 Visualisation of the last two network layer outputs from the baseline model evaluated on MicrobiaS1 validation set with dimensionality reduced by t-SNE of 30 perplexity.



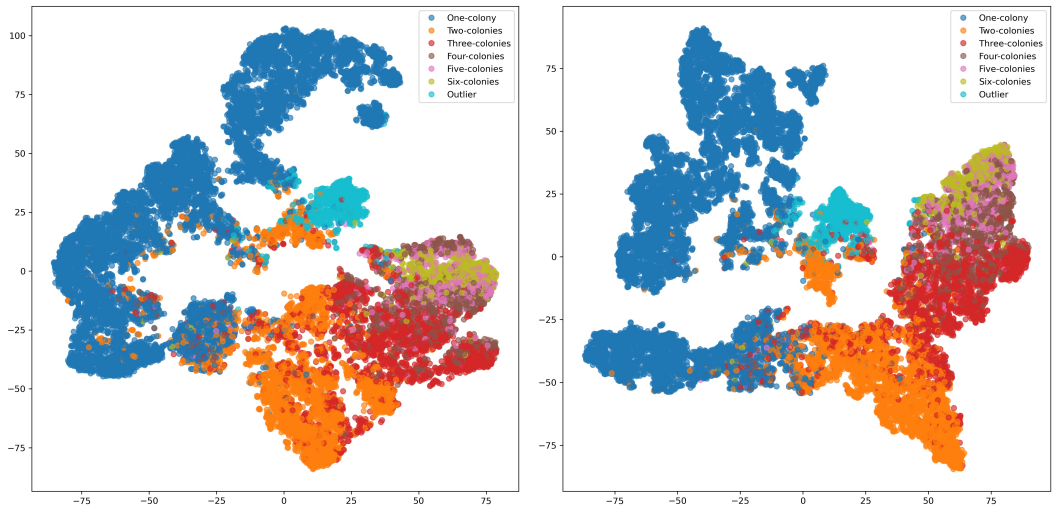
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.14 Visualisation of the last two network layer outputs from the baseline model evaluated on MicrobiaS1 training set with dimensionality reduced by t-SNE of 50 perplexity.



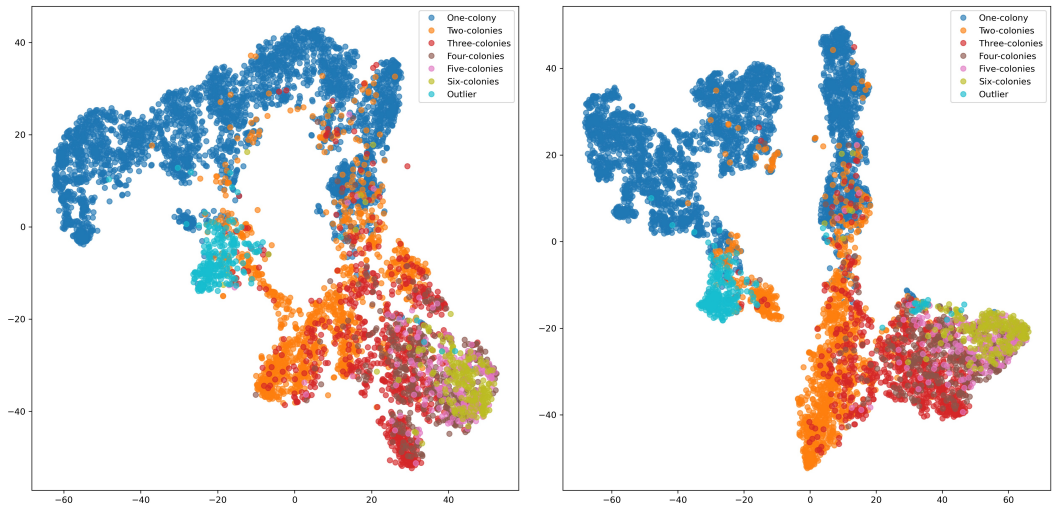
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.15 Visualisation of the last two network layer outputs from the baseline model evaluated on MicrobiaS1 validation set with dimensionality reduced by t-SNE of 50 perplexity.



(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.16 Visualisation of the last two network layer outputs from the baseline model evaluated on MicrobiaS1 training set with dimensionality reduced by t-SNE of 100 perplexity.



(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.17 Visualisation of the last two network layer outputs from the baseline model evaluated on MicrobiaS1 validation set with dimensionality reduced by t-SNE of 100 perplexity.

4.2.2 Feature visualisation

Feature visualisation aims to study what a neural network is looking for by generating example images that maximise neuron activation. This section focuses on the visualisation of trained convolutional kernels, aiming to search for what features in the input image activate the trained MicrobiaNet.

Feature visualisation can be achieved by optimisation. An image of $3 \times 128 \times 128$, which is the same size of MicrobiaNet's input, is initialised with random noise. It is input to the trained MicrobiaNet to reach the target convolutional kernel whose derivatives are back-propagated to the input image iteratively until they converge. In this experiment, 8 images are generated via 512 iterations of back-propagation with a learning rate of 0.05 to visualise some convolutional kernel. These hyper-parameters are suggested by Olah et al. [103] whose visualisation tool is also used in this chapter. Because MicrobiaNet consists of many convolutional kernels, this experiment only visualises the first, second, third, fourth, fifth, and sixth kernels in each convolutional layer for simplicity.

Results

Eight features (presented in eight columns) that activate the first kernels in four convolutional layers (presented in four rows) of the trained MicrobiaNet are illustrated in Fig. 4.18. It is difficult to perceive any colony information intuitively, such as shape and size, from these visualised features. However, some kernels in the convolutional layers, such as the first row in these six figures, have been able to detect some texture. Additionally, it can be seen that the first convolutional kernel in the fourth convolutional layer (the fourth row in Fig. 4.18) has been trained to detect some red blobs which are vaguely similar to the shape of colonies. Meanwhile, features that activate the second, third, fourth, fifth, and sixth kernels in four convolutional layers of the network also show a similar trend as shown in Fig. 4.19, 4.20, 4.21, 4.22, and 4.23. These less informative visualisations match the claim made by Salahuddin et al. [124] that it is hard to interpret structures and patterns in medical images because they are not very obvious.

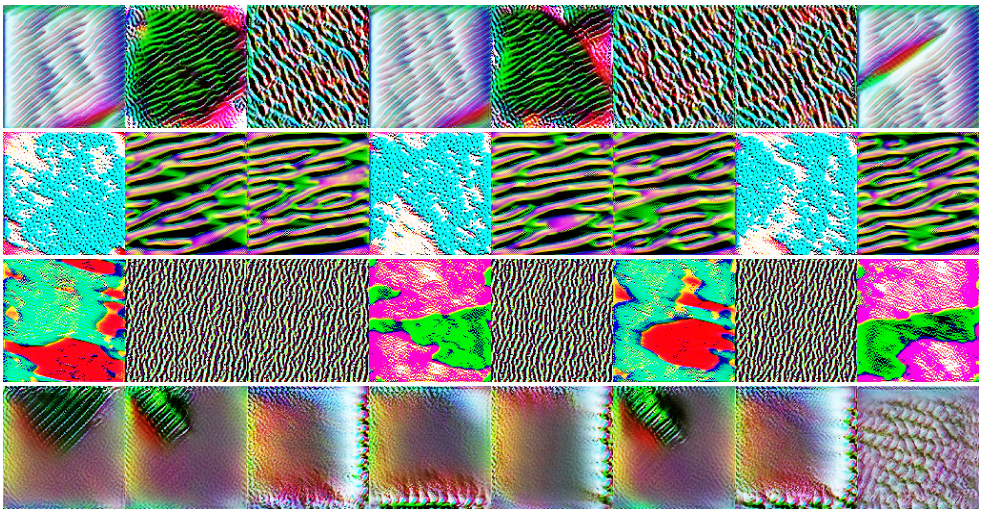


Fig. 4.18 Visualisation of the first convolutional kernels in the trained MicrobiaNet's every convolutional layer.

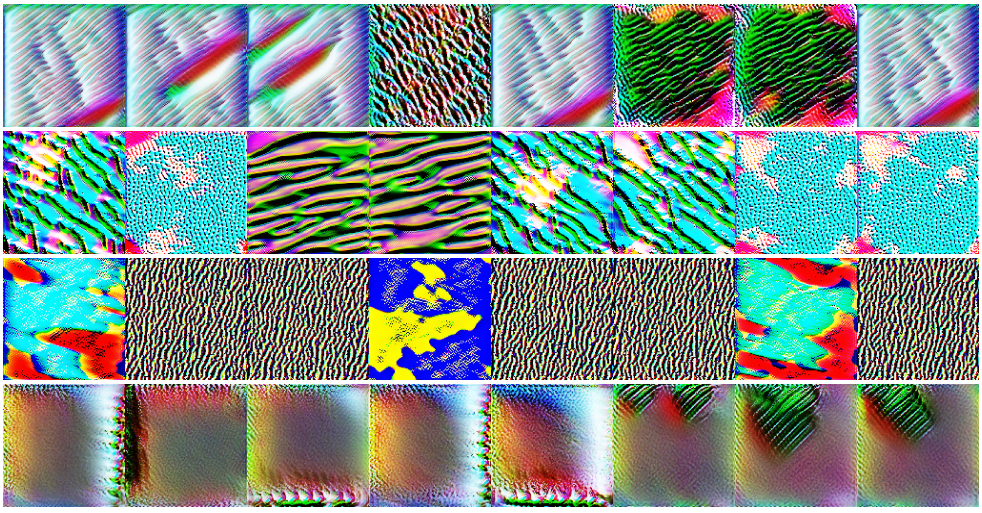


Fig. 4.19 Visualisation of the second convolutional kernels in the trained MicrobiaNet's every convolutional layer.

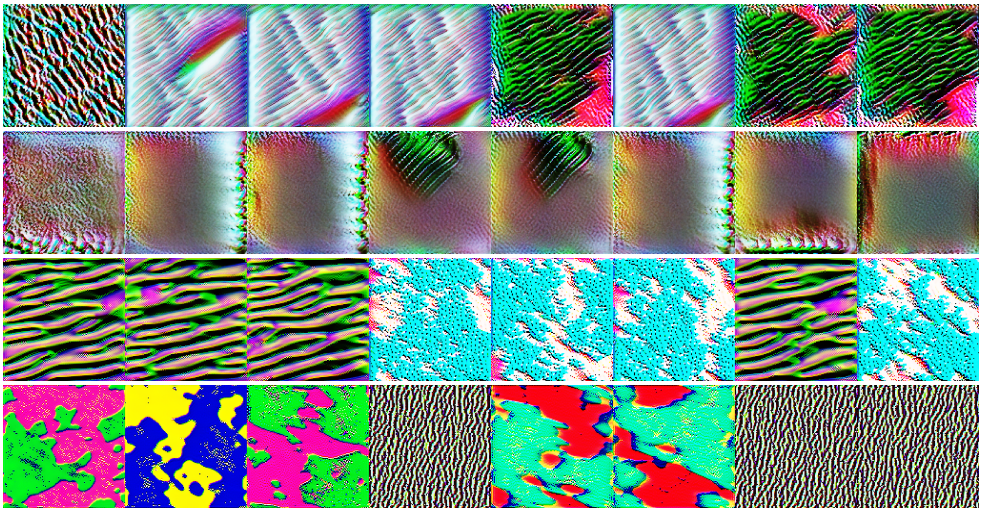


Fig. 4.20 Visualisation of the third convolutional kernels in the trained MicrobiaNet's every convolutional layer.

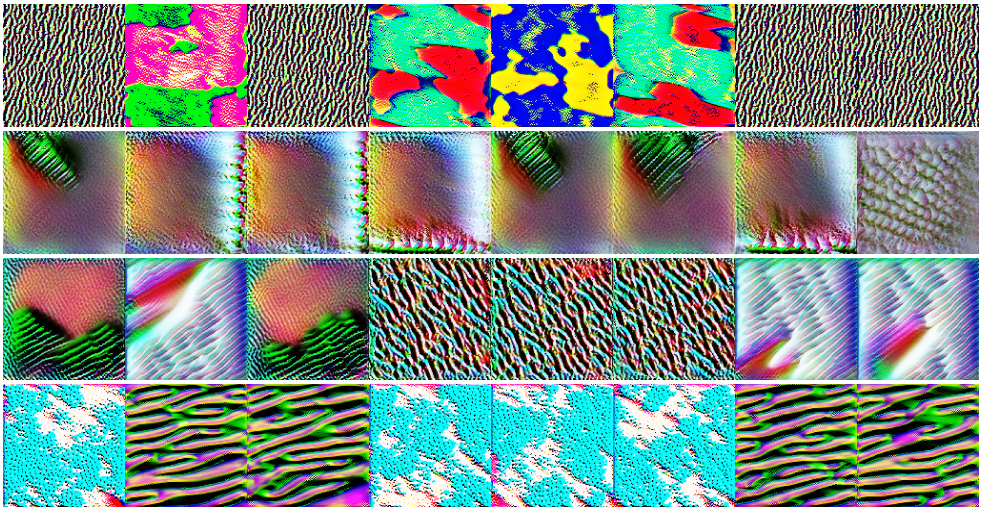


Fig. 4.21 Visualisation of the fourth convolutional kernels in the trained MicrobiaNet's every convolutional layer.

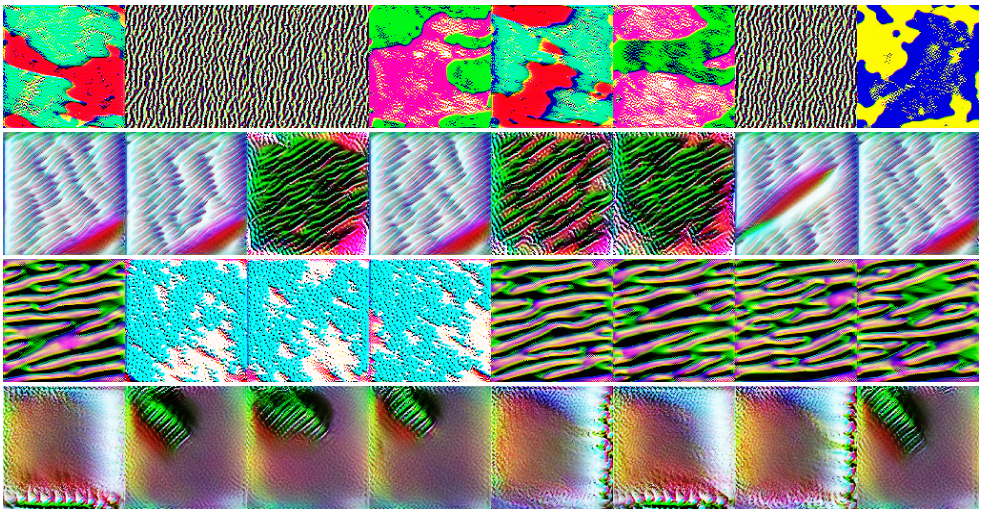


Fig. 4.22 Visualisation of the fifth convolutional kernels in the trained MicrobiaNet's every convolutional layer.

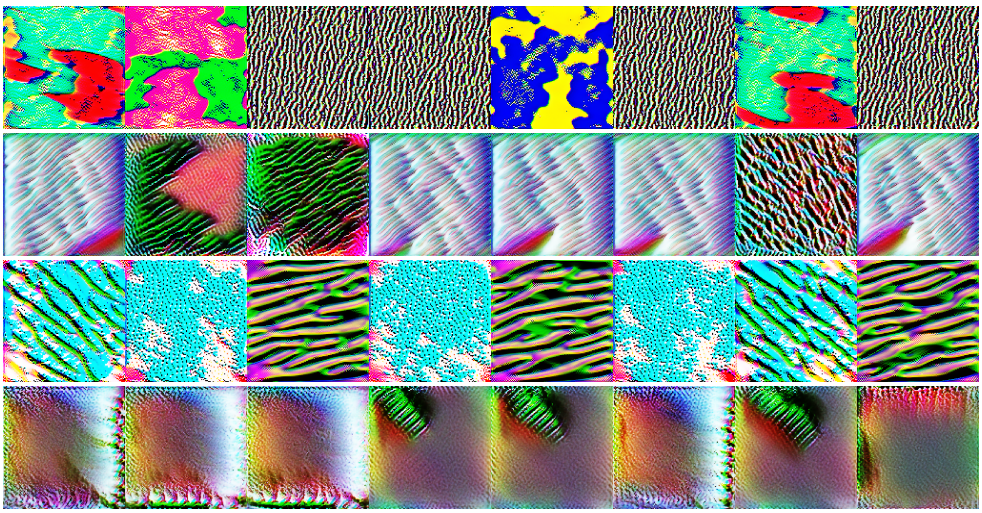


Fig. 4.23 Visualisation of the sixth convolutional kernels in the trained MicrobiaNet's every convolutional layer.

4.2.3 Class activation map visualisation

Class activation map visualisation is a study of what part of an example image activates neurons in the network. It was first introduced by Zhou et al. [166] and sometimes known as feature attribution visualisation. Their method, called CAM, inserts a global average pooling layer between the last convolutional layer and the fully connected layer. Then, the weights of the final network output layer with regard to a specific class are projected back to each convolutional feature map so that the importance of different image regions with regard to the specified class can be identified. Because CAM alters network architectures, it has been overtaken by a generalised method called Grad-CAM [128]. Grad-CAM computes the gradient of the final network output with regard to a specific class to the final convolutional layer in the network, followed by an average pooling operation to obtain the neuron importance weights. These neuron importance weights, i.e. gradients, are multiplied by the feature map in the target convolutional layer, followed by ReLU to keep positive values. Grad-CAM is generalisable to any convolutional layers in a neural network. This case study focuses on the last convolutional layer because it contains the richest semantic information with regard to the specified class.

In addition to the use of Grad-CAM to visualise class activation maps, this chapter also uses GradCAM++ [16], HiResCAM [35], XGradCAM [42], EigenCAM [100] and EigenGradCAM because they have proved useful for some data sets. Among them, GradCAM++ is similar to GradCAM but the second order gradients are used. HiResCAM is also similar to GradCAM but the gradients are element-wise multiplied by the feature map in the target convolutional layer. XGradCAM is similar to GradCAM but the feature map is normalised before multiplying the gradients. EigenCAM focuses on the first principal component of the feature map, whereas the EigenGradCAM discriminates class by multiplying the first principal component of the feature map and the gradients which are computed from the specified class score with respect to the convolutional layer.

This case study focuses on the visualisation of the activation map with regard to its real class label from the trained MicrobiaNet. The example images required for visualisation are randomly selected from the MicrobiaS1 training set for simplicity. The implementation of these visualisation techniques is facilitated by an open-source library [46].

Results

The visualised activation maps for each class in MicrobiaS1 training set are illustrated in Fig. 4.24, 4.25, 4.26, 4.27, 4.28, 4.29, 4.30. Among them, GradCAM has an inferior performance because its class activation maps have a small degree of overlap with the target

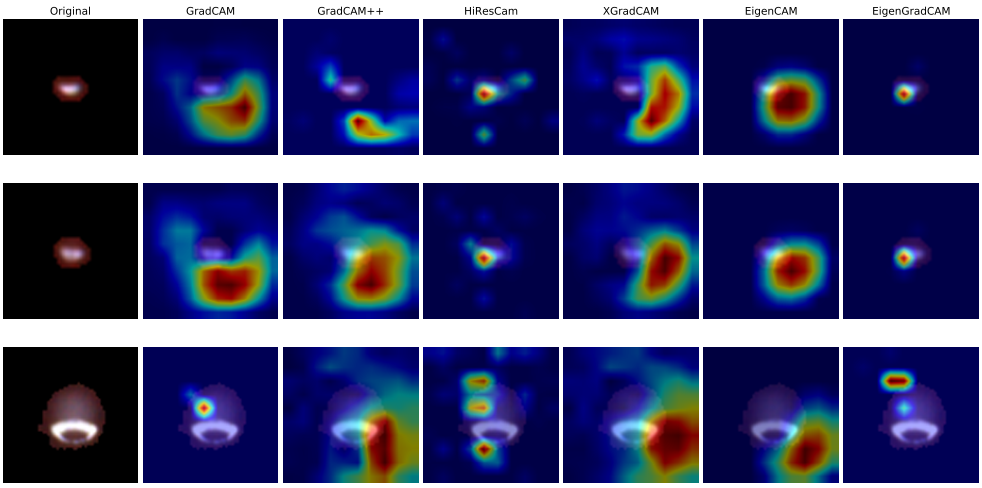


Fig. 4.24 Class activation map visualisation for One-colony images.

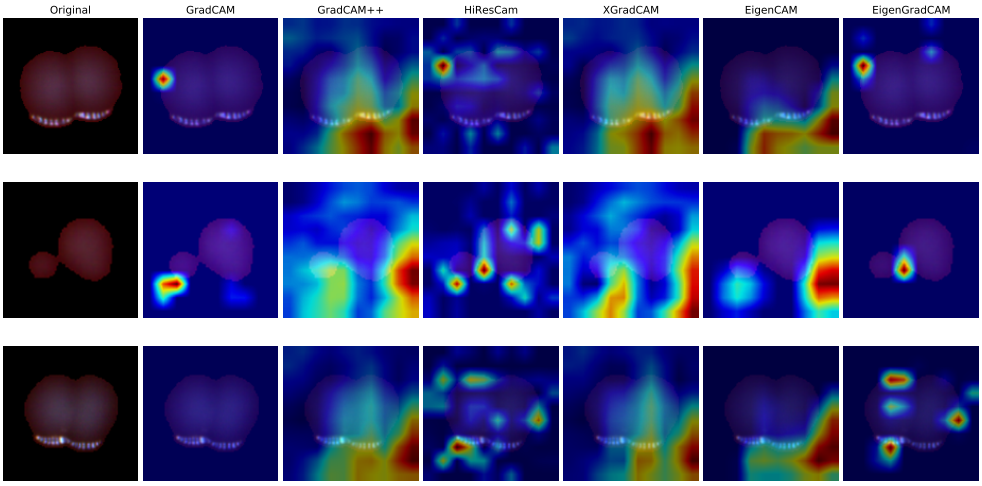


Fig. 4.25 Class activation map visualisation for Two-colonies images.

colony. Contrary to the degraded performance from GradCAM, GradCAM++, HiResCAM, XGradCAM, EigenCAM, and EigenGradCAM produce activation maps that are better overlapped with colonies, where the EigenCAM produces the best maps based on human inspection. This suggests that the first principal component of the feature map is useful for locating the target colony. However, none of these visualised activation maps show a direct link between the colony and its class.

4.2.4 Case study summary

This case study has investigated the interpretability of MicrobiaNet based on its baseline performance by visualising network layer outputs, features and class activation maps. The main

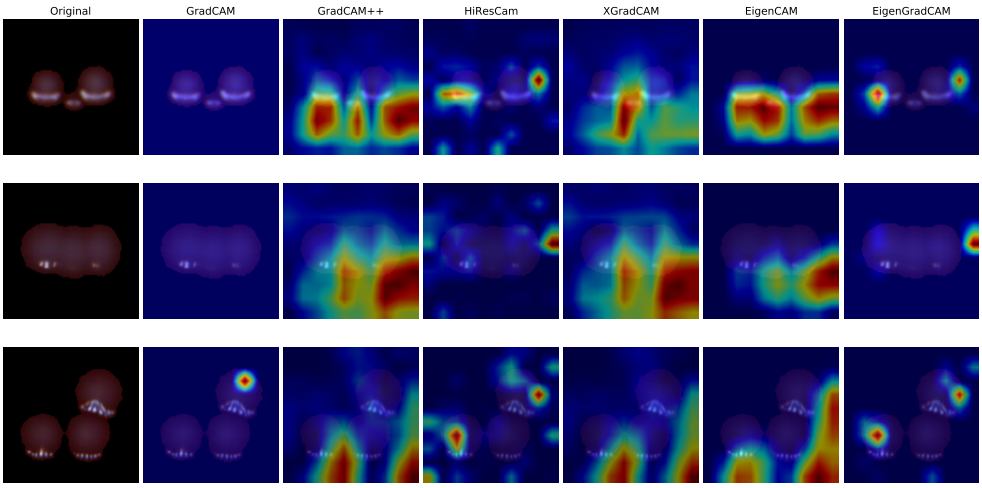


Fig. 4.26 Class activation map visualisation for Three-colonies images.

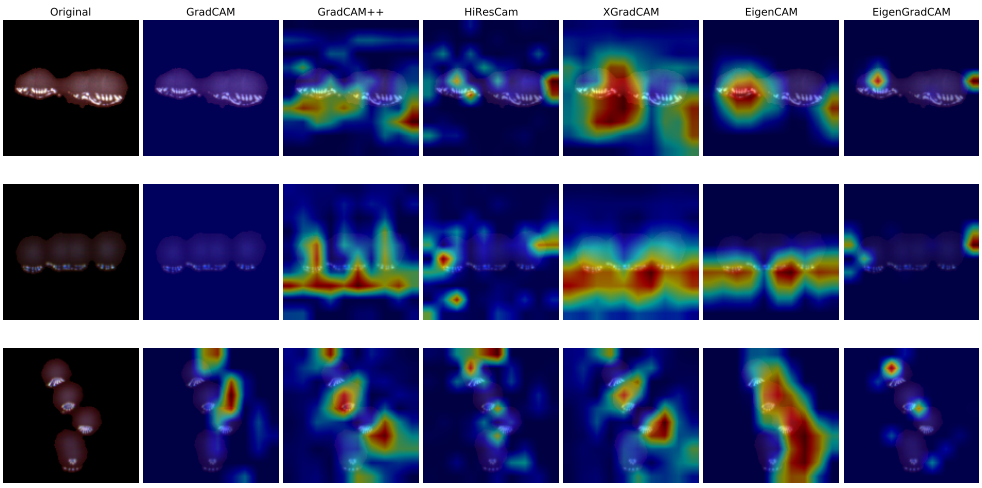


Fig. 4.27 Class activation map visualisation for Four-colonies images.

insights into MicrobiaNet gained by studying its interpretability with PCA dimensionality reduction and t-SNE dimensionality reduction are that One-colony, Two-colonies and Outlier classes are distinct with their own clusters in the feature space, whereas Three-colonies, Four-colonies, Five-colonies, and Six-colonies classes are entangled and overlapped together in the feature space. However, it is unknown if the poor performance is due to the class imbalance or high visual similarity across these classes. Additionally, neither feature visualisation nor class activation map visualisation is capable of providing an informative interpretation of predictions. Despite that, the first kernel in MicrobiaNet's fourth convolutional layer is capable of detecting red blobs which are vaguely similar to the shape of colonies. Meanwhile, the first component in the feature map is useful for locating target colonies as proved by the

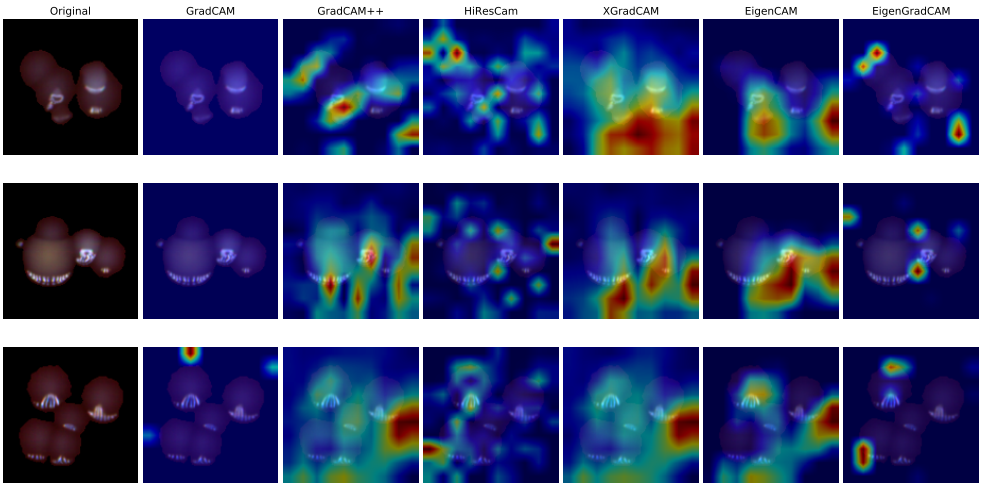


Fig. 4.28 Class activation map visualisation for Five-colonies images.

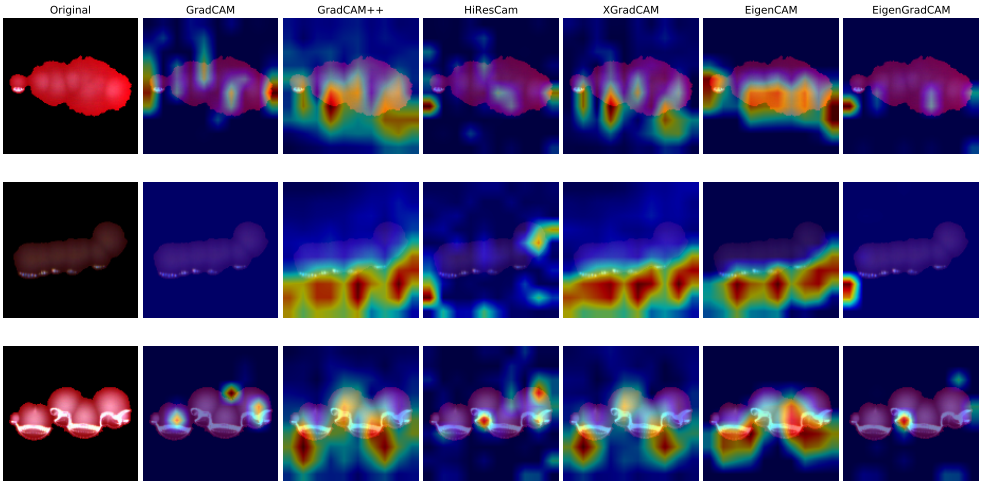


Fig. 4.29 Class activation map visualisation for Six-colonies images.

EigenCAM visualisations. This case study has not applied other methods to interpret the model before disentangling the class imbalance and high similarity across classes.

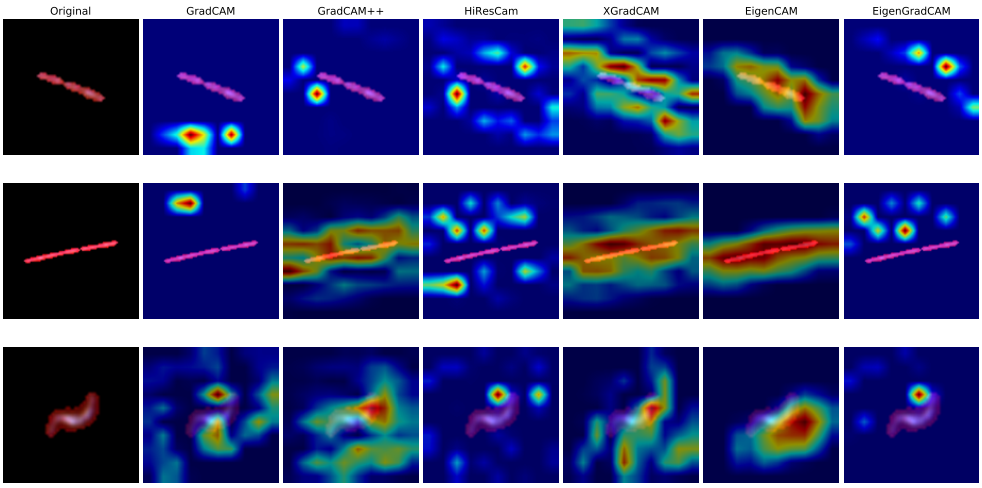


Fig. 4.30 Class activation map visualisation for Outlier images.

4.3 Analysis of class imbalance and high visual similarity

This case study aims to investigate the research question "What has been the impact of class imbalance and high visual similarity on MicrobiaNet?". Specifically, the high visual similarity is the high image similarity across classes. Concretely, MicrobiaNet, illustrated in Fig. 4.1, will be trained on a data set that has a balanced class distribution. The model trained on a balanced data set is referred to as balanced MicrobiaNet model to avoid confusion. The interpretability of the balanced MicrobiaNet model will be given as well.

4.3.1 Analysis of the class imbalance by data downsampling

Data downsampling is a very common method to tackle class imbalance by randomly removing extra data points from each class except the least minority class. It is used in this section due to its simplicity. Another common method called data upsampling, which randomly duplicates data points from each class except the majority class, is not considered here to avoid repetitive features in the feature space during the interpretation process. This means images in One-colony, Two-colonies, Three-colonies, Four-colonies, Six-colonies, and Outlier classes are randomly removed so that these classes have the same number of images as the Five-colonies class does.

The data downsampling is only applied to MicrobiaS1 training set with five different seeds when randomly removing the extra data points. These five balanced training sets are used to train MicrobiaNet individually to produce a robust evaluation. The trained balanced MicrobiaNet model is evaluated on MicrobiaS1 validation set, which is still imbalanced. The MicrobiaS1 set whose training set has data downsampled with a seed is referred to as

MicrobiaS1B1 data set where B indicates balanced and the last 1 indicates the seed for data downsampling.[6] The training and evaluation follows the same procedures as explained in § 4.1.2. Similar to the previous case studies in § 4.1 and § 4.2, one of these five trained balanced MicrobiaNet models will be chosen to interpret by visualising network layer outputs. The visualisation of features and class activation maps are not considered here as § 4.2 concludes that they are not informative.

Results

Table 4.4 Overall evaluation results on MicrobiaS1B1, MicrobiaS1B2, MicrobiaS1B3, MicrobiaS1B4, and MicrobiaS1B5 data sets.

	Metric	MicrobiaS1 Balanced with Different Seeds					Mean	Std
		B1	B2	B3	B4	B5		
Training	Precision	0.87	0.93	0.68	0.76	0.88	0.82	0.0915
	Recall	0.86	0.93	0.68	0.76	0.87	0.82	0.0910
	F1 score	**0.86**	**0.93**	**0.67**	**0.75**	**0.87**	**0.82**	**0.0936**
	Accuracy (%)	86.41	92.99	67.58	75.98	87.39	82.07	9.1000
Validation	Precision	0.77	0.77	0.75	0.76	0.77	0.77	0.0071
	Recall	0.76	0.76	0.75	0.76	0.76	0.76	0.0045
	F1 score	**0.76**	**0.77**	**0.75**	**0.76**	**0.76**	**0.76**	**0.0050**
	Accuracy (%)	76.27	76.21	75.12	76.37	75.95	75.99	0.0045

The overall evaluation results from MicrobiaNet trained on MicrobiaS1B1, MicrobiaS1B2, MicrobiaS1B3, MicrobiaS1B4, and MicrobiaS1B5 data sets are listed in Table 4.4. It can be observed that the balanced MicrobiaNet model achieves a mean of 0.82 in training F1 score and 0.76 in validation F1 score where the former has a standard deviation of 0.0936 and the latter has a standard deviation of 0.0050 across these five different data sets. Comparing with the results when the model is trained on imbalanced data shown in Table 4.2, the data downsampling method is the cause for a marginal reduction of 0.06 in the validation F1 score, despite the significant reduction of training data. This suggests that class imbalance plays a limited role in the baseline performance.

The balanced MicrobiaNet model trained on the MicrobiaS1B1 training set is selected for interpretation. This is because its training and validation F1 scores are the closest to the mean scores. The detailed training and validation results are shown in Table 4.5 and 4.6 which are computed from their corresponding confusion matrix shown in Fig. 4.31 and 4.32. It is discovered that One-colony, Two-colonies and Outliers classes are still the top-3 performing classes. Additionally, the F1 scores from training results have a standard deviation of 0.0668,

[6] MicrobiaS1B1 validation set is identical to MicrobiaS1 validation set.

Table 4.5 Classification results evaluated on MicrobiaS1B1 training set.

Class Name	Precision	Recall	F1 score
One-colony	0.85	0.98	0.91
Two-colonies	0.86	0.94	0.90
Three-colonies	0.80	0.80	0.80
Four-colonies	0.88	0.71	0.78
Five-colonies	0.78	0.83	0.80
Six-colonies	0.91	0.83	0.87
Outlier	0.99	0.97	0.98

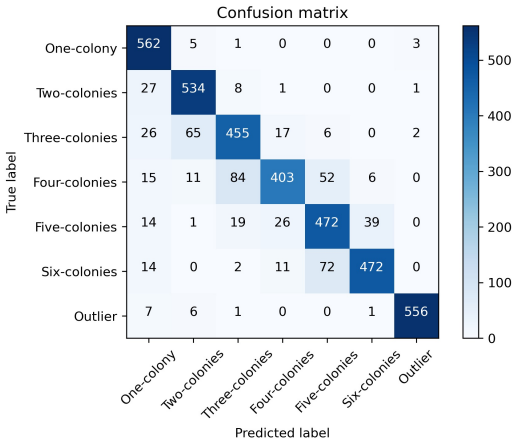


Fig. 4.31 Confusion matrix from MicrobiaS1B1 training results.

Table 4.6 Classification results evaluated on MicrobiaS1(B1) validation set.

Class Name	Precision	Recall	F1 score
One-colony	0.94	0.94	0.94
Two-colonies	0.76	0.73	0.74
Three-colonies	0.58	0.48	0.52
Four-colonies	0.33	0.25	0.28
Five-colonies	0.24	0.48	0.32
Six-colonies	0.54	0.50	0.52
Outlier	0.66	0.88	0.76

whereas the standard deviation of F1 scores from validation results is 0.2232. This suggests that the class imbalance has a very limited impact on the model. Since most of the wrong predictions come from the actual class's neighbouring classes as shown Fig. 4.31 and 4.32, the high image similarity across classes is the major problem to be solved for future performance improvement.

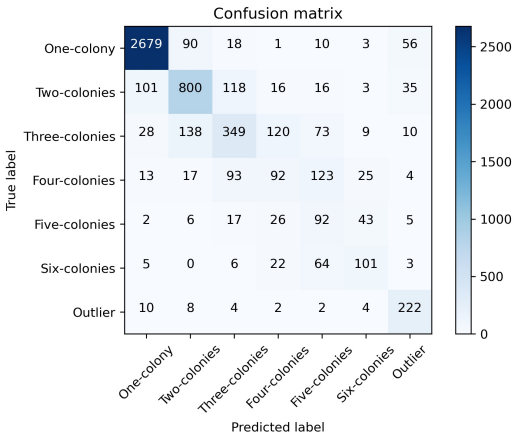
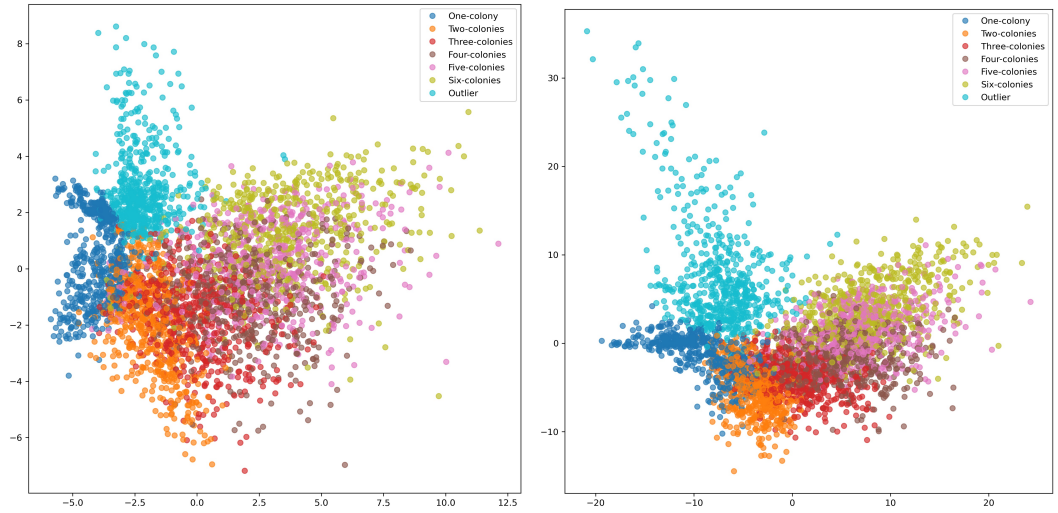


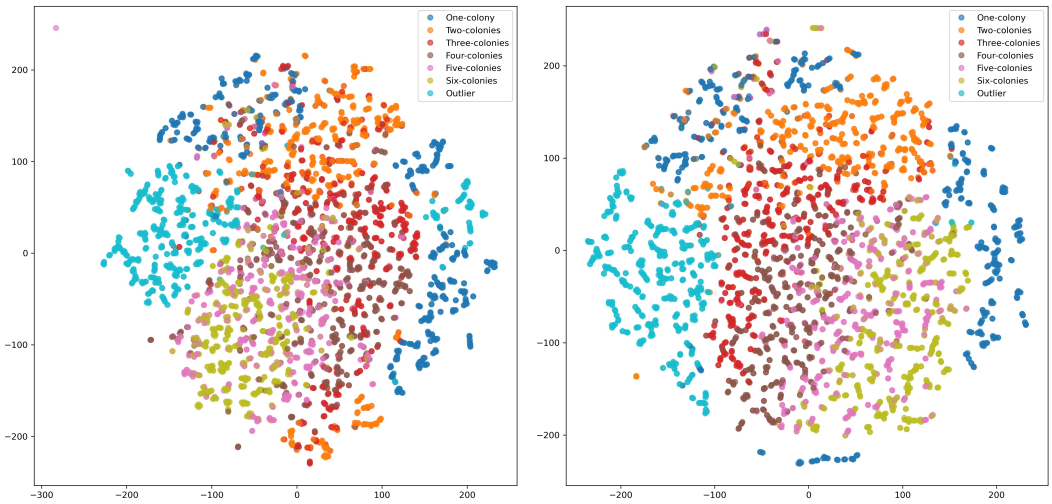
Fig. 4.32 Confusion matrix from MicrobiaS1(B1) validation results.



(a) The second to last network layer output. (b) The last network layer output.

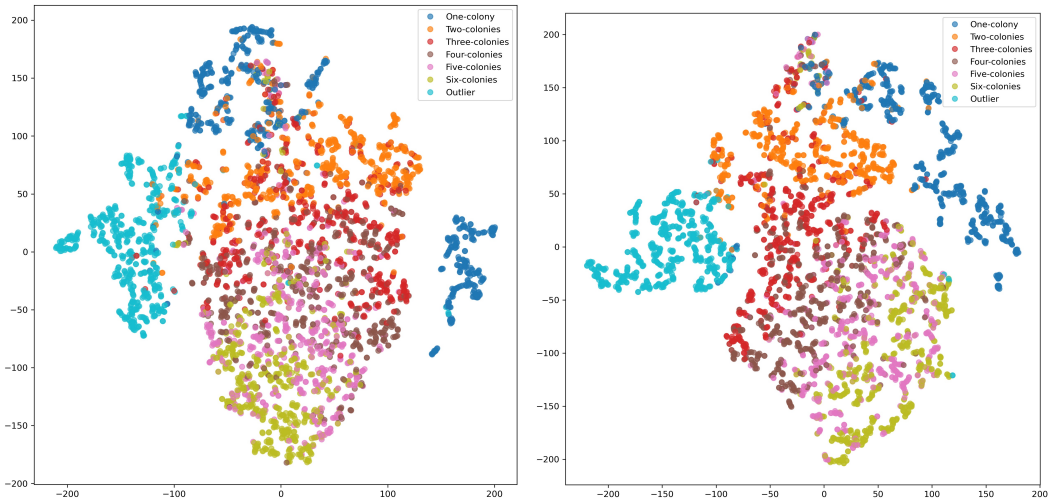
Fig. 4.33 Visualisation of the last two network layer outputs from the balanced MicrobiaNet model evaluated on MicrobiaS1B1 training set with dimensionality reduced by PCA.

According to Fig. 4.33, 4.34, 4.35, 4.36, 4.37, and 4.38, the network layer outputs from balanced MicrobiaS1B1 training set show that One-colony, Two-colonies and Outlier are distinct with their own clusters regardless of the dimensionality reduction method, whereas Three-colonies, Four-colonies, Five-colonies, and Six-colonies are entangled and overlapped together. The same pattern can be observed from the network layer outputs obtained from MicrobiaS1(B1) validation sets illustrated in Fig. 4.39, 4.40, 4.41, 4.42, 4.43, and 4.44. Because the model is trained on a balanced data set, this pattern further suggests that the high image similarity is the main problem to be solved for future performance improvement rather than the class imbalance.



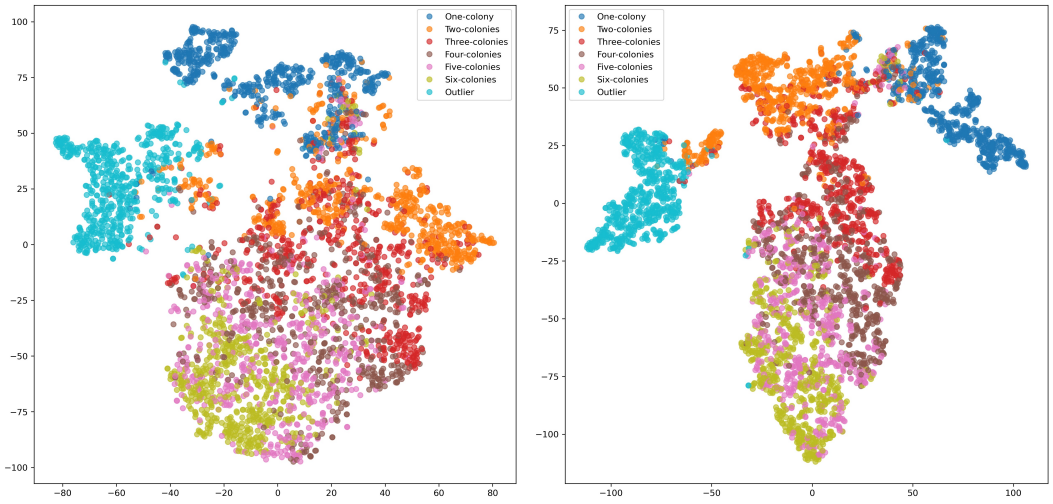
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.34 Visualisation of the last two network layer outputs from the balanced MicrobiaNet model evaluated on MicrobiaS1B1 training set with dimensionality reduced by t-SNE of 2 perplexity.



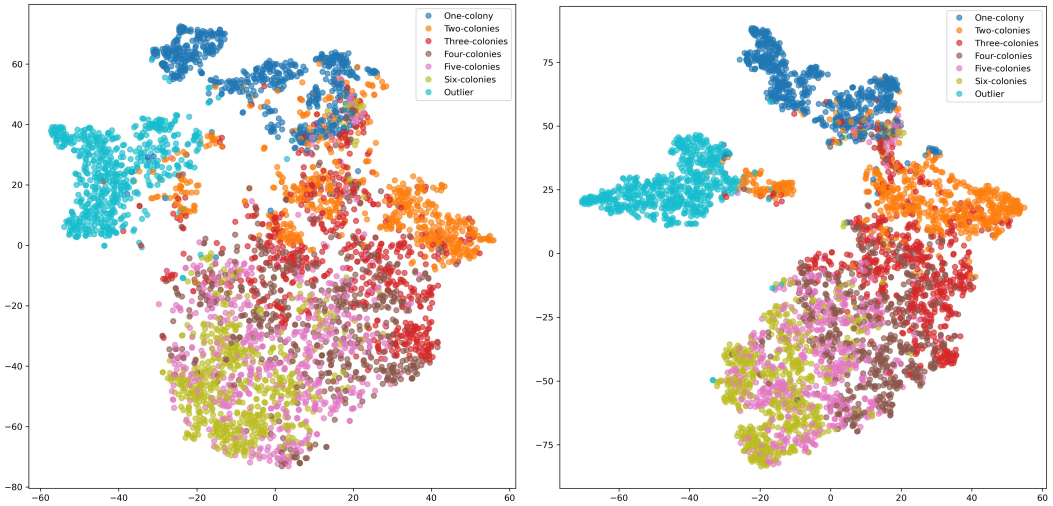
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.35 Visualisation of the last two network layer outputs from the balanced MicrobiaNet model evaluated on MicrobiaS1B1 training set with dimensionality reduced by t-SNE of 5 perplexity.



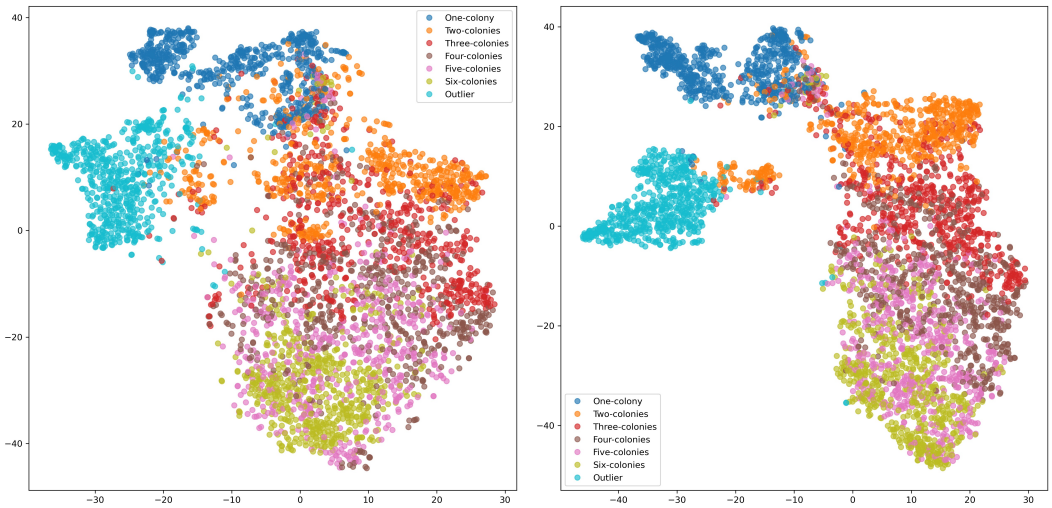
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.36 Visualisation of the last two network layer outputs from the balanced MicrobiaNet model evaluated on MicrobiaS1B1 training set with dimensionality reduced by t-SNE of 30 perplexity.



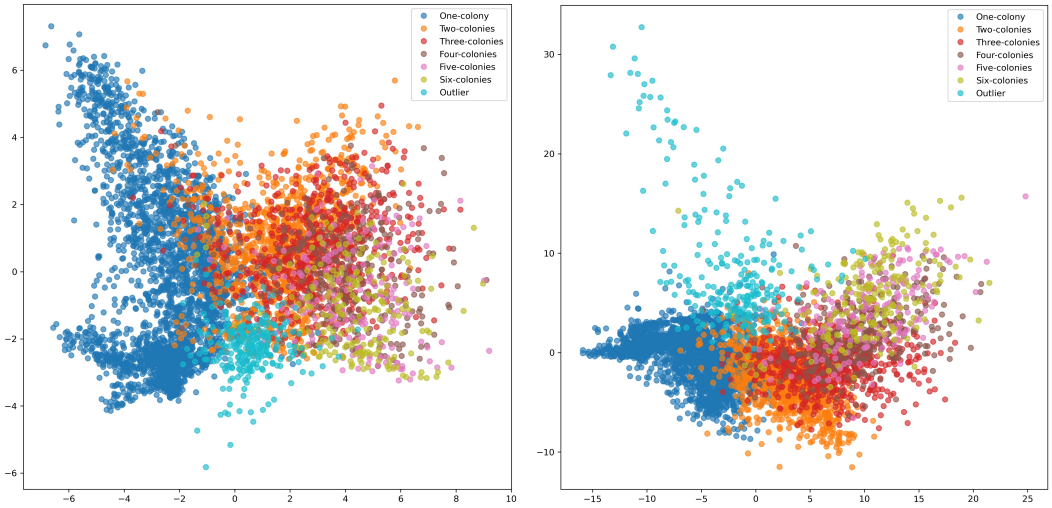
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.37 Visualisation of the last two network layer outputs from the balanced MicrobiaNet model evaluated on MicrobiaS1B1 training set with dimensionality reduced by t-SNE of 50 perplexity.



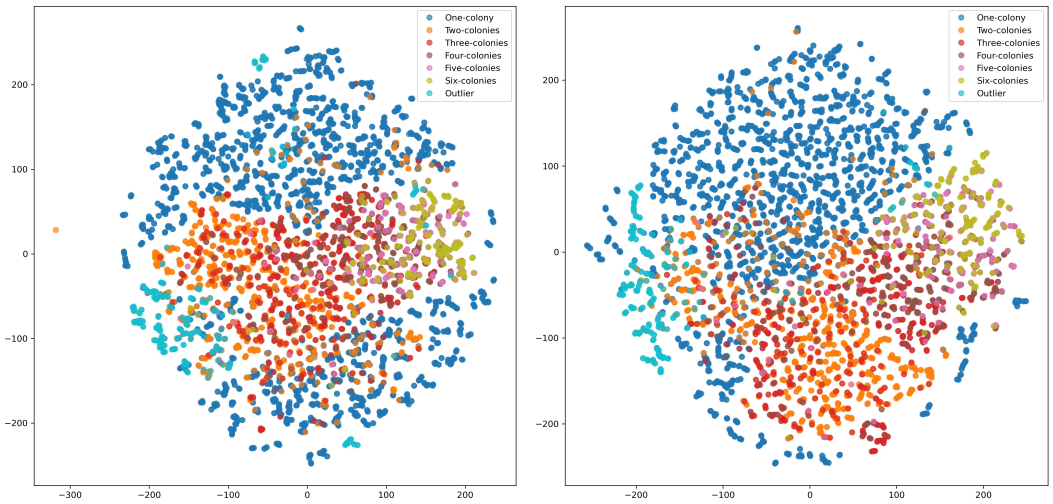
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.38 Visualisation of the last two network layer outputs from the balanced MicrobiaNet model evaluated on MicrobiaS1B1 training set with dimensionality reduced by t-SNE of 100 perplexity.



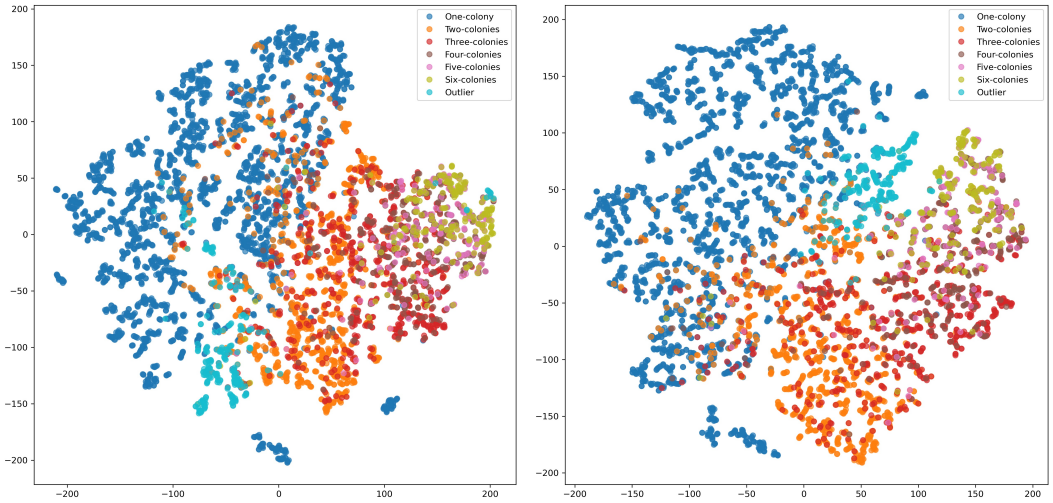
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.39 Visualisation of the last two network layer outputs from the balanced MicrobiaNet model evaluated on MicrobiaS1(B1) validation set with dimensionality reduced by PCA.



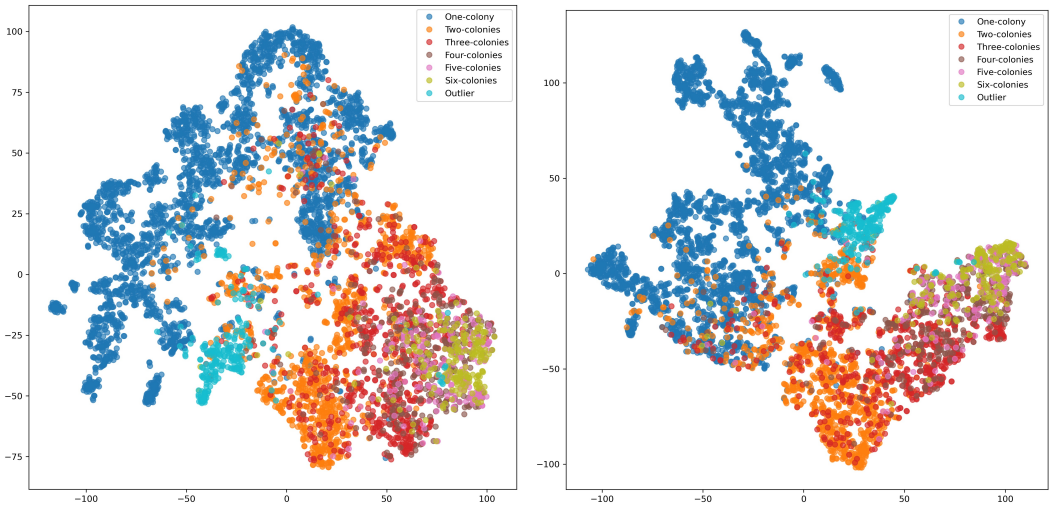
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.40 Visualisation of the last two network layer outputs from the balanced MicrobiaNet model evaluated on MicrobiaS1(B1) validation set with dimensionality reduced by t-SNE of 2 perplexity.



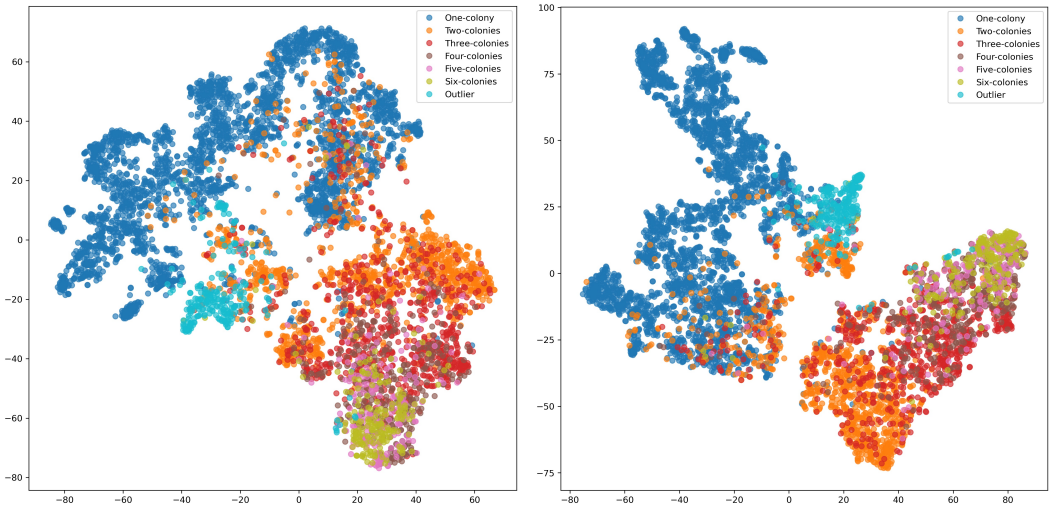
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.41 Visualisation of the last two network layer outputs from the balanced MicrobiaNet model evaluated on MicrobiaS1(B1) validation set with dimensionality reduced by t-SNE of 5 perplexity.



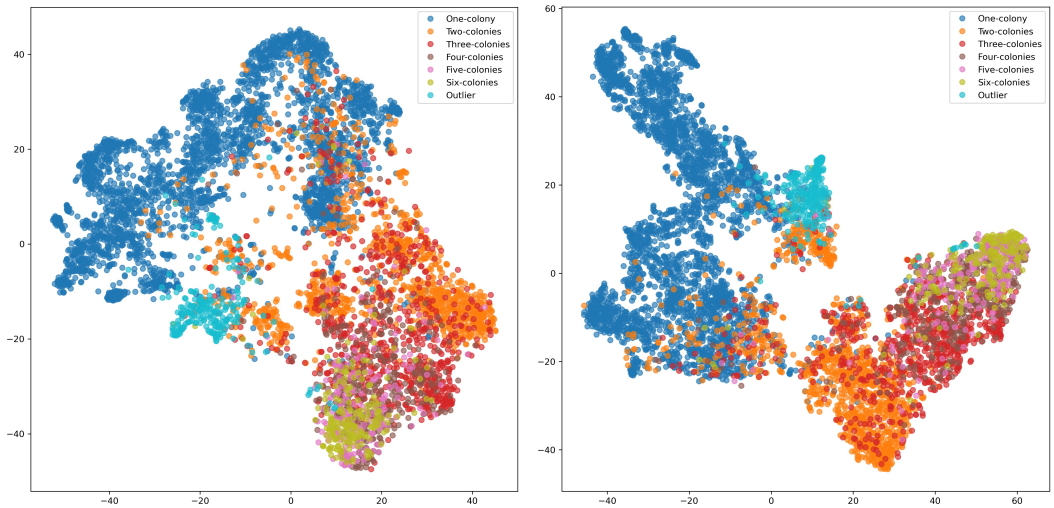
(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.42 Visualisation of the last two network layer outputs from the balanced MicrobiaNet model evaluated on MicrobiaS1(B1) validation set with dimensionality reduced by t-SNE of 30 perplexity.



(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.43 Visualisation of the last two network layer outputs from the balanced MicrobiaNet model evaluated on MicrobiaS1(B1) validation set with dimensionality reduced by t-SNE of 50 perplexity.



(a) The second to last network layer output. (b) The last network layer output.

Fig. 4.44 Visualisation of the last two network layer outputs from the balanced MicrobiaNet model evaluated on MicrobiaS1(B1) validation set with dimensionality reduced by t-SNE of 100 perplexity.

4.3.2 Analysis of the high image similarity by class concatenation

This experiment aims to analyse the high image similarity by class concatenation. The least four performing classes are concatenated into a single class because they are entangled and overlapped in the feature space. This means Three-colonies, Four-colonies, Five-colonies, and Six-colonies classes are lumped into a single class named More-colonies.

The class concatenation is performed on the MicrobiaS1 training and validation sets which are still imbalanced. The new data set that consists of four classes is referred to as MicrobiaS1C1 data set where C is short for the concatenation. MicrobiaNet shown in Fig. 4.1 is also used in this experiment with the final layer modified to produce 4 scores rather than 7 scores. The training and evaluation follow the same procedures explained in § 4.1.2. Additionally, the trained model is interpreted by visualising network layer outputs in the same way as explained in § 4.3.1.

Results

The overall evaluation results on MicrobiaS1C1 data set are listed in Table 4.7. It shows that the model can achieve 0.93 and 0.92 in training F1 score and validation F1 score. The detailed classification results on MicrobiaS1C1 training and validation sets are presented in Table 4.8 and 4.9 which are computed from their corresponding confusion matrix shown in Fig. 4.45 and 4.46. They show the model is not biased towards the majority class because the standard deviation of training F1 scores and validation F1 scores are 0.04 and 0.05. Additionally, the wrong predictions are not dominated by the majority One-colony class as illustrated in Fig. 4.45 and 4.46.

Table 4.7 Overall evaluation results on MicrobiaS1C1 data set.

Data type	Precision	Recall	F1 score	Accuracy (%)
Training	0.93	0.93	0.93	92.54
Validation	0.92	0.92	0.92	91.85

Table 4.8 Classification results evaluated on MicrobiaS1C1 training set.

Class Name	Precision	Recall	F1 score
One-colony	0.95	0.98	0.97
Two-colonies	0.83	0.89	0.86
More-colonies	0.96	0.86	0.91
Outlier	0.92	0.84	0.88

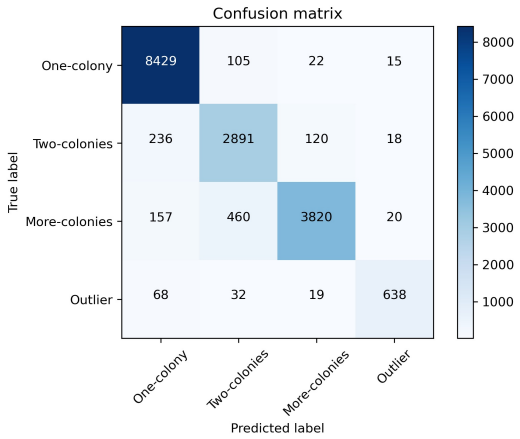


Fig. 4.45 Confusion matrix from MicrobiaS1C1 training results.

Table 4.9 Classification results evaluated on MicrobiaS1C1 validation set.

Class Name	Precision	Recall	F1 score
One-colony	0.94	0.98	0.96
Two-colonies	0.82	0.84	0.83
More-colonies	0.94	0.87	0.90
Outlier	0.91	0.84	0.87

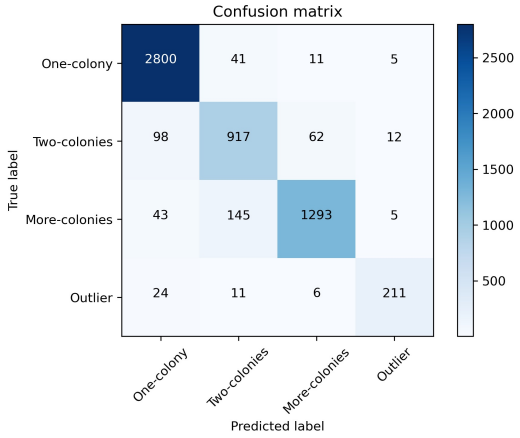


Fig. 4.46 Confusion matrix from MicrobiaS1C1 validation results.

Table 4.10 and Fig. 4.47 show the detailed classification results after converting the baseline validation results (Table 4.3 and Fig. 4.3) to 4 classes. Its overall validation precision, recall, F1 score and accuracy are 0.91, 0.91, 0.91 and 82.79%. The MicrobiaS1C1 validation F1 score is only 0.01 higher than the baseline validation F1 score, suggesting the class concatenation has a trivial contribution to improve colony-cardinality classification. Meanwhile, the network layer outputs from these four classes have a distinct cluster for each

class as illustrated in Appendix B.2. The colony-cardinality classification performance may be improved by carefully designing algorithms to tackle the high image similarity across classes.

Table 4.10 Baseline MicrobiaS1 validation results converted to 4 classes.

Class Name	Precision	Recall	F1 score
One-colony	0.94	0.98	0.96
Two-colonies	0.82	0.81	0.82
More-colonies	0.93	0.86	0.89
Outlier	0.86	0.80	0.83

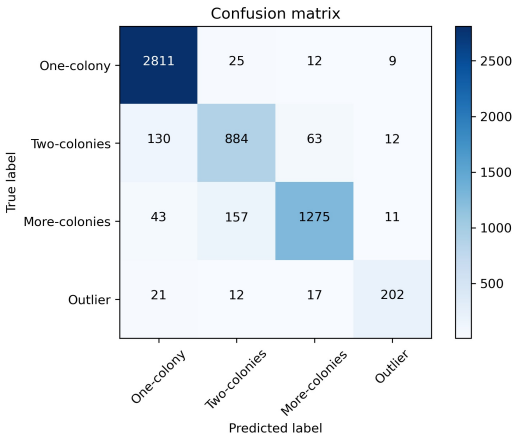


Fig. 4.47 Confusion matrix converted from baseline MicrobiaS1 validation results (Fig. 4.3).

4.3.3 Case study summary

This case study has investigated the research question "What has been the impact of class imbalance and high visual similarity on MicrobiaNet?". It has empirically proved that the class imbalance has a very limited impact on the colony-cardinality classification performance, whereas the high visual similarity across classes is the key issue for improving colony-cardinality classification performance. This is achieved by investigating the neural network trained on a balanced data set by data downsampling, comparing its performance against the baseline performance and interpreting the model in feature space. Despite a dramatic reduction of training data, the balanced model produces an average of 0.76 from 5 different validation F1 scores which is only 0.06 lower than the baseline model. Additionally, an attempt to solve the high visual similarity across classes by class concatenation has made

a marginal improvement of 0.01 in validation F1 score compared with the baseline performance. This suggests that the future work should focus on tackling the high visual similarity across classes in order to further improve colony-cardinality classification performance.

4.4 Re-evaluation of MicrobiaNet with limited data

This section aims to investigate the research question "To what extent does MicrobiaNet perform with limited labelled data?". Specifically, this section re-evaluates MicrobiaNet after identifying that class imbalance has a very limited impact on the baseline performance, and the marginal performance deterioration from a significant reduction of training data via data downsampling. This means MicrobiaNet will be trained on the balanced data set with data downsampling applied, which is relatively limited labelled data. Thereafter, the trained model will be evaluated on the Microbia test set that has never been used in this chapter so far.

4.4.1 Experimental setup

The MicrobiaS1 training data and validation are combined into a single set and with data downsampled to train the best-performing colony-cardinality classification neural network shown in Fig. 4.1. The combined and downsampled training set has totally 5334 images where each class has 762 images. This is significantly reduced from the baseline performance where the training set has totally 17050 images. The MicrobiaS1 test set, which has never been used in this chapter, is used to evaluate the trained model to show how well the model generalises on unseen data. The class distribution of test set remains the same as detailed in Table 4.1. The training hyper-parameters and data normalisation remains the same as explained in § 4.1.2.

4.4.2 Results

The final evaluation results of MicrobiaNet are listed in Table 4.11. It shows that the model is able to generalise on unseen data with a training F1 score of 0.8 and a test F1 score of 0.78 in spite of the significant reduction of training data during the data downsampling process. The detailed training and test results are shown in Table 4.12 and 4.13. These two tables are calculated from their corresponding confusion matrix shown in Fig. 4.48 and 4.49. Most of the wrong predictions are from the actual class's neighbouring classes. For example, the majority of misclassified Five-colonies are from Four-colonies as shown in Fig. 4.48 and 4.49; the majority of misclassified Four-colonies are from Three-colonies as shown in Fig. 4.48

and 4.49. This implies the high image similarity across classes is the main limitation of using colony-cardinality classification to count colonies in a real-life application.

Table 4.11 Final evaluation results of MicrobiaNet.

Data type	Precision	Recall	F1 score	Accuracy (%)
Training	0.81	0.80	0.80	79.98
Test	0.79	0.79	0.78	78.57

Table 4.12 Final training results of MicrobiaNet.

Class Name	Precision	Recall	F1 score
One-colony	0.76	1.00	0.86
Two-colonies	0.84	0.90	0.87
Three-colonies	0.84	0.61	0.71
Four-colonies	0.64	0.81	0.71
Five-colonies	0.79	0.55	0.65
Six-colonies	0.83	0.82	0.83
Outlier	0.97	0.92	0.95

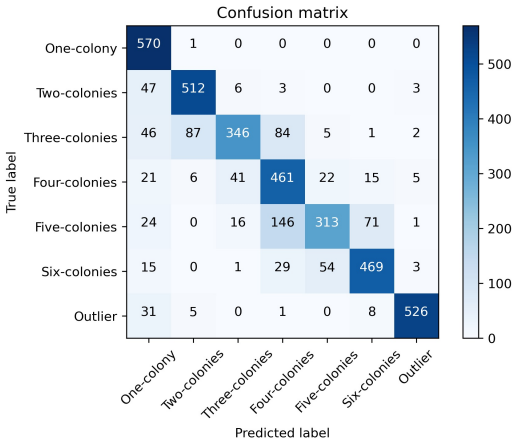


Fig. 4.48 Final confusion matrix from training results of MicrobiaNet.

4.4.3 Case study summary

This experiment has investigated the research question "To what extent does MicrobiaNet perform with limited labelled data?". It has empirically proved that MicrobiaNet is able to produce a F1 score of 0.78 from test set, even if the model is trained on only 5334 images which are dramatically reduced from 17050 images in the baseline performance. The term

Table 4.13 Final test results of MicrobiaNet.

Class Name	Precision	Recall	F1 score
One-colony	0.91	0.98	0.94
Two-colonies	0.81	0.72	0.76
Three-colonies	0.68	0.43	0.52
Four-colonies	0.35	0.53	0.42
Five-colonies	0.33	0.31	0.32
Six-colonies	0.57	0.54	0.55
Outlier	0.81	0.90	0.85

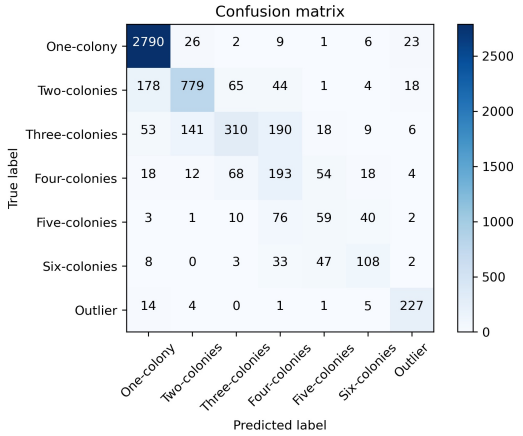


Fig. 4.49 Final confusion matrix from test results of MicrobiaNet.

limited labelled data is considered as relatively limited labelled data because it is well-known that neural networks require a large amount of training data to generalise well on unseen data.

4.5 Conclusions

This chapter has investigated the cardinality classification method to count small and clustered objects with an application to bacterial colonies through four case studies. This investigation has bridged the research gap identified in § 2.5.2. The main contribution to knowledge from this chapter is the analysis of class imbalance and high visual similarity across classes, as well as the finding that image similarity rather than the class imbalance is the key issue for using cardinality classification to count small and clustered objects. The detailed research outcomes from this chapter are explained as following:

1. A baseline performance of MicrobiaNet, which is the best-performing colony-cardinality classification neural network to the best of my knowledge, is formalised.

2. MicrobiaNet is interpreted, and it is concluded that the model suffers from the high visual similarity across classes rather than imbalanced class distribution.

3. It is empirically proven that the existing machine learning interpretation methods, such as feature and class activation map visualisation, struggle to extract useful information to interpret MicrobiaNet.

4. A simple class concatenation method is used to tackle the high visual similarity across classes, and it is concluded that this problem requires a careful algorithm design.

5. MicrobiaNet is re-evaluated after identifying the limited impact of class imbalance on the performance, and it is identified that visual similarity is the key issue of using cardinality classification to count small and clustered objects as a real-life bacterial colony counting application.

In addition to these research outcomes, the industrial partner of this project, Synoptics Ltd, has been advised that MicrobiaNet is capable of counting small and clustered colonies even if the model is trained on 5334 images. However, the counting performance is capped due to the high visual similarity across classes. Synoptics Ltd has also been advised that this method is not able to learn from limited labelled data to count small and clustered object, as well as being generalisable to a different domain/category. This is not only because of the high visual similarity, but also the requirement of retraining the model on data from a different domain/category.

Finally, it can be concluded that cardinality classification counting method is unsuitable for the more challenging industry provided data set, i.e. Synoptics Dataset V2 introduced in § 3.5.2, due to two reasons. One is that the cardinality classification counting algorithm is counting from segments rather than plate images. As a result, it cannot be directly suitable for the Synoptics plate images. The other is that the high visual similarity across classes has yet to be solved by MicrobiaNet, which is the best-performing cardinality classification algorithm for colony counting to the best of my knowledge. This limitation suggests that it could be an unsuccessful investigation, let alone the fact that it is extremely time-consuming and expensive to crop and label segments from each individual plate image in Synoptics data set. Therefore, a different counting method that is based on density map estimation will be introduced and used to tackle the more challenging Synoptics data set in the second part (§ II) of this thesis.

Part II

Aspects of density estimation

Chapter 5

Proposed algorithms

Chapter 4 concludes that MicrobiaNet is not able to address small object size, clustered object, expensive cost to collect and annotate data, and domain/category adaptations collectively. This is not only because of the high visual similarity across classes, but also the requirement of retraining the model on data from a different domain/category.

In order to bridge the research gap identified in § 2.5.1, this chapter introduces a deep convolutional neural network called Aligned Custom Few-shot Adaptation and Matching Network (ACFamNet). It is a particular adaptation of Few-shot Adaptation and Matching Network (FamNet) [112] to count small and clustered objects. The synergy of an end-to-end trainable model, aligned region of interest pooling and optimised feature extraction method empowers ACFamNet to effectively count small and clustered objects.

This chapter also presents an advanced ACFamNet called ACFamNet Pro. It is inspired by Similarity-Aware Feature Enhancement block for object Counting (SAFECount) which is published by You et al. [156] during the research of ACFamNet. ACFamNet Pro is designed with multi-head attention mechanism and residual connections which improve ACFamNet to be readily generalisable to colonies of a different category. The proposal of ACFamNet and ACFamNet Pro commences in chronological order.

5.1 ACFamNet

5.1.1 Overview

The core concept of ACFamNet is illustrated in Fig. 5.1. It is an end-to-end trainable model with two modules: feature correlation module and regression module. In the former module, the support feature which is derived from exemplars performs a feature correlation operation

on the query feature which is derived from an input image. The output of feature correlation is a similarity map that is input to the regression module to produce a density map.

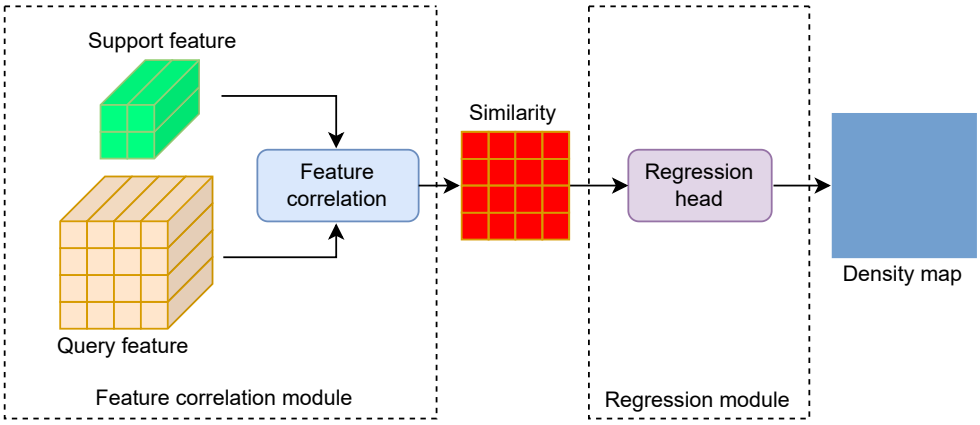


Fig. 5.1 Core concept of ACFamNet.

The design of ACFamNet is inspired by FamNet which can effectively count 147 types of generic objects if three exemplars of the target object are provided. This is because the exemplar can be used as a template to find its occurrence in the input image. FamNet also tackles the difficulty of collecting and annotating data by only using these exemplars. Meanwhile, FamNet counts objects by predicting a density map which is helpful to deal with dense and overlapped objects. However, FamNet has yet to become end-to-end trainable and adapt to small and clustered objects, which motivates the design of ACFamNet.

5.1.2 Feature correlation module

ACFamNet feature correction consists of a simple convolutional layer with k 7×7 kernels to extract features from the input image, where k is a hyper-parameter. This convolution is performed by 2 strides with 0 padding added to preserve the input image's spatial information, followed by batch normalisation and ReLU activation. The output is referred to as query feature. The design of 7×7 kernel size, 2 strides and 0 padding is identical to the first convolutional layer in ResNet-50 [56] which has been widely used in deep learning community nowadays.

The location of exemplars (blue cubes in Fig. 5.2), which is also referred to as RoI, in the input image is proportionally projected to the query feature map (orange cube in Fig. 5.2) by dividing their coordinates by 2 without quantisation. Then, the projected location (location of purple cubes in Fig. 5.2) is used to perform RoI align to produce a support feature map (green cubes in Fig. 5.2). The support feature is used as a kernel to convolve the query feature

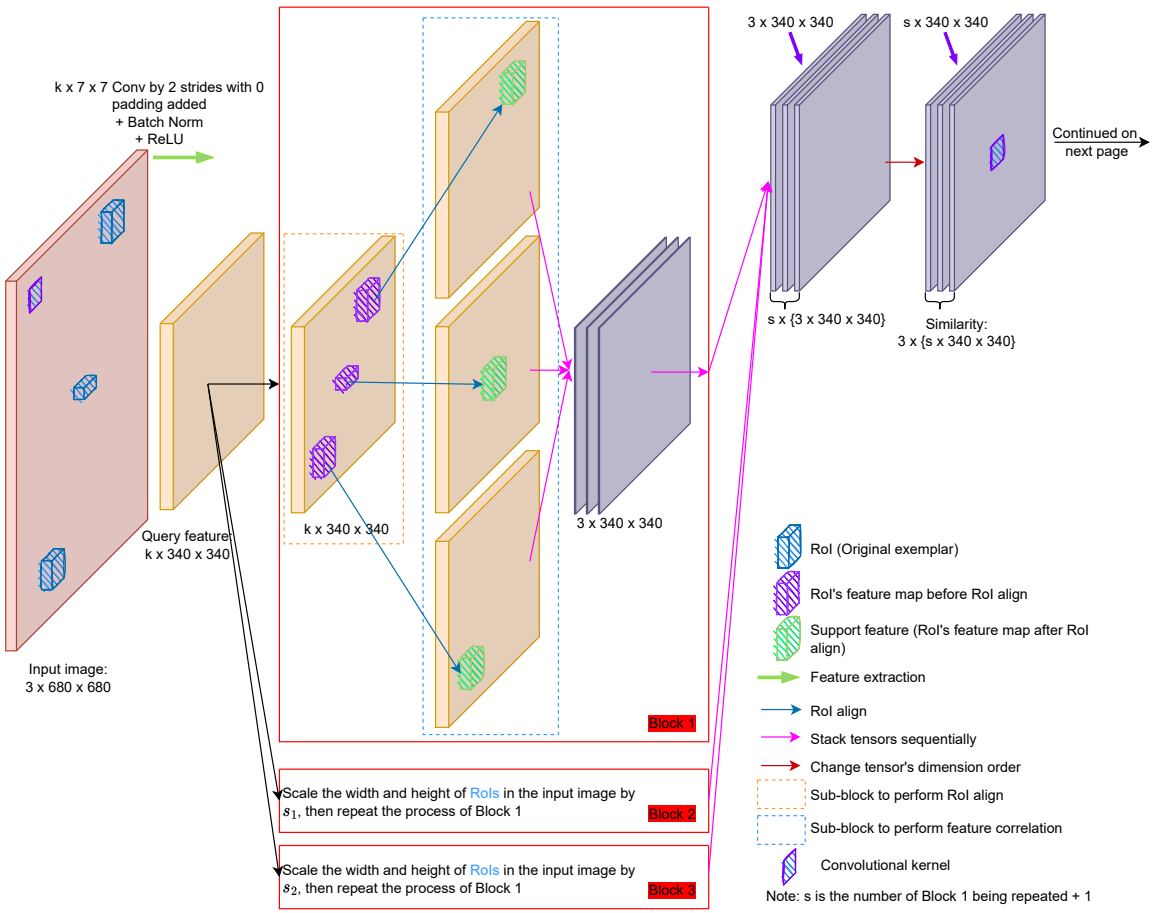


Fig. 5.2 Illustration of ACFamNet feature correlation module.

to produce a similarity map. This process is referred to as feature correlation and is repeated for other exemplars where outputs are stacked together.

The feature correlation process is repeated another two times with the original exemplars scaled by two different factors, aiming to tackle the same exemplar of different sizes. Similarly, the outputs are stacked together before reorganising the dimension order based on the exemplar's dimension. This is because ACFamNet prioritises features from each exemplar over features from resized exemplar. It will be showing that ACFamNet will suffice without repeating feature correction multiple times in the experimental section.

5.1.3 Regression module

ACFamNet regression module receives the similarity map from the previous module as input to predict a density map. The design of this module is similar to FamNet's density prediction module. As illustrated in Fig. 5.3, the regression module consists of five convolutional layers

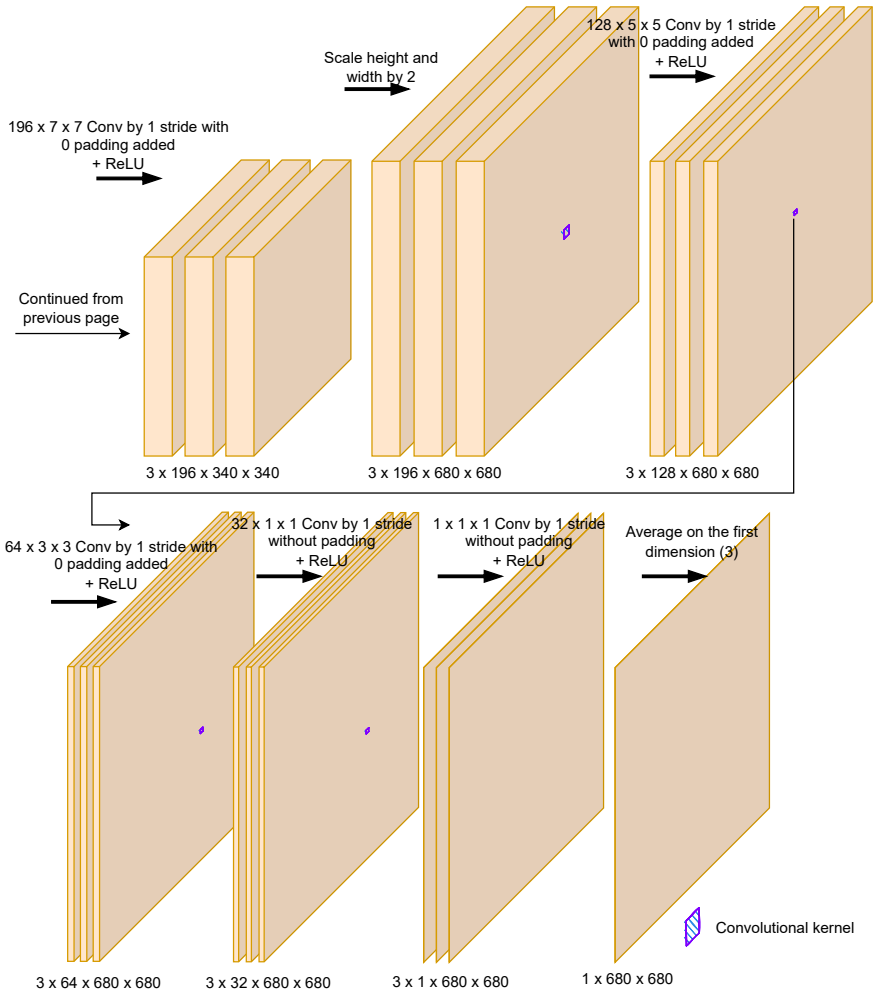


Fig. 5.3 Illustration of ACFamNet regression module.

and an upsampling layers placed after the first convolutional layer. The upsampling layer will double the input's height and width using bilinear interpolation algorithm. The kernel parameters, which are the number of kernels, height and width, for these five convolutional layers are $196 \times 7 \times 7$, $128 \times 5 \times 5$, $64 \times 3 \times 3$, $32 \times 1 \times 1$, and $1 \times 1 \times 1$ respectively. The first, second, and third convolutional layers have zero padding added to preserve the input's height and width. All convolutions are performed by one stride. Additionally, ReLU is used as the activation function to activate the output of each convolutional layer. Finally, the output is averaged on the exemplar dimension to produce a one-dimensional density map that has the same height and width as the original input image.

5.1.4 Comparison with FamNet

The design of ACFamNet is heavily inspired by FamNet which is explained in § 3.4. Compared with FamNet, ACFamNet is end-to-end trainable. This is because an end-to-end trainable model has been proven effective to tackle challenging tasks, such as speech recognition [28, 79], machine translation [150] and autonomous driving [129], since all modules in the model become differentiable and easier to optimise for the entire task [49]. Additionally, ACFamNet chooses RoI Align to tackle RoI misalignment for small objects. This is because the RoI pooling used in FamNet results in severe information loss for small objects. Moreover, ACFamNet has a simplified feature extraction module which is a single convolutional layer. It significantly reduces computational cost without degrading performance. Finally, ACFamNet only requires a single scale factor rather than 3 scale factors in FamNet which also reduce computational cost.

5.2 ACFamNet Pro

This section presents an advanced ACFamNet called ACFamNet Pro. It is designed with additional multi-head attention mechanism and residual connections which improve ACFamNet. The proposal of ACFamNet Pro is inspired by SAFECount which was published during the research of ACFamNet.

5.2.1 Overview

The core concept of ACFamNet Pro is illustrated in Fig. 5.4. It is an end-to-end trainable model with two modules: residual feature enhancement module and regression module. In the former module, the support feature which is derived from exemplars performs a feature correlation operation on the query feature which is derived from an input image. The output of feature correlation is a similarity map which is fused with the support feature and query feature to enhance features. The enhanced features along with the similarity map are input into the regression module to produce a density map.

5.2.2 Query feature and support feature

The query feature and support feature are extracted by a feature extractor illustrated in Fig. 5.5. This feature extractor is often known as *backbone* in literature. Concretely, the feature extractor consists of a simple convolutional layer with k 7×7 kernels, where k is a hyper-parameter. The convolution is performed by 2 strides with 0 padding added to preserve

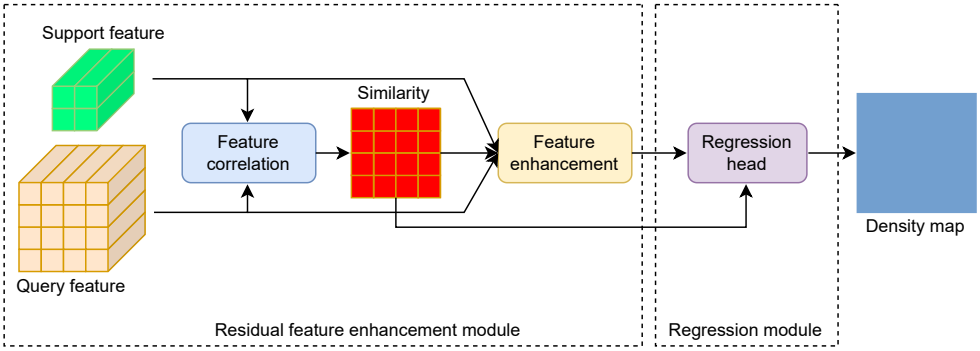


Fig. 5.4 Core concept of ACFamNet Pro.

the input image's spatial information, following by batch normalisation and ReLU activation. This feature extractor is identical to the first convolutional layer of ACFamNet shown in Fig 5.2. The query feature is denoted as $\boldsymbol{f_Q} \in \mathbb{R}^{k \times H_Q \times W_Q}$, where H_Q and W_Q are half of the input image's height and width.

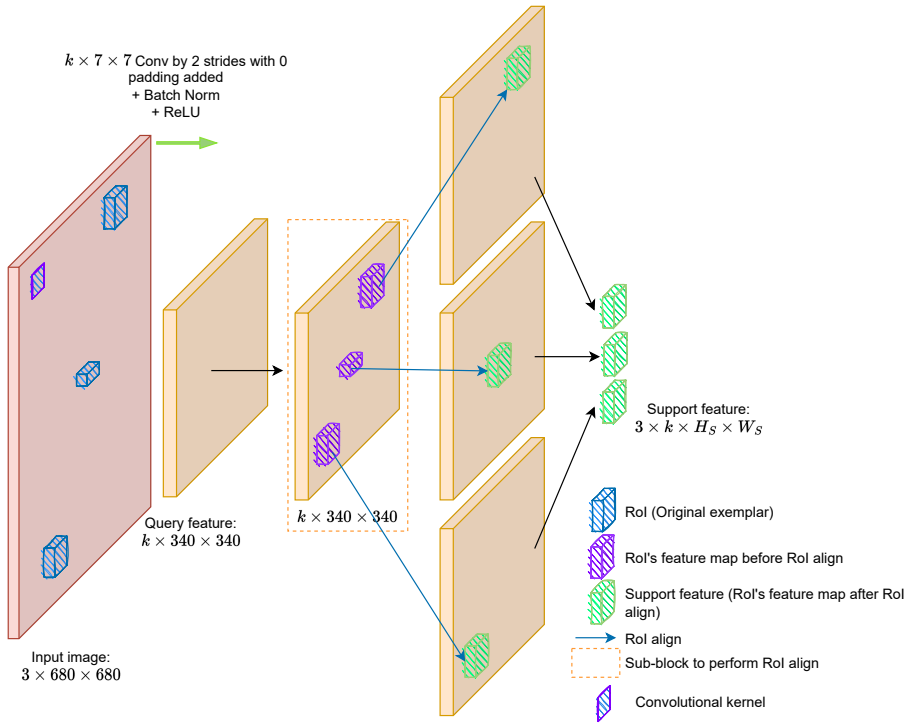


Fig. 5.5 Feature extractor.

The support image is commonly cropped from the query image so that only the specified exemplar is presented in the support image. Therefore, the support feature can be obtained by applying RoI align on the query feature. This process is repeated K times if K support

images are used. As a result, the support feature is denoted as $\boldsymbol{f_S} \in \mathbb{R}^{K \times k \times H_S \times W_S}$, where H_S and W_S are RoI align's height and width.

5.2.3 Residual feature enhancement module

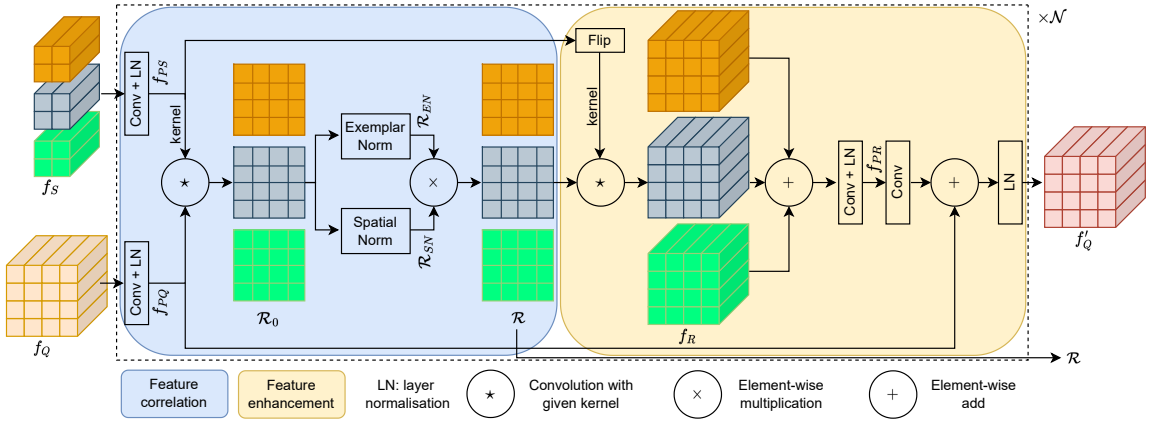


Fig. 5.6 Residual feature enhancement module.

The detailed design of residual feature enhancement module is illustrated in Fig 5.6. It firstly projects the query feature and support feature into the same feature space followed by a comparison at every spatial position to produce a score map. Multiple score maps generated by the support images are concatenated and normalised along the exemplar dimension and the spatial dimensions to produce a reliable similarity map. This is achieved in the feature correlation block. The similarity map is used as weights to integrate the support feature into the query feature to produce an enhanced feature. This is achieved in the feature enhancement block. The enhanced feature along with the similarity map are input to a regression module to produce a density map, where the similarity map is used as a residual connection to improve the regression module. Finally, the whole residual feature enhancement module can be stacked multiple times to further enhance feature representations.

The design of feature enhancement block mimics the attention mechanism used in transformers [141] which describes a weighted average of elements with the weights dynamically calculated based on an input query and elements' keys, where the element is interpreted as the support feature in ACFamNet Pro. Additionally, the residual feature enhancement module mimics the multi-head attention mechanism in transformers which allows the model to control the mixing of information between elements, i.e. support feature, to enrich feature representations. Because of these two mechanisms and residual connection, the model is able to focus more on regions that are similar to the support images in the query image, producing a better counting result.

Feature correlation block

The aim of feature correlation block is to produce a similarity map to robustly highlight regions in the query feature $\boldsymbol{f_Q}$ that are similar to the support feature $\boldsymbol{f_S}$. It has three steps. Learnable feature projection, feature comparison and score normalisation.

Learnable feature projection. The useful features from the support feature $\boldsymbol{f_S}$ and query feature $\boldsymbol{f_Q}$ are dynamically selected by 1×1 convolution whose kernel number is C. Another purpose of this convolution is that both support feature $\boldsymbol{f_S}$ and query feature $\boldsymbol{f_Q}$ are projected into the same feature space so that they can be compared. The convolution is followed by a layer normalisation to bring these two features to the same distribution. The outputs are referred to as *projected support feature* and *projected query feature* with updated notations: $\boldsymbol{f}_{PS} \in \mathbb{R}^{K \times C \times H_S \times W_S}$ and $\boldsymbol{f}_{PQ} \in \mathbb{R}^{C \times H_Q \times W_Q}$. In practice, $\boldsymbol{f_S}$ and $\boldsymbol{f_Q}$ share the same convolution layer and layer normalisation layer because $\boldsymbol{f_S}$ is cropped from $\boldsymbol{f_Q}$ followed by the RoI align operation.

Feature comparison. The projected support feature $\boldsymbol{f}_{PS}$ and projected query feature $\boldsymbol{f}_{PQ}$ are compared in a point-wise fashion by using $\boldsymbol{f}_{PS}$ as a kernel to convolve $\boldsymbol{f}_{PQ}$. This convolution has 0 padding added to preserve the spatial information. The output is a score map $\boldsymbol{R_0} \in \mathbb{R}^{K \times 1 \times H_Q \times W_Q}$:

$$\boldsymbol{R_0} = \text{conv}(\boldsymbol{f}_{PQ}, \text{kernel}), \quad \text{kernel} = \boldsymbol{f}_{PS} \tag{5.1}$$

Score normalisation. The scores in the score map $\boldsymbol{R_0}$ are normalised to prevent extremely large/small values from dominating/un-stabilise the learning process. It is achieved by Exemplar Normalisation (ENorm) and Spatial Normalisation (SNorm) and the element-wise multiplication of their outputs. ENorm normalises $\boldsymbol{R_0}$ along the exemplar dimension which is expressed as $\text{softmax}_{\text{dim}=0}()$ shown in Equation 5.2 to produce $\boldsymbol{R}_{EN}$. Meanwhile, SNorm normalises $\boldsymbol{R_0}$ along the height and width dimension with Equation 5.3 where the $\max_{\text{dim}=(2,3)}()$ finds the maximum value from the corresponding height dimension and width dimension. The spatially normalised score map $\boldsymbol{R}_{SN}$ thus has an important characteristic, the value in the score map from the position that is mostly similar or related to the projected support feature $\boldsymbol{f}_{PS}$ would be close to 1, whereas the other values range from 0 to 1. The last step of score normalisation is the element-wise multiplication of $\boldsymbol{R}_{EN}$ and $\boldsymbol{R}_{SN}$ which is presented in Equation 5.4.

$$\boldsymbol{R}_{EN} = \text{softmax}_{\text{dim}=0}\left(\frac{\boldsymbol{R_0}}{\sqrt{H_S W_S C}}\right) \tag{5.2}$$

$$\boldsymbol{R}_{SN} = \frac{\exp(\boldsymbol{R_0}/\sqrt{H_S W_S C})}{\max_{\text{dim}=(2,3)}(\exp(\boldsymbol{R_0}/\sqrt{H_S W_S C}))} \tag{5.3}$$

$$\boldsymbol{R} = \boldsymbol{R}_{EN} \otimes \boldsymbol{R}_{SN} \tag{5.4}$$

where $\boldsymbol{R_0}, \boldsymbol{R}_{EN}, \boldsymbol{R}_{SN}, \boldsymbol{R} \in \mathbb{R}^{K \times 1 \times H_Q \times W_Q}$.

Feature enhancement block

The aim of feature enhancement block is to exploit the similarity map $\boldsymbol{R}$ as weights to enhance the projected query feature $\boldsymbol{f}_{PQ}$. This is because the similarity map can well represent the relationship between the projected query feature and projected support feature but fails to informatively represent the query image. Feature enhancement block has two steps: weighted feature aggregation and learnable feature fusion.

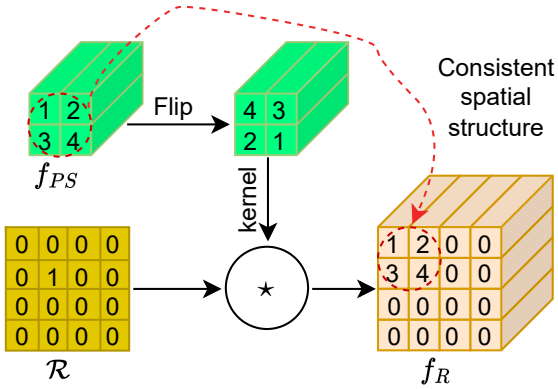


Fig. 5.7 **Illustration of kernel flipping in FEM**. Its purpose is to preserve the spatial structure from the projected support feature $\boldsymbol{f}_{PS}$. In this illustration, $\boldsymbol{R}$, $\boldsymbol{f}_{PS}$, and $\boldsymbol{f}_{\boldsymbol{R}}$ have the K dimension removed for simplicity, meaning only a support image is used in this example. The motivation of this design is that suppose the feature in the projected query feature $\boldsymbol{f}_{PQ}$ corresponding to the position of 1 in $\boldsymbol{R}$ has the maximum similarity with $\boldsymbol{f}_{PS}$ and the other positions in $\boldsymbol{f}_{PQ}$ have no similarity, the similarity-weighted feature $\boldsymbol{f}_R$ should replicate values in $\boldsymbol{f}_{PS}$ to the position in $\boldsymbol{f}_R$ which corresponds to the position of 1 in $\boldsymbol{R}$, whereas other positions in $\boldsymbol{f}_R$ should be zero.

Weighted feature aggregation. Firstly, the projected support feature $\boldsymbol{f}_{PS}$ is flipped horizontally and vertically before using it as a kernel to convolve the similarity map $\boldsymbol{R}$ with 0 padding added. As illustrated in Fig. 5.7, its purpose is to preserve the spatial structure from the projected support feature $\boldsymbol{f}_{PS}$. It is feasible because of the important characteristic gained from the score normalisation step in the feature correlation block.[1] The output is accumulated along the exemplar dimension to produce a similarity-weighted feature $\boldsymbol{f}_{\boldsymbol{R}} \in \mathbb{R}^{C \times H_Q \times W_Q}$ if K support images are used. The weighted feature aggregation is summarised in Equation 5.5.

[1]The important characteristic: the value in the score map from the position that is mostly similar or related to the projected support feature is close to 0, whereas the rest ranges from 0 to 1.

$$\boldsymbol{f_R} = \text{sum}_{\text{dim}=0}(\text{conv}(\boldsymbol{R}, \text{kernel})), \quad \text{kernel} = \text{flip}(\boldsymbol{f_{PS}}) \tag{5.5}$$

Learnable feature fusion. The similarity-weighted feature $\boldsymbol{f_R}$ is projected into the same feature space as the projected query feature $\boldsymbol{f_{PQ}}$ resides with a 1×1 convolution whose kernel number is 1 (since the number of channels of $\boldsymbol{f_R}$ is already C) followed by a layer normalisation to produce the projected similarity-weighted feature $\boldsymbol{f_{PR}} \in \mathbb{R}^{C \times H_Q \times W_Q}$. As shown in Fig. 5.6, $\boldsymbol{f_{PR}}$ is fused into the projected query feature $\boldsymbol{f_{PQ}}$ with an efficient network which contains a convolutional block and a layer normalisation to produce the final enhanced feature $\boldsymbol{f'_Q} \in \mathbb{R}^{C \times H_Q \times W_Q}$. The learnable feature fusion is expressed as:

$$\boldsymbol{f'_Q} = \text{LayerNorm}(\boldsymbol{f_{PQ}} + \text{h}(\text{LayerNorm}(\text{conv}(\boldsymbol{f_R}, \text{kernel})))), \quad \text{kernel} \in \mathbb{R}^{1 \times 1 \times 1} \tag{5.6}$$

where the convolutional block $h(x)$ is implemented as:

$$h(x) = \text{conv}(\text{dropout}(\text{LeakyReLU}(\text{conv}(x, \text{kernel}))), \text{kernel}), \quad \text{kernel} \in \mathbb{R}^{C \times 3 \times 3} \tag{5.7}$$

Multi-block architecture and comparison with attention

As illustrated in Fig. 5.6, the residual feature enhancement module that consists of feature correlation block and feature enhancement block can be stacked N times because the height and width of enhanced feature map remain the same. This multi-block architecture mimics the multi-head attention mechanism in transformers. Additionally, the vanilla transformer attention mechanism, which is shown in Equation 5.8 where Q, K, V, and d_k are query, key, value, and scale factor respectively, can be simplified as $\text{similarity}(Q,K)V$. ACFamNet Pro mimics this attention mechanism by interpreting Q as the query feature, K as the support feature, and V as the query feature. Moreover, feature correlation block and feature enhancement block preserve the spatial information $(C \times H \times W)$, whereas vanilla transformer attention mechanism loses the spatial structure because it flattens feature map $(C \times H \times W)$ to $(C \times HW)$.

$$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{5.8}$$

Multi-scale support features

Similar to ACFamNet, the size of support images can be rescaled by different factors, aiming to address the same object of different sizes. This can be achieved by repeating the feature

extractor shown in Fig 5.5 with scaled support images and concatenating multiple support features along with the first dimension.

5.2.4 Regression module

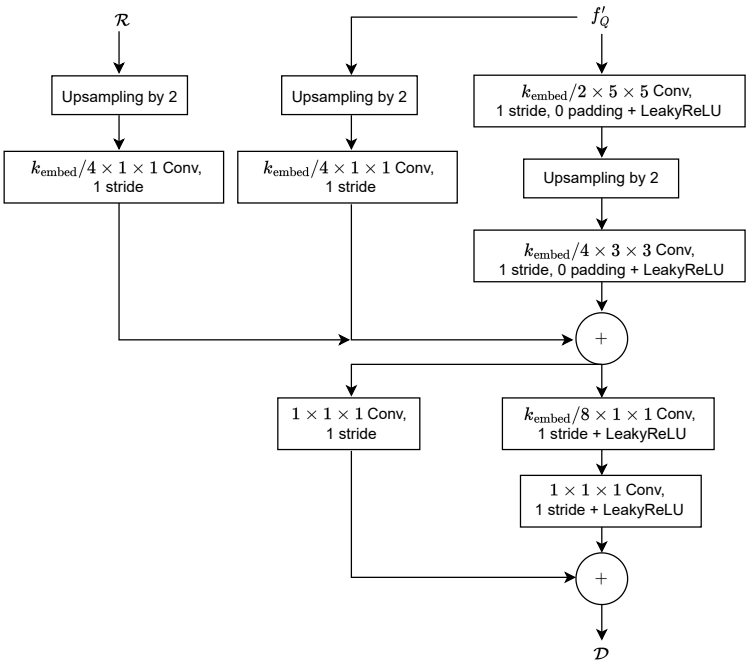


Fig. 5.8 Regression module.

The enhanced feature $\boldsymbol{f'_Q}$ and similarity map $\boldsymbol{R}$ are input to a regression module to predict a density map $D \in \mathbb{R}^{H \times W}$ where the height H and width W are identical to the query image. As illustrated in Fig. 5.8, this regression module consists of four main convolutional layers where the first convolutional layer is followed by a bi-linear upsampling layer to double the height and width of features. These four main convolutional layers are activated by Leaky ReLU activation rather than the traditional ReLU because the former is commonly used for transformer-based models to tackle long sequence data. Additionally, three residual connections are added to the regression module. The enhanced feature $\boldsymbol{f'_Q}$ is upsampled to double the height and width of features followed by a 1×1 convolution before being added to the input to the third convolutional layer. The similarity map $\boldsymbol{R}$ is also upsampled to double the height and width followed by a 1×1 convolution before being added to the input to the third convolutional layer. Finally, the input to the third convolutional layer is added to the output of the final convolutional layer to produce the density map D. The number of

convolutional kernels and the kernel size in each convolutional layer are detailed in Fig. 5.8 where k_{embed} is a hyper-parameter.

5.2.5 Comparison with SAFECount

The design of ACFamNet Pro is inspired by Similarity-Aware Feature Enhancement block for object Counting (SAFECount) proposed by You et al. [156]. The differences between ACFamNet pro and SAFECount can be described from three perspectives: feature extractor, feature enhancement module and the regression module.

Firstly, ACFamNet Pro uses a simple learnable convolutional layer to extract features with RoI align operation, whereas SAFECount uses the first three frozen blocks of ResNet-18 to extract features with RoI pooling operation. Secondly, ACFamNet Pro's feature enhancement module is almost identical to SAFECount except that ACFamNet Pro passes the additional similarity map $\boldsymbol{R}$ as a residual connection to enhance density map estimation. Finally, ACFamNet's regression module is completely redesigned with three residual connections to improve density map estimation.

Chapter 6

Experiments

6.1 Evaluation and training strategies

6.1.1 Evaluation metrics and data

Evaluation metrics

Conventionally, mean absolute error (MAE) and root mean square error (RMSE) are used in many counting methods to evaluate counting results [142, 112]. They are defined as follows. $MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$, $RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$, where N is the number of samples and $\hat{y}_i$, y_i are the predicted count and ground truth count respectively. However, a plate image that has a large colony count can generate a larger error that will ultimately dominate MAE and RMSE. Because of the larger error from some images, MAE and RMSE may not reflect the overall counting performance. To overcome this issue, the absolute error for a prediction is divided by the ground truth count to generate a normalised absolute error. The normalised absolute error for all predictions are averaged to produce the mean normalised absolute error (MNAE), which is independent of colony count. It is defined as $MNAE = \frac{1}{N}\sum_{i=1}^{N}|\frac{\hat{y}_i - y_i}{y_i}|$, where N is the number of samples and $\hat{y}_i$, y_i are the predicted count and ground truth count respectively. MNAE is also known as Mean Absolute Percentage Error (MAPE) in statistics.

Data

Experiments run on Synoptics Dataset V2 introduced in § 3.5.2 which are also illustrated in Appendix A.1 because of the reduced computation on smaller images. The test set, which is 20% of the whole data set discussed in § 3.5.2, is reserved for the comparison of machine learning based counting methods and traditional counting methods. The reservation of the test set also prevents machine learning based counting methods from being overly optimised

on the training set. Instead, the training set, which is 80% of the whole data set, is used to train and evaluate machine learning models by k-fold cross-validation which is detailed in § 6.1.2. Finally, the 20% test set, which will be only used once, is to evaluate the best model identified by the k-fold cross-validation after being trained on the 80% training set from scratch.[1]

The ground truth density function introduced in Equation 3.52 for Synoptics Dataset V2 uses the same parameters introduced by Ranjan et al. [112]. The kernel size k is the average distance between each dot and its nearest neighbour for the whole map. The σ is a quarter of the kernel size. This specific design varies the ground truth function based on each individual image. As a result, the density map is adaptable to each image regardless of its image size and object density.

6.1.2 Training strategy

K-fold cross-validation is used to train and estimate the performance of different neural networks with different hyper-parameters. In order to save computational resources, this chapter focuses on hyper-parameters that determine a network's architecture/structure/design rather than hyper-parameters that determine how a network is trained. Another reason for prioritising such hyper-parameters is that hyper-parameters that determine how a network is trained can reuse some values in some publications, whereas the hyper-parameters that determine a network's architecture/structure/design are more dependent on the network's designer.[2]

Before k-fold cross validation, the data set is equally divided into k folds. Then k iterations of training and validation are performed such that a different fold of data is reserved for validation and the remaining $(k-1)$ folds are used for training in each iteration. This method assures that each example in the data set is used for evaluation so that the performance estimation is less biased [113]. After k iterations of training and validation, the k validation result sets are averaged to generate a final result to estimate the performance of the neural network.

According to Hastie et al. [54], 5 and 10 are common choices for the k value. However, it is set to 5 in this chapter because the training set size of Synoptics Dataset V2 is 100 and 5 is the smallest divisor of 100 except 1. Meanwhile, choosing a larger divisor of 100 as the k value, such as 10, will significantly increase computational cost.

[1] Training the best model from scratch means the learned parameters of the best model are discarded. Instead, they are relearned from new data following the same training strategy.

[2] For example, learning rate, batch size and epoch number are hyper-parameters that determine how the network is trained.

Before training, images in the $(k-1)$ folds are normalised by subtracting the mean of RGB pixel values in the $(k-1)$ folds and dividing by the standard deviation of RGB pixel values in the $(k-1)$ folds to speed up convergence for training. The mean and standard deviation of RGB pixel values in the $(k-1)$ folds are reserved to normalise images in the validation fold during the evaluation process. This data normalisation procedure has been widely used when training neural networks [74, 56].

The loss function is Mean Square Error (MSE) since models used in this chapter are regression models. Meanwhile, MSE loss function is equivalent to the maximum likelihood estimation as discussed in § 3.1.3. Adam [71] is used to train these models with a learning rate of 10^{-5}. The batch size is 1 and the epoch number is 1500. The choice of MSE loss function, Adam optimisation, learning rate, batch size and epoch number is identical to [112]. Additionally, the training process is early stopped if the validation MNAE has not improved 1% for 200 epochs continuously. This early stopping is designed to avoid wasteful training if the performance has not shown any improvement for a long period.

6.2 Experiments on ACFamNet

6.2.1 Training

Experiments in this section follows the same setup explained in § 6.1.2 for data, training and evaluation. Before training, parameters of neural networks reuse some parameters of trained models. Specifically, the first convolutional layer in ACFamNet shown in Fig. 5.2 reuses ResNet-50's first convolutional layer's parameters. This is because ACFamnet can reuse ResNet-50's already learned knowledge to extract features. Similarly, the density map prediction module in ACFamNet is pre-trained on the FSC-147 data set [112] which consists of 6135 images across 147 object categories. This is because the density prediction module can utilise the knowledge of counting 147 types of object, which is helpful to learn to count colonies. When tuning hyper-parameters of ACFamNet, models whose first convolutional layer has more than 64 kernels repetitively duplicate trained model's kernels until the parameter dimension is matched. Likewise, models whose scale factor is 1 only partially transfer trained model's parameters to match the parameter dimension.

6.2.2 Hyper-parameter tuning

Setup

A hyper-parameter tuning is performed on ACFamNet. They are the number of kernels in ACFamNet's first convolutional kernel, the RoI align output size and the number of scales. Among these hyper-parameters, the number of kernels in ACFamNet's first convolutional layer, i.e. the value of k in Fig. 5.2, is fine-tuned with 64, 128, 256 and 512 because they are commonly used in the design of convolutional kernels. The RoI align output size is fine-tuned with 1×1, 3×3, 5×5, and 7×7 since an odd RoI align output size can provide a symmetrical kernel for feature correlation. Additionally, the number of scales is fine-tuned with 1 and 3 where 1 scale factor indicates no scaling,[3] 3 scale factors consists of 1, 0.9 and 1.1.[4] The choice of 3 scale factors is identical to FamNet.

This hyper-parameter tuning aims to answer the first part of the research question "How can the feature engineering in FamNet be modified to learn from limited labelled data to count small and clustered colonies, and be readily generalisable to a different domain or category? And what has been the effect of the modified feature engineering on addressing these problems?". That is "How does ACFamNet learn from limited labelled data to count small and clustered colonies?" since ACFamNet is derived from FamNet with modified feature engineering.

Results

Table 6.1 presents the results of tuning ACFamNet. The mean and standard deviation of each validation MNAE are obtained from 5-fold cross-validation. Comparing the results from 3 scale factors against those from 1 scale factor in Table 6.1, ACFamNet tends to produce a lower validation MNAE when 1 scale factor is used regardless of the RoI align output size and convolutional kernel number. There is a possibility that this tendency is due to the nature of small objects, meaning resizing small objects with different scale factors increases feature space that are not useful and degrades the counting performance.

The increase of RoI align output size, i.e. from 3×3 to 7×7, leads to an increase of validation MNAE regardless of the number of scale factors and convolutional kernels. However, this pattern is not applicable to the 1×1 RoI align output size. The validation MNAE is almost close to 100% when RoI align output size is 1×1 and the kernel number is 64 and 128. Surprisingly, the validation MNAE is no longer close to 100% when the number of kernels is greater than 256 even though the RoI align output size is 1×1. There is a strong

[3] It is equivalent to the removal of Block 2 and 3 in Fig. 5.2.
[4] It means s_1 and s_2 in Fig. 5.2 are 0.9 and 1.1 respectively.

Table 6.1 ACFamNet hyper-parameter tuning results.

	RoI Align	Validation MNAE(%)	
		3 Scale Factors	1 Scale Factor
k = 64	1×1	99.99 ± 0.10[a]	99.98 ± 0.03
	3×3	13.29 ± 2.33	12.06 ± 2.48
	5×5	14.06 ± 1.75	12.86 ± 2.39
	7×7	17.21 ± 1.88	15.63 ± 1.63
k = 128	1×1	99.98 ± 0.02	31.49 ± 34.37
	3×3	12.23 ± 2.15	12.28 ± 1.85
	5×5	14.60 ± 2.45	13.69 ± 2.28
	7×7	17.24 ± 2.40	15.30 ± 1.35
k = 256	1×1	14.27 ± 3.93	13.52 ± 1.30
	3×3	12.94 ± 1.91	$\mathbf{11.85 \pm 2.53}$
	5×5	14.13 ± 2.74	13.91 ± 2.01
	7×7	16.95 ± 2.45	16.18 ± 1.19
k = 512	1×1	13.46 ± 3.16	13.50 ± 1.96
	3×3	13.78 ± 2.12	13.60 ± 2.69
	5×5	15.93 ± 3.08	14.74 ± 2.05
	7×7	17.64 ± 1.57	16.22 ± 3.00

[a] Each validation MNAE presented in this table has a mean MNAE and a standard deviation of MNAE from 5-fold cross-validation. The lower mean MNAE, the better.

possibility that 1×1 RoI align output size requires more kernels because more kernels can boost ACFamNet's ability to tackle more difficult features which compensates the extremely small 1×1 RoI align output size.

The increase of kernel number in ACFamNet's first convolutional layer results in a significant computational cost increase but only a fluctuation of validation MNAE regardless of RoI align output size and number of scale factors. These results would seem to suggest that the performance gain is trivial even if it is at the cost of increasing the number of kernels.

The best result from hyper-parameter tuning is obtained from 256 kernels, 3×3 RoI align output size and 1 scale factor, producing a mean validation MNAE of 11.85% with 2.53% in standard deviation. This means ACFamNet, which is modified from FamNet, is able to count small and clustered colonies because 11.85% is a reasonably small value. Considering only three labelled exemplars is used for training ACFamNet, it indicates that ACFamNet is also

Table 6.2 Detailed 5-fold cross-validation results of ACFamNet with the best hyper-parameters (k=256, 3×3 RoI align and 1 scale factor).

	Metric	Fold					Mean	Std
		1	2	3	4	5		
Training	MAE	16.80	14.89	12.68	14.19	15.80	14.87	1.40
	RMSE	27.79	26.58	23.83	25.68	27.14	26.20	1.38
	MNAE (%)	20.93	15.42	16.24	14.21	15.05	16.37	2.37
Validation	MAE	15.42	13.06	19.24	5.45	7.64	12.16	5.04
	RMSE	37.98	19.04	32.09	7.42	11.29	21.56	11.76
	MNAE (%)	13.80	14.03	13.91	8.45	9.08	**11.85**	**2.53**

able to learn from limited labelled data to count small and clustered colonies. The detailed 5-fold cross-validation results are reported in Table 6.2. The fine-tuned ACFamNet is able to generalise on unseen data since most of validation metrics are lower than training metrics. Particularly, the training MNAE is higher than validation MNAE. A possible explanation for this might be that a smaller data set has smaller intrinsic variance, meaning ACFamNet captures the complexity of data and the inner variance of training set is greater than that of validation set.

The prediction results on two example validation images are illustrated in Fig. 6.1 and 6.2. These three yellow bounding boxes indicate three exemplars input to the model, and the number near each bounding box indicates the predicted number of objects in the corresponding exemplar region.[5] Fig. 6.1 and 6.2 both show that ACFamNet is able to count the small and clustered colonies.

6.2.3 Ablation studies

Setup

Two ablation studies are conducted with Synoptics Dataset V2 to analyse the effectiveness of different components of ACFamNet and the impact of the number of exemplars on the counting performance. These two studies are designed to investigate the latter part of the research question "How can the feature engineering in FamNet be modified to learn from limited labelled data to count small and clustered colonies, and be readily generalisable to a different domain or category? And what has been the effect of the modified feature

[5]These three exemplars highlighted by the yellow bounding box are equivalent to the three blue cubes illustrated in Fig. 5.2.

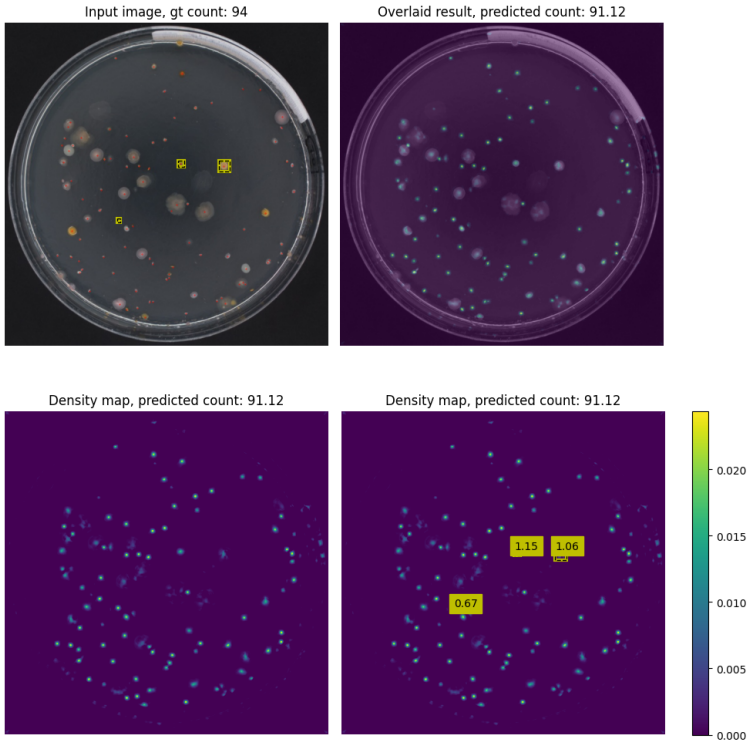


Fig. 6.1 ACFamNet's prediction on an unseen image from validation set.

engineering on addressing these problems?". That is "What has been the effect of the modified feature engineering of ACFamNet on address these problems?".

ACFamNet used in the first ablation study is the one with optimised hyper-parameters, meaning the first convolutional kernel number is 256. The feature engineering of ACFamNet involves single scale factor component and RoI Align component. When ACFamNet has the RoI align component, its RoI align output size is 3×3. In contrast, the RoI align component is replaced with RoI pooling if ACFamNet does not have the RoI align component. Similarly, when ACFamNet does not have the single scale factor component, the single single factor component is replaced with 3 scale factors which are 1, 0.9 and 1.1.[6] In the second ablation study, ACFamNet is also the one with optimised hyper-parameters, meaning the first convolutional kernel number is 256, the RoI align output size is 3×3, and the scale factor is 1. The evaluation method, data and training method used in these studies are identical to those introduced in § 6.1.1 and § 6.1.2.

[6]ACFamNet does not have the single scale factor component is equivalent to the architecture illustrated in Fig. 5.2. In contrast, ACFamNet has the single scale factor component is equivalent to the removal of Block 2 and 3 in Fig. 5.2.

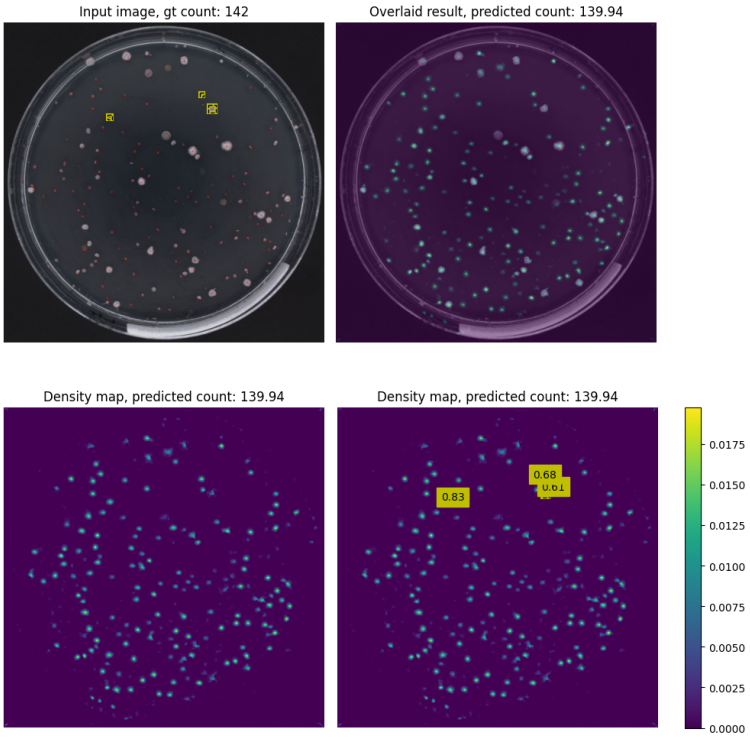


Fig. 6.2 ACFamNet's prediction on another unseen image from validation set.

Results

Table 6.3 Analysis of the effectiveness of different components of ACFamNet.

Components	Combinations			
Single scale factor	×	×	✓	✓
RoI Align	×	✓	×	✓
Validation MNAE(%)	17.73 ± 3.37	12.94 ± 1.91	20.39 ± 4.43	11.85 ± 2.53

The results of analysing the effectiveness of different components of ACFamNet are listed in Table 6.3. These results confirms the importance of individual component of ACFamNet, i.e. the single scale factor and RoI align. It is also discovered that the combination of single scale factor and RoI align can further reduce the validation MNAE from 17.73% to 11.85%. Table 6.4 shows that ACFamNet's counting performance improves with the increase of exemplars. ACFamNet can even produce a reasonable counting result when only one exemplar is provided. These two patterns are similar to those discovered by FamNet's authors, suggesting ACFamNet is a successful adaptation of FamNet to count small and clustered colonies.

Table 6.4 Performance of ACFamNet that is trained with different number of exemplars.

Number of exemplars	Validation MNAE (%)
1	14.94 ± 2.32
2	13.07 ± 2.35
3	11.85 ± 2.53

6.2.4 Comparison with FamNet

Setup

It is necessary to compare ACFamNet against FamNet since the former is inspired by the latter. This experiment aims to investigate the research question "To what extent does FamNet address small bacterial colonies?". The vanilla FamNet has 3 scale factors, RoI pooling and a non-trainable and complex feature extraction module which are explained with more details in § 3.4. FamNet is examined with 3 scale factors and 1 scale factor. Additionally, FamNet is fine-tuned with different RoI align operations. The evaluation method, data and training method used in this section are identical to those introduced in § 6.1.1 and § 6.1.2.

Results

Table 6.5 Results of tuning scale factor for FamNet.

Model	Validation MNAE (%)	
	3 Scale Factors	1 Scale Factor
FamNet	$\mathbf{22.33 \pm 6.53}$	23.83 ± 6.66

The results shown in Table 6.5 indicate that the vanilla FamNet is able to count small bacterial colonies with a mean MNAE of 22.33% and a standard deviation of 6.53% from 5-fold cross-validation. These results also suggest that it is slightly detrimental to reduce FamNet's scale factors from 3 to 1. It is possible that theses results are due to the non-trainable feature extraction module, meaning FamNet needs more features to compensate the non-trainable feature extraction module. It is worth mentioning that the two standard deviation values presented in Table 6.5 are higher than those presented in Table 6.1, suggesting FamNet is less stable than ACFamNet. The detailed 5-fold cross-validation results of vanilla FamNet which has 3 scale factors and RoI pooling are listed in Table 6.6. These results reveal that FamNet's ability to generalise on unseen data is less stable because the validation MNAE has a higher standard deviation than training MNAE across 5-fold data sets.

Table 6.6 Detailed 5-fold cross-validation results of vanilla FamNet (3 scale factors and RoI pooling).

	Metric	Fold					Mean	Std
		1	2	3	4	5		
Training	MAE	16.75	18.39	13.60	27.82	22.10	19.73	4.89
	RMSE	20.93	27.42	18.14	38.30	32.11	27.38	7.33
	MNAE (%)	24.86	22.46	19.02	27.60	22.31	23.25	2.86
Validation	MAE	28.99	16.41	36.04	12.47	15.82	21.94	9.01
	RMSE	49.46	25.73	54.00	17.64	23.07	33.98	14.79
	MNAE (%)	24.68	17.59	32.90	13.77	22.72	**22.33**	**6.53**

Table 6.7 Results of tuning RoI align output size for FamNet.

RoI Align Output Size	Validation MNAE (%)	
	3 Scale Factors	1 Scale Factor
1×1	100.0 ± 0.00	100.0 ± 0.00
3×3	$\mathbf{25.29 \pm 5.60}$	46.14 ± 27.31
5×5	84.38 ± 30.38	46.76 ± 26.90
7×7	99.92 ± 0.03	68.14 ± 25.83

Table 6.8 Comparison between ACFamNet and vanilla FamNet.

Model	Validation MNAE (%)
ACFamNet	$\mathbf{11.85 \pm 2.53}$
FamNet	22.33 ± 6.53

RoI align operations have a limited positive impact on FamNet's counting performance as shown in Table 6.7. The change of RoI align output size has yet to improve FamNet's counting performance since all results shown in Table 6.7 are worse than the vanilla FamNet which has RoI pooling. The observed negative impact of RoI align operations on FamNet's counting performance may be due to the non-learnable feature extraction module, meaning FamNet needs to learn from data to handle interpolated features from RoI align operations. Finally, the fine-tuned ACFamNet outperforms the vanilla FamNet by 10.48% in validation MNAE as presented in Table 6.8.

6.2.5 Comparison with traditional methods

Setup

In order to compare machine learning based counting methods against traditional counting methods, OpenCFU [45] and AutoCellSeg [69] are chosen to compare against ACFamNet. OpenCFU and AutoCellSeg are popular open-source solutions that are based on traditional image thresholding algorithms which require users to predefine some parameters for object detection. These three methods will be evaluated on the same Synoptics Dataset V2 test set that has never been used before to have a fair comparison. In other words, ACFamNet with optimised hyper-parameters, i.e. 256 kernels in the first convolutional layer, 3×3 RoI align and 1 scale factor, will be trained on the training set of Synoptics Dataset V2 and evaluated on the test set of Synoptics Dataset V2 in a hold-out evaluation fashion. The training of ACFamNet has the same learning rate, batch size, epoch number, early stopping strategy, loss function, and Adam optimisation as introduced in §6.1.2. The comparison of running time for these methods is not included in this thesis because deep learning based methods outperform traditional methods due to shared computation for batched input.

Results

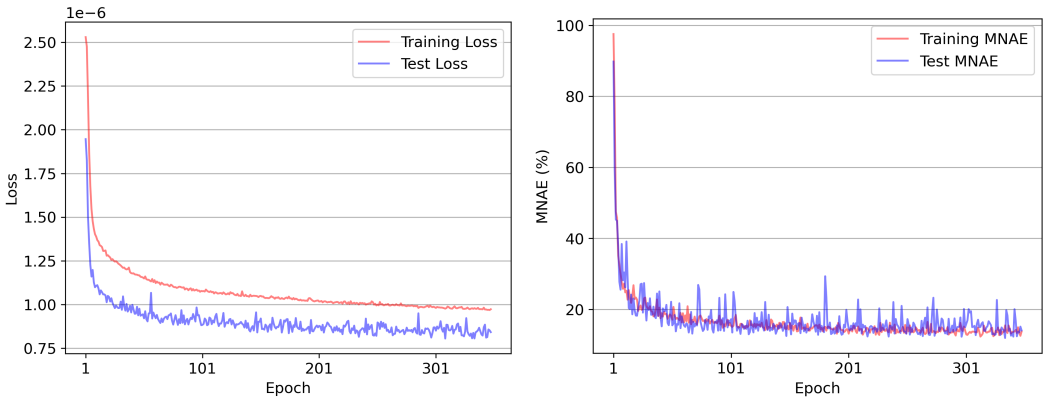
Table 6.9 Comparison between ACFamNet and traditional counting methods.

Metric	OpenCFU	AutoCellSeg	ACFamNet
MAE	41.12	60.92	11.54
RMSE	47.76	69.87	15.56
MNAE (%)	46.57	68.73	**12.52**

Table 6.10 Detailed hold-out evaluation results of ACFamNet.

Metric	Training set	Test set
MAE	16.50	11.54
RMSE	27.54	15.56
MNAE (%)	16.64	**12.52**

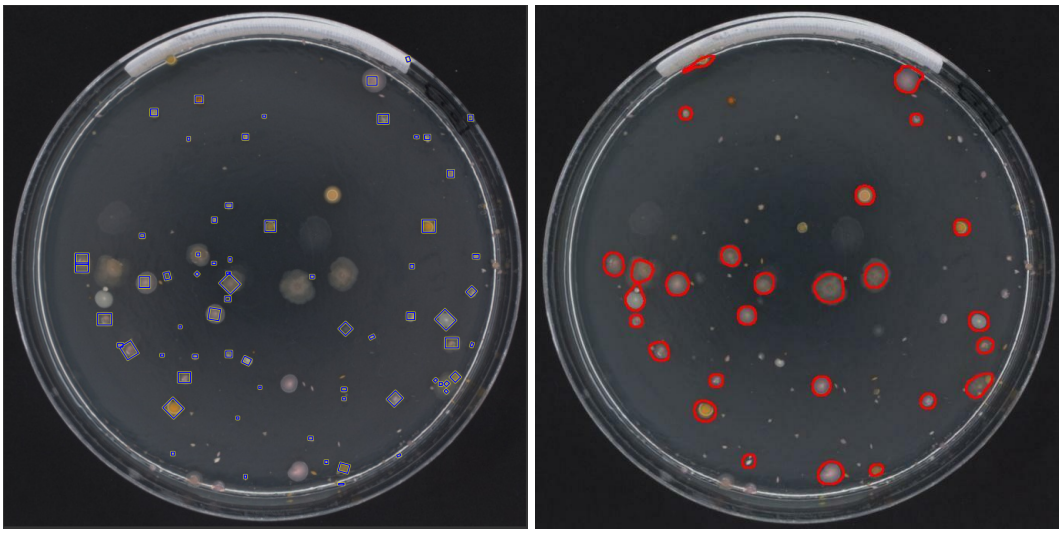
The results of OpenCFU, AutoCellseg and ACFamNet evaluated on the same Synoptics Dataset V2 test set are listed in Table 6.9. These results suggest that ACFamNet outperforms traditional counting methods by a large margin, producing a MNAE of 12.52%. The detailed hold-out evaluation results of ACFamNet presented in Table 6.10 reveals that ACFamNet



(a) Loss curves. (b) MNAE curves.

Fig. 6.3 Loss and MNAE values throughout the training process of ACFamNet.

is able to generalise on unseen data. Similar to the previous 5-fold cross-validation results shown in Table 6.2, the performance on unseen data is better than that on training data. This might be due to the same reason that ACFamNet captures the complexity of data and the inner variance of training data is greater than that of test data, where the detailed variance values are shown in Table 3.3. The finding that ACFamNet is able to generalise on unseen data can also be discovered from the loss and MNAE values throughout the training process plotted in Fig. 6.3a and 6.3b.



(a) 66 colonies detected by OpenCFU. (b) 27 colonies detected by AutoCellSeg.

Fig. 6.4 Illustration of counting result from traditional methods on an image with 83 colonies.

An analysis of prediction results from these three methods is conducted. It is observed from Fig 6.4a and 6.4b that traditional counting methods fail to detect colonies in different

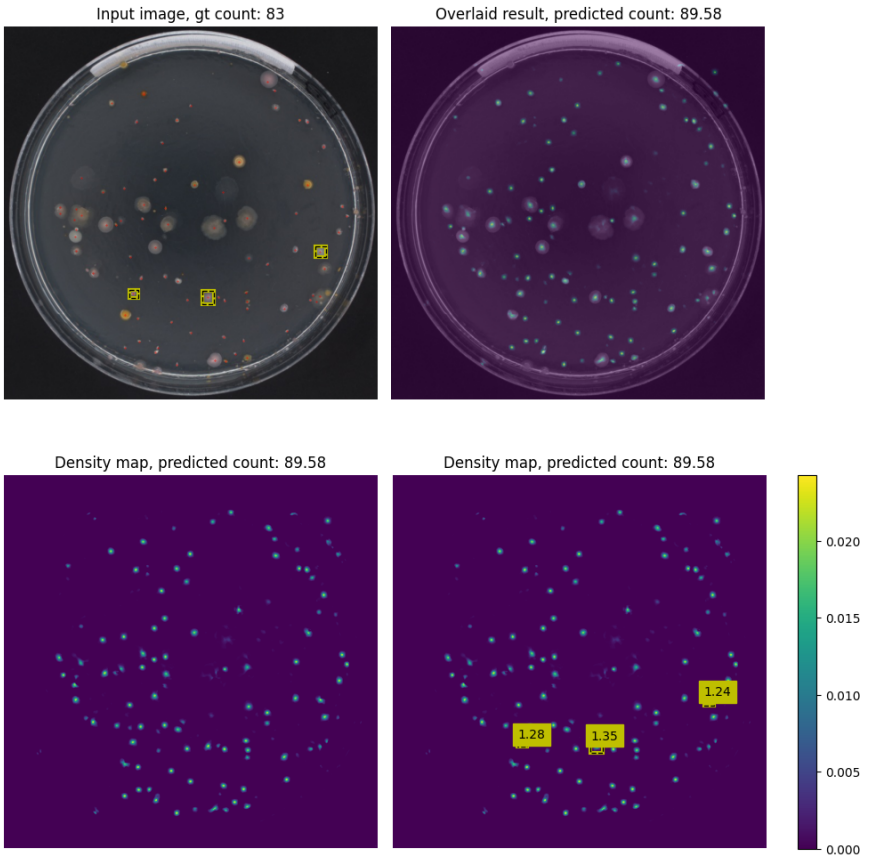


Fig. 6.5 Illustration of ACFamNet's prediction on an image with 83 colonies.

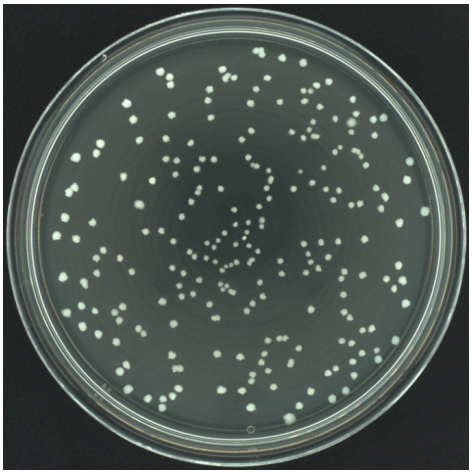
colour, size, shape, and density. A possible explanation might be due to the nature of traditional image processing algorithms, such as thresholding, that are heavily based on human's intervention to adjust parameters to handle the high variety of colonies. Contrary to traditional counting methods, ACFamNet that is based on machine learning is able to learn from data to tackle the high variety of colonies as illustrated in Fig 6.5. Moreover, the neural network of ACFamNet shares computation when tackling a large volume of data, producing a high scalability to ACFamNet in a real laboratory.

6.2.6 Domain or category adaptation

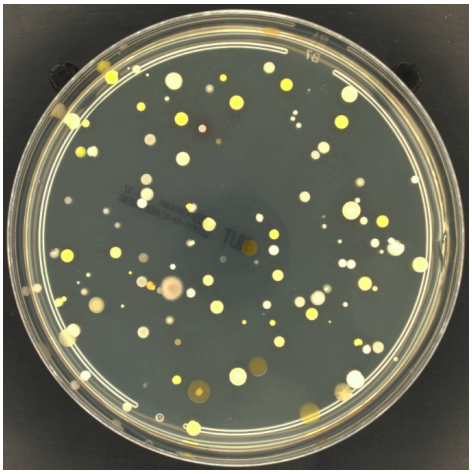
Setup

This experiment aims to evaluate how well ACFamNet count small and clustered colonies of a different category. Because FamNet, which is based on few-shot learning, is capable of counting objects of a different domain/category as long as three exemplars are provided, it is naturally hypothesised that ACFamNet, which is modified from FamNet, is also capable

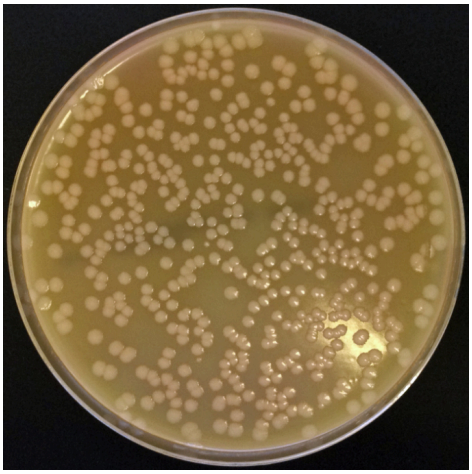
of counting objects of a different domain/category. However, the assumption in few-shot learning methods that object classes in training set do not overlap with object classes in test set is breached in Synoptics Dataset V2 due to the sparsity of colony species. Therefore, this thesis only investigates the cross-category adaptation for ACFamNet. In other words, this experiment investigates the "be readily able to generalise to a domain or category" part of the research question "How can the feature engineering in FamNet be modified to learn from limited labelled data to count small and clustered colonies, and be readily generalisable to a different domain or category? And what has been the effect of the modified feature engineering on addressing these problems?".



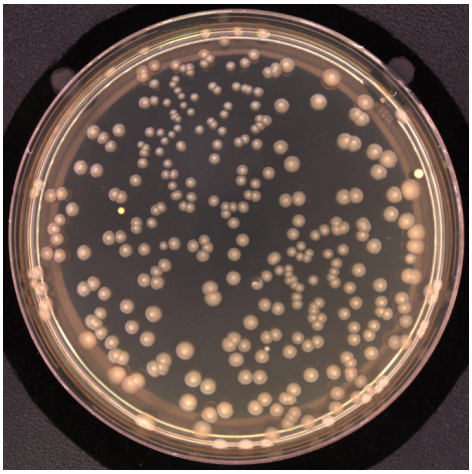
(a) Plate image one with 228 colonies.



(b) Plate image two with 124 colonies.



(c) Plate image three with 529 colonies.



(d) Plate image four with 302 colonies.

Fig. 6.6 Four plate images with colonies that are completely different to these in Synoptics Dataset V2.

Four plate images that contain completely different colony species presented in the whole Synoptics Dataset V2 are used in this experiment, even though the exact colony species in these four plate images remain unknown. These four limited plate images are also provided by Synoptics Ltd who is not able to provide more images due to the difficulties of collecting more plate images. Despite that, the visual difference between these four plate images shown in Fig.6.6 and Synoptics Dataset V2 plate images shown in Appendix § A.1 is significant in colour, background, density and species.

The ACFamNet used in this experiment is the one obtained from § 6.2.5, i.e. the ACFamNet retrained on the whole Synoptics Dataset V2 training set with optimised hyper-parameters. This is to avoid repetitive model training and to exploit the whole Synoptics Dataset training set. In other words, ACFamNet obtained from § 6.2.5 is evaluated on four plate images that contain completely different colonies species shown in Fig. 6.6.

Results

Table 6.11 Results of ACFamNet's cross-category prediction.

Plate image	Ground truth	ACFamNet
Fig. 6.6a	228	306.31
Fig. 6.6b	124	475.85
Fig. 6.6c	529	3.1
Fig. 6.6d	302	832.12
MAE	RMSE	MNAE (%)
371.55	414.58	148.26

Table 6.11 lists detailed results of ACFamNet's cross-category prediction. The MNAE from four plage images is 148.26%. It indicates that ACFamNet is not readily able to generalise on colonies of a different category. A possible reason is that ACFamNet is trained on the data set in which each plate image may contain colonies of the same species, which breaches the most important assumption of few-shot learning. It is also likely caused by the limited images in the data set. Moreover, the significant change of colour in the plate area may attribute to the inaccuracy. For example, plate image shown in Fig. 6.9 has a dramatically different colour in the plate area compared with Synoptics Dataset V2 images shown in Appendix A.1. In spite of inaccurate predictions on colonies of a different category, detailed predictions illustrated in Fig. 6.7, 6.8, 6.9 and 6.10 imply that ACFamNet is still able to predict the location of different colonies based on these three exemplars. This is

because the location of predicted dots in the predicted density map has a tendency to match the location of colonies in the input image.

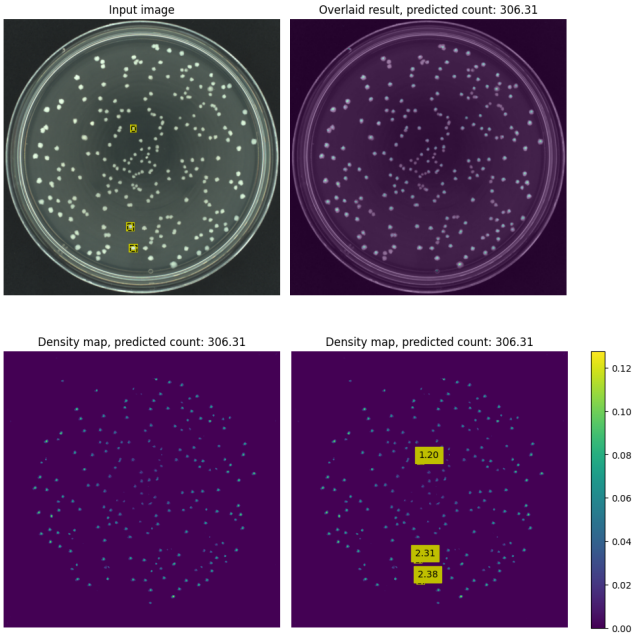


Fig. 6.7 Illustration of ACFamNet's prediction on Fig. 6.6a. Predicted count and ground truth count are 306.31 and 228 respectively.

6.2.7 Summary

This section has introduced ACFamNet, which is a special adaptation of FamNet, to count small and clustered colonies. The fine-turned ACFamNet produces an average of 11.85% in validation MNAE via 5-fold cross-validation with a standard deviation of 2.53% on Synoptics Dataset V2 training set. Ablation studies reveal that single scale factor and RoI align are ACFamNet's important components. Without them, ACFamNet is degraded to 17.73% $\pm$ 3.37% in validation MNAE during the same 5-fold cross-validation setup. Meanwhile, FamNet only produces 22.33% $\pm$ 6.53% in validation MNAE via the same 5-fold cross-validation setup, showing that ACFamNet can outperform FamNet by a large margin. FamNet is also evaluated with different scale factors and RoI align operations with a finding that these two components are detrimental. There is a possibility that it might be due to FamNet's overly complicated and non-learnable feature extraction module. Moreover, ACFamNet is evaluated on Synoptics Dataset V2 test set after it is trained from scratch on Synoptics Dataset V2 training set in a hold-out evaluation fashion. The MNAE calculated from the test set is 12.52% which is a significant improvement from traditional counting methods, such

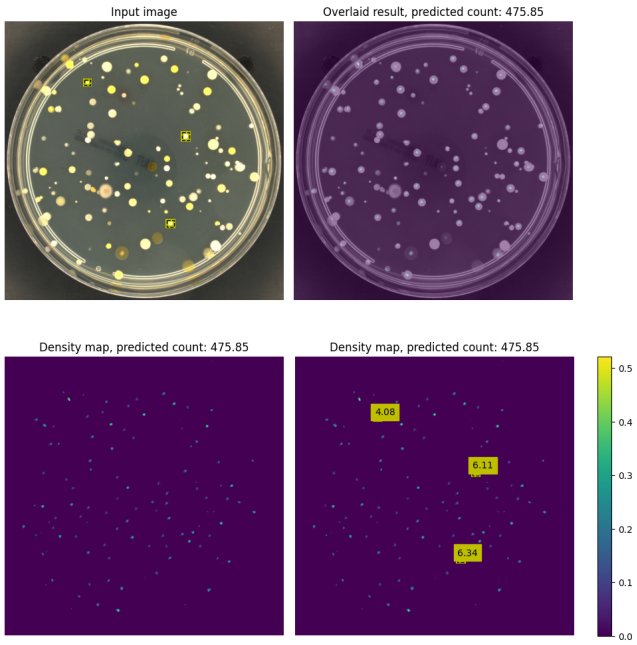


Fig. 6.8 Illustration of ACFamNet's prediction on Fig. 6.6b. Predicted count and ground truth count are 475.85 and 124 respectively.

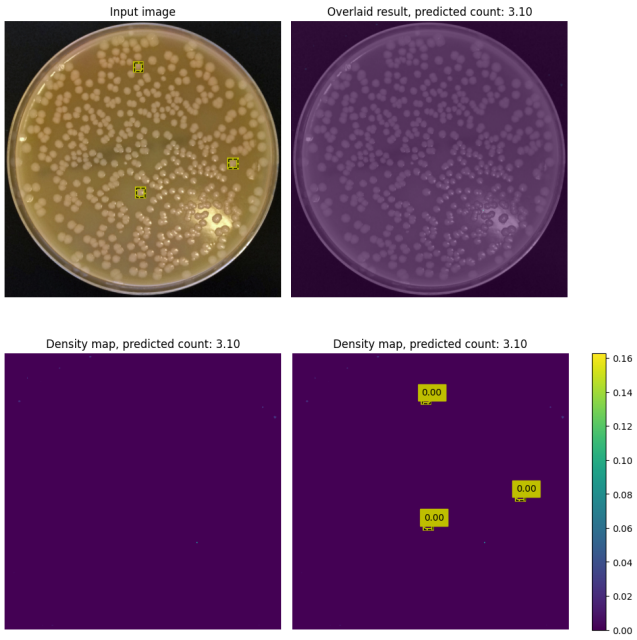


Fig. 6.9 Illustration of ACFamNet's prediction on Fig. 6.6c. Predicted count and ground truth count are 3.1 and 529 respectively.

as OpenCFU and AutoCellSeg, which generate 46.57% and 68.73% in MNAE respectively. Finally, ACFamNet is evaluated on four plate images which contain colonies of completely

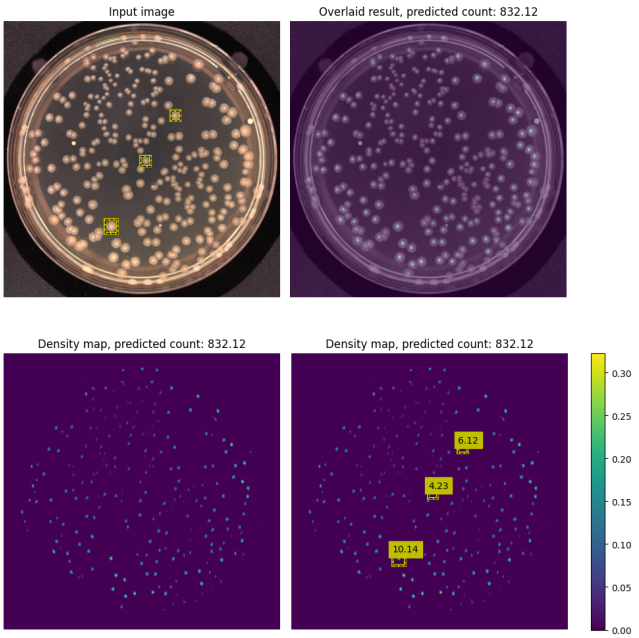


Fig. 6.10 Illustration of ACFamNet's prediction on Fig. 6.6d. Predicted count and ground truth count are 832.12 and 302 respectively.

different species. The excessive 148.26% MNAE may be attributed from the data set in which ACFamNet is trained. These include the overlap of colony categories between training images, the limited number of training images and the significant change of colour of plate areas between training images and these four images.

This section also has investigated two research questions listed in § 1. It has been empirically proved that FamNet can address small bacterial colonies producing 22.33% $\pm$ 6.53% in validation MNAE via a 5-fold cross-validation setup. Additionally, ACFamNet, which is modified from FamNet, is capable of learning from limited labelled data to count small and clustered colonies, producing a result that is better than FamNet and traditional counting methods. The performance gain from ACFamNet is achieved by tackling region of interest misalignment and improving feature extraction during the feature engineering process. However, ACFamNet is not able to readily generalise to a different category. This could be caused by the overlap of colony categories between training images, the limited number of training images and the dramatic change of colour of plate areas between training images and evaluation images. Despite the inability of cross-category generalisation, the single scale and RoI align are identified as two important components of ACFamNet to count small and clustered colonies from limited data.

6.3 Experiments on ACFamNet Pro

6.3.1 Training

Similar to the training of ACFamNet, ACFamNet Pro is trained on Synoptics Dataset V2 training set via the same 5-fold cross-validation. Before training, images in other 4 folds are also normalised. The weights of feature extractor in ACFamNet Pro illustrated in Fig. 5.5 are initialised with a zero-mean Gaussian distribution of 0.01 in standard deviation. This initialisation strategy is the same to SAFECount. The evaluation metrics and data are identical to those introduced in § 6.1.1. Likewise, the loss function is still MSE. Adam is used to train ACFamNet Pro with a learning rate of 10^{-5}. The batch size is 1 and the epoch number is 1500. The training process is early stopped if validation MNAE has not improved 1% for 200 epochs continuously. The choice of MSE loss function, Adam optimisation, learning rate, batch size, epoch number, and early stopping is identical to those used for ACFamNet.

6.3.2 Hyper-parameter tuning

Setup

A hyper-parameter tuning is performed on ACFamNet Pro. This experiment aims to investigate the first part of the research question "How can the feature engineering of SAFECount be transferred to FamNet or the modified FamNet to better adapt it to address small and clustered bacterial colonies, limited labelled data and cross domain/category generalisation since SAFECount is newer and superior to FamNet? And what has been the effect of the modified feature engineering on addressing these problems?". That is "How can the feature engineering of SAFECount be transferred to ACFamNet Pro (the modified FamNet) to better adapt it to address small and clustered bacterial colonies, limited labelled data and cross domain/category generalisation?", since ACFamNet Pro consists of ACFamNet (the modified FamNet) and multi-head attention mechanism from SAFECount.

These hyper-parameters are related to backbone (feature extractor), RoI operation and scale factors. Concretely, the backbone with 128 kernels, which is the same number of kernels used in ACFamNet, is evaluated with/without frozen weights. The RoI operation whose output size is 3×3, which is the optimal size identified in ACFamNet, is fine-tuned with RoI align and RoI pooling. Additionally, the number of scales is fine-tuned with 1 and 3 where 1 scale factor indicates no scaling, 3 scale factors consists of 1, 0.9 and 1.1. The choice of 3 scale factors is identical to ACFamNet.

The dimension of the projected features are 256. The number of residual feature enhancement module is 4. The embed dimension k_{embed} in regression module is 1024. Because these hyper-parameters are optimal in SAFECount, they are not tuned in ACFamNet Pro.

Results

Table 6.12 ACFamNet Pro hyper-parameter tuning results.

	RoI operation (3×3)	Validation MNAE(%)	
		3 Scale Factors	1 Scale Factor
Learnable backbone	RoI pool	10.76 ± 3.35	11.23 ± 2.94
	RoI align	$\mathbf{9.62 \pm 3.35}$	10.27 ± 3.72
Frozen backbone	RoI pool	12.18 ± 3.12	11.61 ± 3.55
	RoI align	10.85 ± 1.86	11.52 ± 2.82

Table 6.12 presents the results of tuning ACFamNet Pro. Comparing the results from learnable backbone against those from frozen backbone, ACFamNet Pro performs better when the backbone is learnable regardless of the choice of RoI operation or scale factors. This can be explained by the same reason why end-to-end trainable ACFamNet outperforms FamNet: all modules in the model become differentiable and easier to optimise for the entire task.

ACFamNet Pro tends to perform better when RoI align operation is used regardless of the choice of backbone or scale factors. This pattern is similar to the one identified in ACFamNet because the model no longer suffers from RoI misalignment. Contrary to ACFamNet, ACFamNet Pro has a tendency to perform better when 3 scale factors are used. A possible explanation is that the multi-head attention mechanism in ACFamNet Pro requires a large feature space which is provided by additional scale factors.

The best hyper-parameter tuning result is achieved by using learnable backbone, RoI align operation and 3 scale factors, producing a mean validation MNAE of 9.62% with 3.35% in standard deviation. Comparing against ACFamNet's best performance in Table 6.1, ACFamNet Pro outperforms ACFamNet by 2.23% in the mean validation MNAE. This reveals that ACFamNet Pro, which consists of modified FamNet, multi-head mechanism from SAFECount and redesigned regression module, can better count small and clustered

Table 6.13 Detailed 5-fold cross-validation results of ACFamNet Pro with the best hyper-parameters (learnable backbone, 3×3 RoI align and 3 scale factors).

	Metric	Fold					Mean	Std
		1	2	3	4	5		
Training	MAE	5.95	10.47	11.25	7.75	7.88	8.66	1.94
	RMSE	11.09	17.23	14.06	17.49	22.48	16.47	3.81
	MNAE (%)	8.11	15.97	17.15	8.68	8.10	11.60	4.07
Validation	MAE	8.21	9.54	18.21	5.06	5.96	9.40	4.68
	RMSE	15.46	12.65	28.72	6.81	9.15	14.56	7.67
	MNAE (%)	7.41	11.49	15.36	7.23	6.61	**9.62**	**3.35**

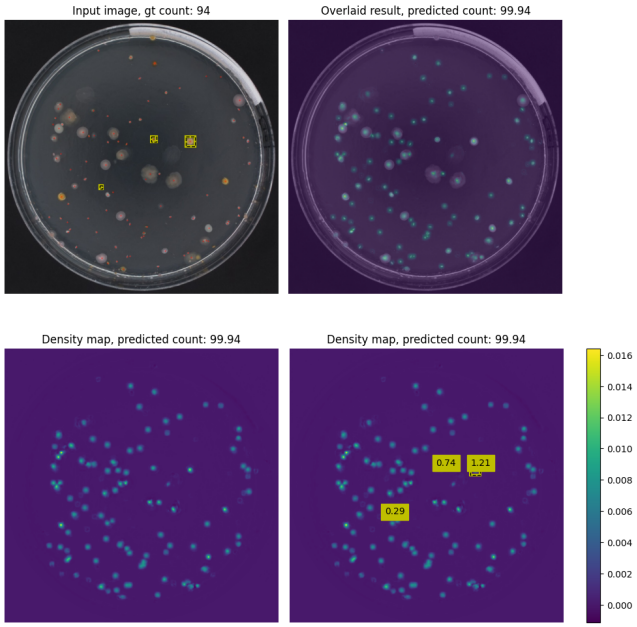


Fig. 6.11 ACFamNet Pro's prediction on an unseen image from validation set.

colonies. The fine-tuned ACFamNet Pro is able to generalise on unseen data as the detailed 5-fold cross-validation results shown in Table 6.13. Similar to ACFamNet's performance, some validation results are better than training results, which might be due to the small data set size.

The prediction results on two example validation images by the fine-tuned ACFamNet Pro are illustrated in Fig. 6.11 and 6.12. They both show that ACFamNet Pro is capable of counting small and clustered colonies since the predicted count is very close to the actual count and the predicted dot in the density map can match its location in the input image.

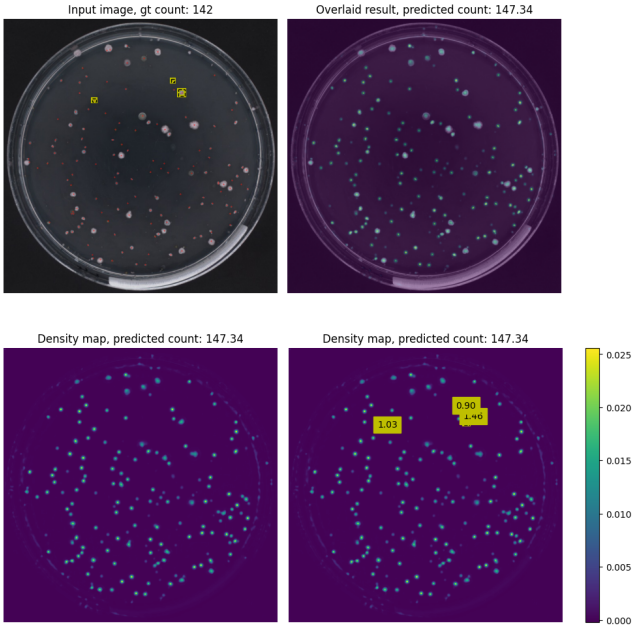


Fig. 6.12 ACFamNet Pro's prediction on another unseen image from validation set.

6.3.3 Ablation studies

Setup

An ablation study is conducted with Synoptics Dataset V2 to analyse the effectiveness of different components of the fine-tuned ACFamNet Pro. This study is designed to investigate the latter part of the research question "How can the feature engineering of SAFECount be transferred to FamNet or the modified FamNet to better adapt it to address small and clustered bacterial colonies, limited labelled data and cross domain/category generalisation since SAFECount is newer and superior to FamNet? And what has been the effect of the modified feature engineering on addressing these problems?". That is "What has been the effect of the modified feature engineering of ACFamNet Pro on address these problems".

ACFamNet Pro used in this study has a learnable backbone, 3×3 RoI align and 3 scale factors. These components during feature engineering include RoI align operation, residual similarity map and learnable backbone. When ACFamNet Pro has the RoI align component, its RoI align output size is 3×3. In contrast, RoI align is replaced with 3×3 RoI pooling if ACFamNet Pro does not have the RoI align component. Similarly, when ACFamNet Pro does not have residual similarity map, it means the similarity map $\boldsymbol{R}$ (the leftmost column) shown in Fig. 5.8 is removed. Likewise, if ACFamNet Pro does not have learnable backbone component, it means the backbone is frozen during training and evaluation phases. The

training method and evaluation method used in this study are identical to those used in § 6.3.2.

Results

Table 6.14 Analysis of the effectiveness of different components of ACFamNet Pro.

Components	Combinations							
RoI align	×	✓	×	×	✓	✓	×	✓
Residual similarity	×	×	✓	×	✓	×	✓	✓
Learnable backbone	×	×	×	✓	×	✓	✓	✓
Valid MNAE (%)	11.91 ±1.80	11.16 ±2.09	12.18 ±3.12	12.14 ±2.39	10.85 ±1.86	10.62 ±2.73	10.76 ±3.35	**9.62 ±3.35**

The results of analysing the effectiveness of different components of ACFamNet Pro during feature engineering are presented in Table 6.14. These results confirms the prominent importance of RoI align component. This is because the validation MNAE is reduced whenever the RoI align component is included. For example, the combination of RoI align and residual similarity, as well as the combination of RoI align and learnable backbone perform better than those without the RoI align component. In other words, the performance is deteriorated when residual similarity and learnable backbone are included individually. This may because ACFamNet Pro is required to learn from data using learnable backbone in order to take advantage of residual similarity map. The synergy of RoI align, residual similarity map and learnable backbone improves ACFamNet Pro from 11.91% to 9.62% in validation MNAE.

6.3.4 Comparison with SAFECount

Setup

It is necessary to compare ACFamNet Pro against SAFECount since the former is inspired by the latter. This experiment aims to investigate the research question "To what extent does SAFECount address small bacterial colonies?". The vanilla SAFECount uses the frozen top three blocks of ResNet18 (frozen backbone) to extract features. It also uses 3 scale factors and RoI pooling. In this study, the first frozen two blocks of ResNet18 are used to extract features. This is because images in Synoptics Dataset V2 are too small to go deeper in ResNet18. Likewise, three upsampling layers in vanilla SAFECount are reduced to two upsampling layers to accommodate smaller images in Synoptics Dataset V2. Additionally, SAFECount

is fine-tuned with RoI align since it is proven effective in ACFamNet and ACFamNet Pro. The evaluation method, data and training method used in this section are the same to those introduced in § 6.3.1.

Results

Table 6.15 Results of tuning RoI operation for SAFECount.

Model	Validation MNAE (%)	
	3×3 RoI pooling	3×3 RoI align
SAFECount	9.86 ± 1.61	$\mathbf{9.79 \pm 2.11}$

Table 6.16 Comparison between ACFamNet Pro and SAFECount.

Model	Validation MNAE (%)
ACFamNet Pro	$\mathbf{9.62 \pm 3.35}$
Vanilla SAFECount	9.86 ± 1.61
Fine-tuned SAFECount	9.79 ± 2.11

The results in Table 6.15 suggest the vanilla SAFECount is able to count small colonies with a mean validation MNAE of 9.86% with a standard deviation of 1.61% across 5-fold cross-validation. Meanwhile, this table reveals that RoI align is better than RoI pooling in SAFECount. This pattern has been consistent in ACFamNet and ACFamNet Pro. The fine-tuned SAFECount with RoI align produces a mean validation MNAE of 9.79% with a standard deviation of 2.11% across 5-fold cross-validation. The comparison between ACFamNet Pro and SAFECount in Table 6.16 shows that ACFamNet Pro can outperform vanilla SAFECount and fined-tuned SAFECount by 0.24% and 0.17% in validation MNAE respectively.

6.3.5 Comparison with other counting methods

Setup

ACFamNet Pro is also compared against traditional counting methods introduced in § 6.2.5. Similarly, ACFamNet Pro with optimal hyper-parameters is trained on the training set of Synoptics Dataset V2 and evaluated on the test set of Synoptics Dataset V2 in a hold-out evaluation fashion. These hyper-parameters include 3×3 RoI align, learnable backbone and 3 scale factors. The training of ACFamNet Pro has the same learning rate, batch size,

epoch number, early stopping strategy, loss function, and Adam optimisation as introduced in § 6.3.2. Additionally, the same hold-out training strategy is applied to train the vanilla SAFECount on Synoptics Dataset V2 so that ACFamNet Pro can be compared against SAFECount.

Results

Table 6.17 Detailed hold-out evaluation results of ACFamNet Pro.

Metric	Training set	Test set
MAE	10.38	8.88
RMSE	20.34	11.66
MNAE (%)	11.97	**11.25**

Table 6.18 Comparison between ACFamNet Pro and other counting methods.

Metric	OpenCFU	AutoCellSeg	ACFamNet	ACFamNet Pro	Vanilla SAFECount
MAE	41.12	60.92	11.54	8.88	10.91
RMSE	47.76	69.87	15.56	11.66	14.64
MNAE (%)	46.57	68.73	12.52	**11.25**	13.73

ACFamNet Pro is able to produce 11.97% and 11.25% in training MNAE and test MNAE as presented in Table 6.17. This suggest ACFamNet Pro is able to generalise on unseen data. However, the test MNAE is slightly better than training validation which might be due to the small data set size. The comparison between ACFamNet Pro and other counting methods in Table 6.18 reveals that ACFamNet Pro is the best method with a MNAE in 11.25%. More importantly, ACFamNet Pro outperforms the vanilla SAFECount by 2.48% in MNAE. This means ACFamNet Pro is better than SAFECount when the comparison is conducted either in a hold-out fashion or 5-fold cross-validation shown in Table 6.16. Fig. 6.13 illustrates an example prediction from ACFamNet Pro.

6.3.6 Domain or category adaptation

Setup

Similar to the evaluation of ACFamNet's cross-category generalisation in § 6.2.6, this experiment aims to evaluate how well ACFamNet Pro count small and clustered colonies of a different category, which is part of the research question "How can the feature engineering

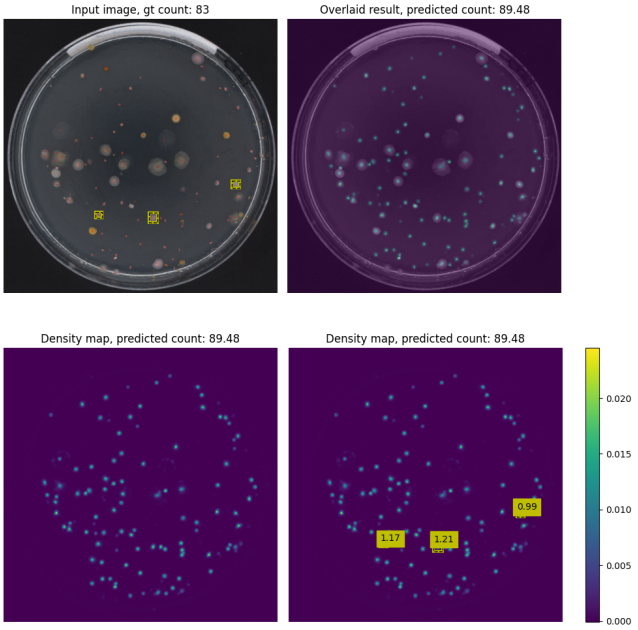


Fig. 6.13 Illustration of ACFamNet Pro's prediction. Predicted count and ground truth count are 89.5 and 83 respectively.

of SAFECount be transferred to FamNet or the modified FamNet to better adapt it to address small and clustered bacterial colonies, limited labelled data and cross domain/category generalisation since SAFECount is newer and superior to FamNet? And what has been the effect of the modified feature engineering on addressing these problems?". Four plate images shown in Fig 6.6 are used to evaluated the ACFamNet Pro obtained from § 6.3.5. The trained ACFamNet Pro and SAFECount are obtained from the experiment in § 6.3.5 to avoid repetitive training.

Results

Table 6.19 Results of ACFamNet Pro's cross-category prediction.

Plate image	Ground truth	ACFamNet Pro
Fig. 6.6a	228	211.02
Fig. 6.6b	124	285.85
Fig. 6.6c	529	257.14
Fig. 6.6d	302	324.28
MAE	RMSE	MNAE (%)
118.24	79.40	49.19

Table 6.20 Comparison of ACFamNet, ACFamNet Pro and SAFECount on cross-category generalisation.

Plate image	Ground truth	ACFamNet	ACFamNet Pro	SAFECount
Fig. 6.6a	228	306.31	211.02	124.82
Fig. 6.6b	124	475.85	285.85	78.19
Fig. 6.6c	529	3.1	257.14	310.79
Fig. 6.6d	302	832.12	324.28	241.98
MAE		371.55	118.24	106.81
RMSE		414.58	79.40	126.45
MNAE(%)		148.26	49.19	35.83

Table 6.19 lists detailed results of ACFamNet Pro's cross-category prediction. The MNAE from four plate images is 49.19%. It reveals ACFamNet Pro is able to readily generalise on colonies of a different category, even though ACFamNet Pro suffers the same three problems in data set: the overlap of colony categories between training images, the limited number of training images and the significant change of colour of plate areas between training images and evaluation images.

ACFamNet Pro's prediction on these four plage images are illustrated in Fig. 6.14, 6.15, 6.16, and 6.17. Among these four predictions, ACFamNet Pro has a very accurate prediction on Fig. 6.14 and 6.17. However, ACFamNet Pro's prediction on Fig. 6.15 and 6.16 attribute significantly to the overall MNAE. Several factors could explain this observation. Firstly, the multi-head attention mechanism, which has been proven effective in many natural language process applications [141] and computer vision applications [34], is able to help ACFamNet Pro to pay more attention to the object of interest dynamically. However, this attention is specified by exemplars which causes ACFamNet Pro to fail to count colonies that are not included in these three exemplars as shown in Fig. 6.15. Secondly, the dramatic change of colour of plate areas between training images and evaluation images as shown in Fig. 6.16 may still disrupt the model's ability to count colonies of a different species.

The comparison of ACFamNet, ACFamNet Pro and SAFECount on cross-category generalisation is presented in Table 6.20. It can be seen that SAFECount outperforms other two models in relation to the cross-category generalisation, producing 35.83% in MNAE. According to SAFECount's authors, this is due to the frozen backbone which prevents the model from being overly optimised on the training set, thus producing a better cross-category generalisation performance. However, SAFECount's better cross-category generalisation is at the cost of having a higher error when counting objects in a similar category as shown in Table 6.16 and 6.18. Additionally, the plate images in Fig. 6.6b and Fig. 6.6c may play a

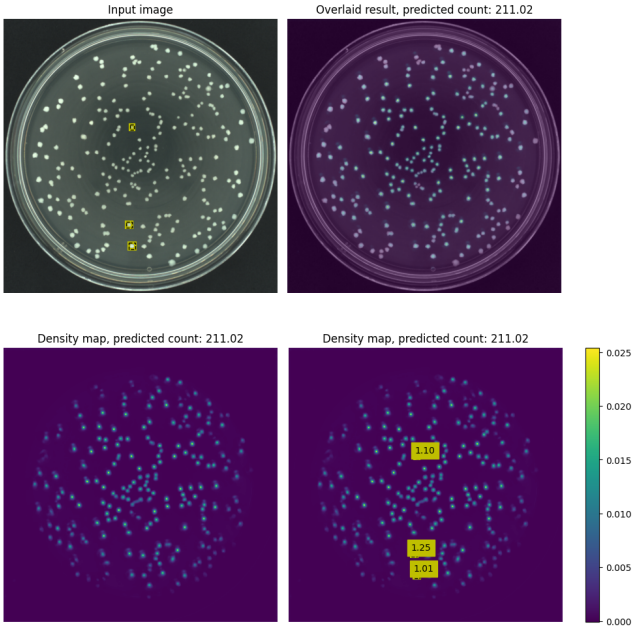


Fig. 6.14 Illustration of ACFamNet Pro's prediction on Fig. 6.6a. Predicted count and ground truth count are 211.02 and 228 respectively.

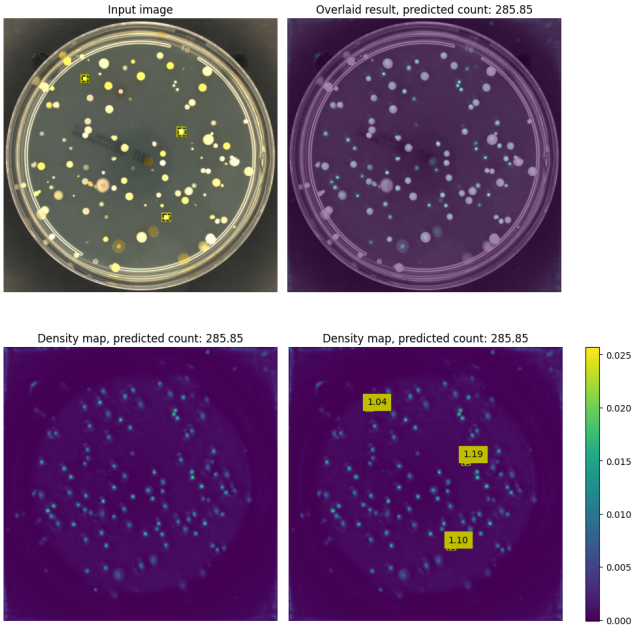


Fig. 6.15 Illustration of ACFamNet Pro's prediction on Fig. 6.6b. Predicted count and ground truth count are 285.85 and 124 respectively.

key role in SAFECount's better cross-category generalisation performance than ACFamNet Pro. This is because ACFamNet Pro does not perform well on these two images but excels in

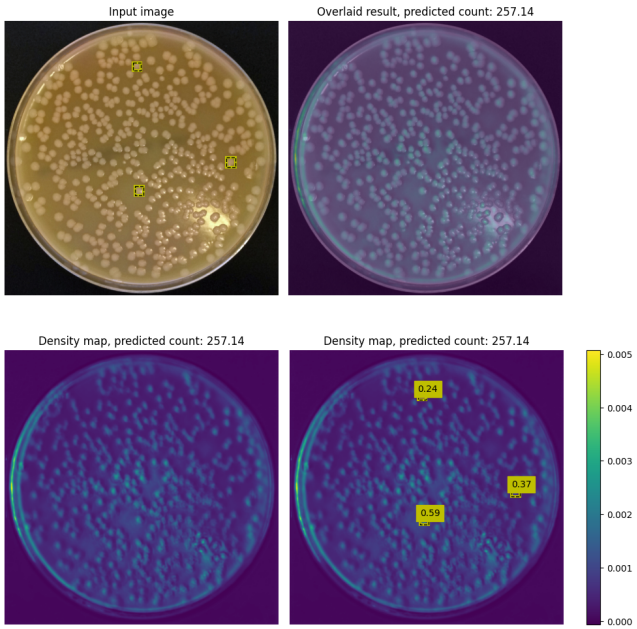


Fig. 6.16 Illustration of ACFamNet Pro's prediction on Fig. 6.6c. Predicted count and ground truth count are 257.14 and 529 respectively.

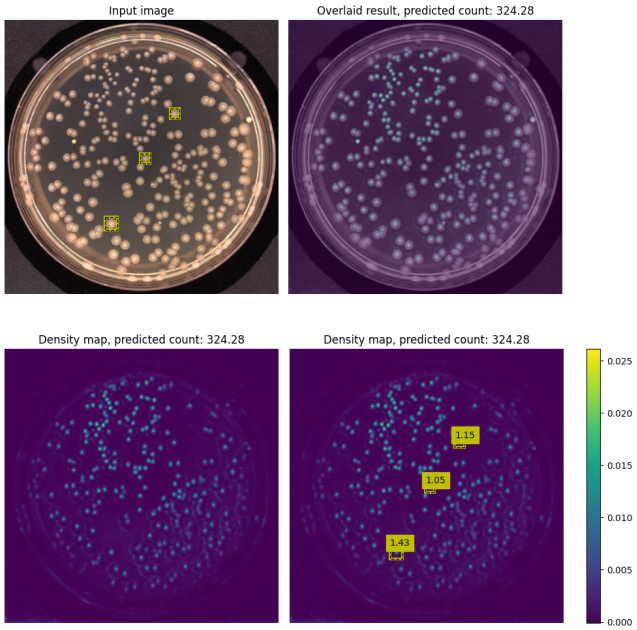


Fig. 6.17 Illustration of ACFamNet Pro's prediction on Fig. 6.6d. Predicted count and ground truth count are 324.28 and 302 respectively.

other two plate images shown in Fig. 6.6a and Fig 6.6d. SAFECount's prediction on these four plate image are illustrated in Appendix C.1.

6.3.7 Summary

This section has introduced ACFamNet Pro, which is an improved version of ACFamNet inspired by SAFECount, to count small and clustered colonies. The fine-tuned ACFamNet Pro generates an average of 9.62% in validation MNAE via 5-fold cross-validation with a standard deviation of 3.35% on Synoptics Dataset V2 training set. Ablation study uncovers that RoI align operation can improve the counting performance consistently. However, the residual similarity map and learnable backbone must co-work together to improve ACFamNet Pro. The synergy of RoI align, residual similarity map and learnable backbone reduces validation MNAE for ACFamNet Pro from 11.91% to 9.62% via the same 5-fold cross-validation setup. The fine-tuned ACFamNet Pro outperforms vanilla SAFECount and fine-tuned SAFECount by 0.24% and 0.17% in validation respectively. Finally, ACFamNet Pro is evaluated on Synoptics Dataset V2 test set after training from scratch on Synoptics Dataset V2 training set in a hold-out evaluation fashion. The MNAE computed from the test set is 11.25% which is 1.27%, 2.48%, 35.32% and 57.48% lower than ACFamNet, vanilla SAFECount, OpenCFU and AutoCellSeg respectively.

This section also has investigated two research questions listed in § 1. It has been empirically proved that SAFECount is able to count small bacterial colonies with a mean validation MNAE of 9.86% and a standard deviation of validation MNAE of 1.61% via a 5-fold cross-validation setup. Additionally, ACFamNet Pro, which is ACFamNet modified from FamNet with multi-head attention mechanism from SAFECount, residual connection and RoI align, can better count small and clustered colonies from limited labelled data. The performance gain is achieved by dynamically weighting objects of interest, optimising gradient flow and tackling region of interest misalignment. Meanwhile, ACFamNet Pro can readily generalise on colonies of a different category, producing a MNAE of 49.19% from four plate images that include completely different species of colonies. Moreover, the ablation study reveals that the RoI align is the key component attributed to ACFamNet Pro's outstanding performance on counting small and clustered colonies.

6.4 Conclusions

This chapter has conducted a series of experiments to evaluate the proposed ACFamNet and ACFamNet Pro. These experiments and the proposed algorithms aim to address the research gap that none of the existing counting methods is able to address small object, clustered objects, limited labelled data, and domain/category adaptation collectively. These two algorithms have been proven effective at learning from limited labelled data to count small and clustered colonies. However, ACFamNet fails to readily generalise to count colonies of a

different species which could be caused by three problems in the data. They are the overlap of colony categories between training images, the limited number of training images and the dramatic change of colour of plate areas between training images and evaluation images. Instead, ACFamNet Pro outperforms ACFamNet and becomes capable of counting colonies of a different species. This is mainly attributed from its additional multi-head attention mechanism and residual connection.

There are two main contributions to knowledge arise from this chapter. One is the proposal of ACFamNet, which is modified from FamNet with end-to-end trainable model, RoI align and optimised feature extraction module, to learn from limited labelled data to count small and clustered colonies. The performance gain from ACFamNet is achieved by solving region of interest misalignment and improving feature extraction. The other is the proposal of ACFamNet Pro, which is an advanced ACFamNet with additional multi-head attention mechanism and residual connection, to learn from limited labelled data to count small and clustered colonies, as well as being readily able to generalise on colonies of a different category. The performance gain from ACFamNet Pro is achieved by dynamically weighting objects of interest, optimising gradient flow and solving region of interest misalignment.

Chapter 7

Discussion and conclusions

7.1 Research outcomes

In this thesis, several solutions are proposed in an attempt to learn from limited labelled data to count small and clustered objects, as well as the model being readily generalisable to a different domain/category with an example application to bacterial colonies. Prior to this thesis, the existing counting approaches only address some aspects of the counting task. The main focus of this thesis is to develop a counting algorithm that is able to specifically address small and clustered objects and to generalise to different categories without a large set of labelled data.

This thesis starts with casting the counting task into a colony-cardinality classification task. Many attempts are made to interpret MicrobiaNet, which is the best-performing cardinality classification algorithm for colony counting to the best of my knowledge, by visualising network layer output, feature and class activation map. The class imbalance and high image similarity across classes are thoroughly investigated, uncovering that the latter is the main issue of counting colonies via cardinality classification.

Since the identification of the main issue of colony-cardinality classification, this thesis focuses on density map estimation based counting methods. Counting small and clustered objects is addressed by predicting a density map in which the density value represents the object count. This is because a density value can range from 0 to any positive number, which can provide an accurate count for small and clustered objects without detecting each of them individually. Few-shot learning is used to address the lack of annotated data and various domain/category adaptations because the model can exploit exemplars provided by users. A model named ACFamNet, which is an adaptation of FamNet, is proposed in this thesis. The synergy of single scale factor, end-to-end trainable network architecture and RoI align helps ACFamNet outperform FamNet by 10.48% in MNAE. The performance

gain from ACFamNet is achieved by tackling region of interest misalignment and improving feature extraction during the feature engineering process. It is also shown that ACFamNet is superior to non-machine learning based solutions, such as OpenCFU and AutoCellSeg, by 34.05% and 56.21% in MNAE. However, ACFamNet is not able to generalise to colonies of a different category for three possible reasons: the overlap of colony categories between training images, the limited number of training images and the significant change of colour of plate areas between training images and evaluation images.

With the high popularity of transformers in natural language processing applications, this thesis attempts to exploit the multi-head attention mechanism used in transformers which dynamically weights features of interest to improve ACFamNet. Inspired by a model named SAFECount, this thesis proposes ACFamNet Pro to improve the counting performance. ACFamNet Pro consists of the multi-head attention mechanism, residual connections and RoI align which are required to co-work together according to an ablation study. ACFamNet Pro outperforms SAFECount by 0.24% and 2.48% in validation MNAE when the evaluation is performed in the 5-fold cross-validation and hold-out evaluation respectively. Additionally, ACFamNet Pro outperforms ACFamNet by 2.23% in MNAE, as well as being able to readily generalise to colonies of a different species due to the multi-head attention mechanism. The performance gain is achieved by dynamically weighting objects of interest, optimising gradient flow and tackling region of interest misalignment. Similar to ACFamNet, ACFamNet Pro is also superior to non-machine learning based solutions, such as OpenCFU and AutoCellSeg, by 35.32% and 57.48% in MNAE.

7.2 Research limitations and future research directions

Although this thesis has proposed several solutions to learn from limited data to count small and clustered objects, as well as being readily generalisable to a different category, it has several limitations. The biggest limitation is that the best proposed algorithm ACFamNet Pro has yet to be adapted so that it is readily generalisable to a different category with a low error. Secondly, ACFamNet and ACFamNet Pro have yet to achieve a high level of automation. This is mainly because few-shot object counting requires users to provide some exemplars. In practice, exemplars can be stored in a local machine and reused for other tasks to achieve a semi-level of automation. Ideally, zero-shot object counting can be explored to fully automate the counting process.

There are several ideas for future work. Firstly, in order to improve ACFamNet Pro's performance, it could be further trained on a larger data set that contains a wide variety of small and clustered objects. Another way to improve ACFamNet Pro is to prune it by

evaluating the importance of each neuron and removing some of the least important neurons to make it smaller and faster. Secondly, it would be interesting to know if ACFamNet Pro can be used in other applications which similarly include small and clustered objects. For example, ACFamNet Pro might be useful, and potentially improve on current results in the existing bee counting studies [95, 119], fly counting studies [93, 165] and corn plant counting studies [99, 144]. Thirdly, it is also interesting to investigate if ACFamNet Pro can be used in applications based on satellite imagery. Counting objects, such as trees [1, 155], mammals [152, 76] and vehicles [82], from satellite imagery is useful for monitoring purposes. Furthermore, it may be worth investigating if ACFamNet Pro can be used on 3D objects given the fact that it is feasible to produce a 3D object image from an advanced image capturing system or a 2D object image nowadays [25]. Finally, it may be possible to exploit the depth information captured by a true depth camera, enriching features for the counting task.

7.3 Take-home messages

There are four take-home messages from this research. The key lesson is that, to count small objects, the algorithm must choose the aligned region of interest pooling rather than the conventional region of interest pooling to focus on small differences for better feature engineering. The second lesson is that the multi-head attention mechanism must be included in the algorithm to focus on the target objects to improve feature extraction. The third lesson is that the algorithm must be end-to-end trainable to improve gradient flow and performance, although at the cost of reducing cross-category generalisation. The final lesson is that interpretability and ablation studies are essential in artificial intelligence. Interpretability aims to make an artificial intelligence algorithm's decision transparent to human beings. Similarly, an ablation study investigates how different components of the whole artificial intelligence system contribute to the performance of the whole system by removing certain components. Without the interpretability, it would not have been possible to discover that the image similarity across classes rather than the class imbalance is the main issue for the cardinality classification method to count small and clustered colonies. Without the ablation studies, it would not have been possible to identify the key elements to count small and clustered objects which lead to the proposal of ACFamNet and ACFamNet Pro.

References

[1] Abozeid, A., Alanazi, R., Elhadad, A., Taloba, A. I., and Abd El-Aziz, R. M. (2022). A Large-Scale Dataset and Deep Learning Model for Detecting and Counting Olive Trees in Satellite Imagery. *Computational Intelligence and Neuroscience*, 2022(1):1549842.

[2] Agarap, A. F. (2018). Deep Learning using Rectified Linear Units (ReLU). *arXiv:1803.08375 [cs, stat]*.

[3] Amirgholipour, S., He, X., Jia, W., Wang, D., and Zeibots, M. (2018). A-CCNN: Adaptive CCNN for Density Estimation and Crowd Counting. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 948–952.

[4] Andreini, P., Bonechi, S., Bianchini, M., Mecocci, A., and Scarselli, F. (2018). A Deep Learning Approach to Bacterial Colony Segmentation. volume 11141 of *Lecture Notes in Computer Science*, pages 522–533. Springer International Publishing, Cham.

[5] Anthimopoulos, M., Christodoulidis, S., Ebner, L., Christe, A., and Mougiakakou, S. (2016). Lung Pattern Classification for Interstitial Lung Diseases Using a Deep Convolutional Neural Network. *IEEE Transactions on Medical Imaging*, 35(5):1207–1216.

[6] Ates, H. and Gerek, O. N. (2009). An image-processing based automated bacteria colony counter. In *2009 24th International Symposium on Computer and Information Sciences*, pages 18–23. IEEE.

[7] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer Normalization.

[8] Barber, P. R., Vojnovic, B., Kelly, J., Mayes, C. R., Boulton, P., Woodcock, M., and Joiner, M. C. (2001). Automated counting of mammalian cell colonies. *Physics in Medicine and Biology*, 46(1):63–76.

[9] Beck, B. R., Shin, B., Choi, Y., Park, S., and Kang, K. (2020). Predicting commercially available antiviral drugs that may act on the novel coronavirus (SARS-CoV-2) through a drug-target interaction deep learning model. *Computational and Structural Biotechnology Journal*, 18:784–790.

[10] Beznik, T., Smyth, P., de Lannoy, G., and Lee, J. A. (2020). Deep Learning to Detect Bacterial Colonies for the Production of Vaccines.

[11] Bochkovskiy, A., Wang, C. Y., and Liao, H. Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv*.

[12] Boominathan, L., Kruthiventi, S. S. S., and Babu, R. V. (2016). CrowdNet: A Deep Convolutional Network for Dense Crowd Counting. In *Proceedings of the 24th ACM international conference on Multimedia*, MM '16, pages 640–644, New York, NY, USA. Association for Computing Machinery.

[13] Brugger, S. D., Baumberger, C., Jost, M., Jenni, W., Brugger, U., and Mühlemann, K. (2012). Automated Counting of Bacterial Colony Forming Units on Agar Plates. *PLoS ONE*, 7(3):e33695.

[14] Chan, A. B. and Vasconcelos, N. (2009). Bayesian Poisson regression for crowd counting. In *2009 IEEE 12th International Conference on Computer Vision*, pages 545–551.

[15] Chang, K.-t. (2018). *Introduction to Geographic Information Systems*. McGraw Hill, 9th edition edition.

[16] Chattopadhay, A., Sarkar, A., Howlader, P., and Balasubramanian, V. N. (2018). Grad-CAM++: Generalized Gradient-Based Visual Explanations for Deep Convolutional Networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 839–847.

[17] Chen, C., Liu, M.-Y., Tuzel, O., and Xiao, J. (2017). R-CNN for small object detection. In Lai, S.-H., Lepetit, V., Nishino, K., and Sato, Y., editors, *Computer Vision – ACCV 2016*, volume 10115, pages 214–230. Springer International Publishing, Cham.

[18] Chen, J., Su, W., and Wang, Z. (2020). Crowd counting with crowd attention convolutional neural network. *Neurocomputing*, 382:210–220.

[19] Chen, W.-B. and Zhang, C. (2009). An automated bacterial colony counting and classification system. *Information Systems Frontiers*, 11(4):349–368.

[20] Cheng, G., Yuan, X., Yao, X., Yan, K., Zeng, Q., Xie, X., and Han, J. (2023). Towards Large-Scale Small Object Detection: Survey and Benchmarks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–20.

[21] Cheng, R. C. H. and Amin, N. A. K. (1983). Estimating Parameters in Continuous Univariate Distributions with a Shifted Origin. *Journal of the Royal Statistical Society. Series B (Methodological)*, 45(3):394–403.

[22] Chiang, P.-J., Tseng, M.-J., He, Z.-S., and Li, C.-H. (2015). Automated counting of bacterial colonies by image analysis. *Journal of Microbiological Methods*, 108:74–82.

[23] Choi, Y., Shin, B., Kang, K., Park, S., and Beck, B. R. (2020). Target-Centered Drug Repurposing Predictions of Human Angiotensin-Converting Enzyme 2 (ACE2) and Transmembrane Protease Serine Subtype 2 (TMPRSS2) Interacting Approved Drugs for Coronavirus Disease 2019 (COVID-19) Treatment through a Drug-Target Interaction Deep Learning Model. *Viruses*, 12(11):E1325.

[24] Choudhry, P. (2016). High-Throughput Method for Automated Colony and Cell Counting by Digital Image Analysis Based on Edge Detection. *PLOS ONE*, 11(2):e0148469.

[25] Choy, C. B., Xu, D., Gwak, J., Chen, K., and Savarese, S. (2016). 3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction.

[26] Clarke, M. L., Burton, R. L., Hill, A. N., Litorja, M., Nahm, M. H., and Hwang, J. (2010). Low-cost, high-throughput, automated counting of bacterial colonies. *Cytometry Part A*, 77A(8):790–797.

[27] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 160–167, New York, NY, USA. Association for Computing Machinery.

[28] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural Language Processing (Almost) from Scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

[29] Croarkin, C., Guthrie, W., Trutna, L., J. Filliben, J., Hembree, B., Moore, T., Hartley, S., and Hurwitz, A. (2008). NIST/SEMATECH e-Handbook of Statistical Methods.

[30] Dana H. Ballard (1981). Generalizing the Hough Transform to Detect Arbitrary Shapes. 13(2):111–122.

[31] Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization.

[32] Davies, A. C., Yin, J. H., and Velastin, S. A. (1995). Crowd monitoring using image processing. *Electronics & Communication Engineering Journal*, 7(1):37–47.

[33] De Angeli, K., Gao, S., Danciu, I., Durbin, E. B., Wu, X.-C., Stroup, A., Doherty, J., Schwartz, S., Wiggins, C., Damesyn, M., Coyle, L., Penberthy, L., Tourassi, G. D., and Yoon, H.-J. (2022). Class imbalance in out-of-distribution datasets: Improving the robustness of the TextCNN for the classification of rare cancer types. *Journal of Biomedical Informatics*, 125:103957.

[34] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929 [cs]*.

[35] Draelos, R. L. and Carin, L. (2021). Use HiResCAM instead of Grad-CAM for faithful explanations of convolutional neural networks.

[36] Fan, Z., Zhang, H., Zhang, Z., Lu, G., Zhang, Y., and Wang, Y. (2022). A survey of crowd counting and density estimation based on convolutional neural network. *Neurocomputing*, 472:224–251.

[37] Ferrari, A., Lombardi, S., and Signoroni, A. (2015). Bacterial colony counting by Convolutional Neural Networks. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, volume 2015-Novem, pages 7458–7461. IEEE.

[38] Ferrari, A., Lombardi, S., and Signoroni, A. (2017). Bacterial colony counting with Convolutional Neural Networks in Digital Microbiology Imaging. *Pattern Recognition*, 61:629–640.

[39] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1126–1135, Sydney, NSW, Australia. JMLR.org.

[40] Flaccavento, G., Lempitsky, V., Pope, I., Barber, P., Zisserman, A., Noble, J., and Vojnovic, B. (2011). Learning to Count Cells: applications to lens-free imaging of large fields. pages 1–6.

[41] Fu, J., Sun, X., Wang, Z., and Fu, K. (2021). An Anchor-Free Method Based on Feature Balancing and Refinement Network for Multiscale Ship Detection in SAR Images. *IEEE Transactions on Geoscience and Remote Sensing*, 59(2):1331–1344.

[42] Fu, R., Hu, Q., Dong, X., Guo, Y., Gao, Y., and Li, B. (2020). Axiom-based Grad-CAM: Towards Accurate Visualization and Explanation of CNNs.

[43] Gao, G., Gao, J., Liu, Q., Wang, Q., and Wang, Y. (2020a). CNN-based Density Estimation and Crowd Counting: A Survey. *arXiv:2003.12783 [cs]*.

[44] Gao, G., Liu, Q., and Wang, Y. (2020b). Counting Dense Objects in Remote Sensing Images. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4137–4141.

[45] Geissmann, Q. (2013). OpenCFU, a New Free and Open-Source Software to Count Cell Colonies and Other Circular Objects. *PLoS ONE*, 8(2):1–10.

[46] Gildenblat, J. and contributors (2021). Pytorch library for cam methods. https://github.com/jacobgil/pytorch-grad-cam.

[47] Girshick, R. (2015). Fast r-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:1440–1448.

[48] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587.

[49] Glasmachers, T. (2017). Limits of End-to-End Learning.

[50] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research*, 9:249–256.

[51] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. The MIT Press.

[52] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, 63(11):139–144.

[53] Guo, D., Li, K., Zha, Z.-J., and Wang, M. (2019). DADNet: Dilated-Attention-Deformable ConvNet for Crowd Counting. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, pages 1823–1832, New York, NY, USA. Association for Computing Machinery.

[54] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer New York, New York, second edi edition.

[55] He, K., Gkioxari, G., Dollar, P., and Girshick, R. (2017). Mask R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:2980–2988.

[56] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:1026–1034.

[57] Hilsenbeck, O., Schwarzfischer, M., Loeffler, D., DImopoulos, S., Hastreiter, S., Marr, C., Theis, F. J., and Schroeder, T. (2017). FastER: A User-Friendly tool for ultrafast and robust cell segmentation in large-scale microscopy. *Bioinformatics*, 33(13):2020–2028.

[58] Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. (2022). Meta-Learning in Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5149–5169.

[59] Hossain, M., Hosseinzadeh, M., Chanda, O., and Wang, Y. (2019). Crowd Counting Using Scale-Aware Attention Networks. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1280–1288.

[60] Hu, Y., Jiang, X., Liu, X., Zhang, B., Han, J., Cao, X., and Doermann, D. (2020). NAS-Count: Counting-by-Density with Neural Architecture Search. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII*, pages 747–766, Berlin, Heidelberg. Springer-Verlag.

[61] Huang, S., Li, X., Zhang, Z., Wu, F., Gao, S., Ji, R., and Han, J. (2018). Body Structure Aware Deep Crowd Counting. *IEEE Transactions on Image Processing*, 27(3):1049–1059.

[62] Ioffe, S. and Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456, Lille, France. JMLR.org.

[63] Jeanson, S., Floury, J., Gagnaire, V., Lortal, S., and Thierry, A. (2015). Bacterial Colonies in Solid Media and Foods: A Review on Their Growth and Interactions with the Micro-Environment. *Frontiers in Microbiology*, 6(December).

[64] Jia Deng, Wei Dong, Socher, R., Li-Jia Li, Kai Li, and Li Fei-Fei (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

[65] Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., NanoCode012, Kwon, Y., Michael, K., TaoXie, Fang, J., and Imyhxy (2022). ultralytics/yolov5: v7.0 - yolov5 sota realtime instance segmentation.

[66] Johnson, J. M. and Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):1–54.

[67] Kachouie, N. N., Kang, L., and Khademhosseini, A. (2009). Arraycount, an algorithm for automatic cell counting in microwell arrays. *BioTechniques*, 47(3S):x–xvi.

[68] Kang, D. and Chan, A. (2018). Crowd Counting by Adaptively Fusing Predictions from an Image Pyramid.

[69] Khan, A. U. M., Torelli, A., Wolf, I., and Gretz, N. (2018). AutoCellSeg: robust automatic colony forming unit (CFU)/cell analysis using adaptive image segmentation and easy-to-use post-editing techniques. *Scientific Reports*, 8(1):7302.

[70] Kiefer, J. and Wolfowitz, J. (1956). Consistency of the Maximum Likelihood Estimator in the Presence of Infinitely Many Incidental Parameters. *The Annals of Mathematical Statistics*, 27(4):887–906.

[71] Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15.

[72] Kong, X., Zhao, M., Zhou, H., and Zhang, C. (2020). Weakly Supervised Crowd-Wise Attention For Robust Crowd Counting. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2722–2726.

[73] Krishna, H. and Jawahar, C. (2017). Improving Small Object Detection. In *2017 4th IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 340–345.

[74] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *NIPS'12: Proceedings of the 25th International Conference on Neural Information Processing Systems*, volume 1, pages 1097–1105.

[75] Kullback, S. and Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

[76] Laradji, I., Rodriguez, P., Kalaitzis, F., Vazquez, D., Young, R., Davey, E., and Lacoste, A. (2020). Counting Cows: Tracking Illegal Cattle Ranching From High-Resolution Satellite Imagery.

[77] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[78] Lempitsky, V. and Zisserman, A. (2010). Learning to count objects in images. *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010, NIPS 2010*, pages 1–9.

[79] Lewis, M., Yarats, D., Dauphin, Y., Parikh, D., and Batra, D. (2017). Deal or No Deal? End-to-End Learning of Negotiation Dialogues. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2443–2453, Copenhagen, Denmark. Association for Computational Linguistics.

[80] Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., Ke, Z., Li, Q., Cheng, M., Nie, W., Li, Y., Zhang, B., Liang, Y., Zhou, L., Xu, X., Chu, X., Wei, X., and Wei, X. (2022). YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications.

[81] Liang, Z., Shao, J., Zhang, D., and Gao, L. (2018). Small Object Detection Using Deep Feature Pyramid Networks. In Hong, R., Cheng, W.-H., Yamasaki, T., Wang, M., and Ngo, C.-W., editors, *Advances in Multimedia Information Processing – PCM 2018*, Lecture Notes in Computer Science, pages 554–564, Cham. Springer International Publishing.

[82] Liao, L., Xiao, J., Yang, Y., Ma, X., Wang, Z., and Satoh, S. (2023). High temporal frequency vehicle counting from low-resolution satellite images. *ISPRS Journal of Photogrammetry and Remote Sensing*, 198:45–59.

[83] Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017). Feature Pyramid Networks for Object Detection.

[84] Liu, C., Weng, X., and Mu, Y. (2019a). Recurrent Attentive Zooming for Joint Crowd Counting and Precise Localization. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1217–1226.

[85] Liu, F. and Yang, L. (2017). A Novel Cell Detection Method Using Deep Convolutional Neural Network and Maximum-Weight Independent Set. In *Advances in Computer Vision and Pattern Recognition*, volume 9351, pages 63–72.

[86] Liu, N., Long, Y., Zou, C., Niu, Q., Pan, L., and Wu, H. (2019b). ADCrowdNet: An Attention-Injective Deformable Convolutional Network for Crowd Understanding. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3220–3229, Long Beach, CA, USA. IEEE.

[87] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). SSD: Single shot MultiBox detector. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 21–37, Cham. Springer International Publishing.

[88] Liu, W., Salzmann, M., and Fua, P. (2019c). Context-Aware Crowd Counting. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5094–5103.

[89] Loukas, C. G., Wilson, G. D., Vojnovic, B., and Linney, A. (2003). An image analysis-based approach for automated counting of cancer cell nuclei in tissue sections. *Cytometry*, 55A(1):30–42.

[90] Lu, X., Ji, J., Xing, Z., and Miao, Q. (2021). Attention and Feature Fusion SSD for Remote Sensing Object Detection. *IEEE Transactions on Instrumentation and Measurement*, 70:1–9.

[91] Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

[92] Maaten, L. v. d. and Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605.

[93] Mamdouh, N. and Khattab, A. (2021). YOLO-Based Deep Learning Framework for Olive Fruit Fly Detection and Counting. *IEEE Access*, 9:84252–84262.

[94] Marsden, M., McGuinness, K., Little, S., and O'Connor, N. E. (2017). ResnetCrowd: A residual deep learning architecture for crowd counting, violent behaviour detection and crowd density level classification. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–7.

[95] Marstaller, J., Tausch, F., and Stock, S. (2019). DeepBees - Building and Scaling Convolutional Neuronal Nets For Fast and Large-Scale Visual Monitoring of Bee Hives. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 271–278.

[96] Matic, T., Vidovic, I., Siladi, E., and Tkalec, F. (2016). Semi-automatic prototype system for bacterial colony counting. In *2016 International Conference on Smart Systems and Technologies (SST)*, pages 205–210. IEEE.

[97] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[98] Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2022). A Simple Neural Attentive Meta-Learner.

[99] Mota-Delfin, C., López-Canteñs, G. d. J., López-Cruz, I. L., Romantchik-Kriuchkova, E., and Olguín-Rojas, J. C. (2022). Detection and Counting of Corn Plants in the Presence of Weeds with Convolutional Neural Networks. *Remote Sensing*, 14(19):4892.

[100] Muhammad, M. B. and Yeasin, M. (2020). Eigen-CAM: Class Activation Map using Principal Components. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7.

[101] Naets, T., Huijsmans, M., and Sorber, L. (2020). An agile machine learning project in pharma - developing a Mask R-CNN-based web application for bacterial colony counting. (October):2–4.

[102] Niyazi, M., Niyazi, I., and Belka, C. (2007). Counting colonies of clonogenic assays by using densitometric software. *Radiation Oncology*, 2(1):3–5.

[103] Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature visualization. *Distill*.

[104] Olivas, E. S., Guerrero, J. D. M., Sober, M. M., Benedito, J. R. M., and Lopez, A. J. S. (2009). *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques*. Information Science Reference, Hershey, Pa, 1st edition edition.

[105] Olmschenk, G., Chen, J., Tang, H., and Zhu, Z. (2019). Dense Crowd Counting Convolutional Neural Networks with Minimal Data using Semi-Supervised Dual-Goal Generative Adversarial Networks. *IEEE Conference on Computer Vision and Pattern Recognition: Learning with Imperfect Data Workshop*.

[106] Pan, X., Mo, H., Zhou, Z., and Wu, W. (2020). Attention Guided Region Division for Crowd Counting. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2568–2572.

[107] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. page 12.

[108] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., and Cournapeau, D. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[109] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151.

[110] Qiao, S., Wang, H., Liu, C., Shen, W., and Yuille, A. (2020). Micro-Batch Training with Batch-Channel Normalization and Weight Standardization.

[111] Qin, Y., Wang, W., Liu, W., and Yuan, N. (2013). Extended-maxima transform watershed segmentation algorithm for touching corn kernels. *Advances in Mechanical Engineering*, 2013:268046.

[112] Ranjan, V., Sharma, U., Nguyen, T., and Hoai, M. (2021). Learning To Count Everything. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3393–3402.

[113] Raschka, S. (2020). Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. *arXiv:1811.12808 [cs, stat]*.

[114] Ravi, S. and Larochelle, H. (2017). Optimization as a Model for Few-Shot Learning. page 11.

[115] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:779–788.

[116] Redmon, J. and Farhadi, A. (2016). YOLO9000: Better, Faster, Stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017-Janua:6517–6525.

[117] Redmon, J. and Farhadi, A. (2018). YOLOv3: An Incremental Improvement.

[118] Ren, S., He, K., Girshick, R., and Sun, J. (2017). Faster r-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149.

[119] Rodriguez, I. F., Megret, R., Acuna, E., Agosto-Rivera, J. L., and Giray, T. (2018). Recognition of Pollen-Bearing Bees from Video Using Convolutional Neural Network. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 314–322.

[120] Rosenblatt, F. (1957). *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory.

[121] Ruder, S. (2017). An overview of gradient descent optimization algorithms.

[122] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

[123] Sadanandan, S. K., Ranefall, P., Le Guyader, S., and Wählby, C. (2017). Automated Training of Deep Convolutional Neural Networks for Cell Segmentation. *Scientific Reports*, 7(1):1–7.

[124] Salahuddin, Z., Woodruff, H. C., Chatterjee, A., and Lambin, P. (2022). Transparency of deep neural networks for medical image analysis: A review of interpretability methods. *Computers in Biology and Medicine*, 140:105111.

[125] Salimans, T. and Kingma, D. P. (2016). Weight normalization: a simple reparameterization to accelerate training of deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 901–909, Red Hook, NY, USA. Curran Associates Inc.

[126] Saxena, D. and Cao, J. (2021). Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions. *ACM Computing Surveys*, 54(3):63:1–63:42.

[127] Schmidhuber, J. (1987). *Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook*. Diploma Thesis, Technische Universitat Munchen, Germany.

[128] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 618–626.

[129] Serban, A. C., Poll, E., and Visser, J. (2018). A Standard Driven Software Architecture for Fully Autonomous Vehicles. In *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 120–127.

[130] Shi, M., Yang, Z., Xu, C., and Chen, Q. (2019). Revisiting Perspective Information for Efficient Crowd Counting. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7271–7280.

[131] Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition.

[132] Sindagi, V. A. and Patel, V. M. (2019). Inverse Attention Guided Deep Crowd Counting Network. In *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–8, Taipei, Taiwan. IEEE.

[133] Sindagi, V. A. and Patel, V. M. (2020). HA-CCN: Hierarchical Attention-Based Crowd Counting Network. *IEEE Transactions on Image Processing*, 29:323–335.

[134] Smith, K. and Horvath, P. (2014). Active learning strategies for phenotypic profiling of high-content screens. *Journal of Biomolecular Screening*, 19(5):685–695.

[135] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[136] Sun, Y., Wong, A. K. C., and Kamel, M. S. (2009). CLASSIFICATION OF IMBALANCED DATA: A REVIEW. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(04):687–719.

[137] Thrun, S. and Pratt, L. (1998). Learning to Learn: Introduction and Overview. In Thrun, S. and Pratt, L., editors, *Learning to Learn*, pages 3–17. Springer US, Boston, MA.

[138] Tian, Z., Shen, C., Chen, H., and He, T. (2022). FCOS: A Simple and Strong Anchor-Free Object Detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(4):1922–1933.

[139] Tong, K., Wu, Y., and Zhou, F. (2020). Recent advances in small object detection based on deep learning: A review. *Image and Vision Computing*, 97:103910.

[140] Uijlings, J. R. R., van de Sande, K. E. A., Gevers, T., and Smeulders, A. W. M. (2013). Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2):154–171.

[141] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

[142] Wan, J. and Chan, A. (2019). Adaptive Density Map Generation for Crowd Counting. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1130–1139, Seoul, Korea (South). IEEE.

[143] Wang, C.-Y., Bochkovskiy, A., and Liao, H.-Y. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.

[144] Wang, L., Xiang, L., Tang, L., and Jiang, H. (2021). A Convolutional Neural Network-Based Method for Corn Stand Counting in the Field. *Sensors*, 21(2):507.

[145] Wang, Q., Gao, J., Lin, W., and Yuan, Y. (2019). Learning from synthetic data for crowd counting in the wild. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:8190–8199.

[146] Wattenberg, M., Viégas, F., and Johnson, I. (2016). How to use t-sne effectively. *Distill*.

[147] Wienert, S., Heim, D., Saeger, K., Stenzinger, A., Beil, M., Hufnagl, P., Dietel, M., Denkert, C., and Klauschen, F. (2012). Detection and segmentation of cell nuclei in virtual microscopy images: A minimum-model approach. *Scientific Reports*, 2:1–7.

[148] Wong, C.-F., Yeo, J. Y., and Gan, S. K.-e. (2016). APD Colony Counter App : Using Watershed Algorithm for improved colony counting. *Nature Methods Application Notes*, (August):1–3.

[149] Wu, Y. and He, K. (2018). Group Normalization. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision – ECCV 2018*, Lecture Notes in Computer Science, pages 3–19, Cham. Springer International Publishing.

[150] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.

[151] Xu, C., Qiu, K., Fu, J., Bai, S., Xu, Y., and Bai, X. (2019). Learn to Scale: Generating Multipolar Normalized Density Maps for Crowd Counting. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8381–8389.

[152] Xue, Y., Wang, T., and Skidmore, A. K. (2017). Automatic Counting of Large Mammals from Very High Resolution Panchromatic Satellite Imagery. *Remote Sensing*, 9(9):878.

[153] Yan, Z., Yuan, Y., Zuo, W., Tan, X., Wang, Y., Wen, S., and Ding, E. (2019). Perspective-Guided Convolution Networks for Crowd Counting. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 952–961.

[154] Yang, X., Yang, J., Yan, J., Zhang, Y., Zhang, T., Guo, Z., Sun, X., and Fu, K. (2019). SCRDet: Towards More Robust Detection for Small, Cluttered and Rotated Objects. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8231–8240.

[155] Yao, L., Liu, T., Qin, J., Lu, N., and Zhou, C. (2021). Tree counting with high spatial-resolution satellite imagery based on deep neural networks. *Ecological Indicators*, 125:107591.

[156] You, Z., Yang, K., Luo, W., Lu, X., Cui, L., and Le, X. (2023). Few-shot Object Counting with Similarity-Aware Feature Enhancement. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 6304–6313.

[157] Zhang, A., Shen, J., Xiao, Z., Zhu, F., Zhen, X., Cao, X., and Shao, L. (2019a). Relational Attention Network for Crowd Counting. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6787–6796.

[158] Zhang, A., Yue, L., Shen, J., Zhu, F., Zhen, X., Cao, X., and Shao, L. (2019b). Attentional Neural Fields for Crowd Counting. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5713–5722.

[159] Zhang, C. and Chen, W.-B. (2007). An Effective and Robust Method for Automatic Bacterial Colony Enumeration. In *International Conference on Semantic Computing (ICSC 2007)*, pages 581–588. IEEE.

[160] Zhang, M., Zhao, W., Li, X., and Wang, D. (2020). Shadow Detection Of Moving Objects In Traffic Monitoring Video. In *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, volume 9, pages 1983–1987.

[161] Zhang, S., Zhou, J., Hu, H., Gong, H., Chen, L., Cheng, C., and Zeng, J. (2016a). A deep learning framework for modeling structural features of RNA-binding protein targets. *Nucleic Acids Research*, 44(4):e32.

[162] Zhang, Y., Zhou, C., Chang, F., and Kot, A. C. (2019c). Multi-resolution attention convolutional neural network for crowd counting. *Neurocomputing*, 329:144–152.

[163] Zhang, Y., Zhou, D., Chen, S., Gao, S., and Ma, Y. (2016b). Single-Image Crowd Counting via Multi-Column Convolutional Neural Network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 589–597.

[164] Zheng, A. (2018). *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly, Beijing : Boston.

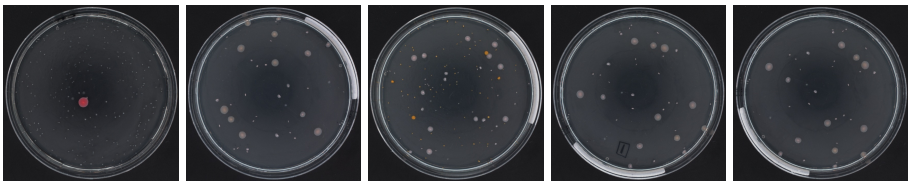
[165] Zhong, Y., Gao, J., Lei, Q., and Zhou, Y. (2018). A Vision-Based Counting and Recognition System for Flying Insects in Intelligent Agriculture. *Sensors*, 18(5):1489.

[166] Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2015). Learning Deep Features for Discriminative Localization. *arXiv:1512.04150 [cs]*.

Appendix A

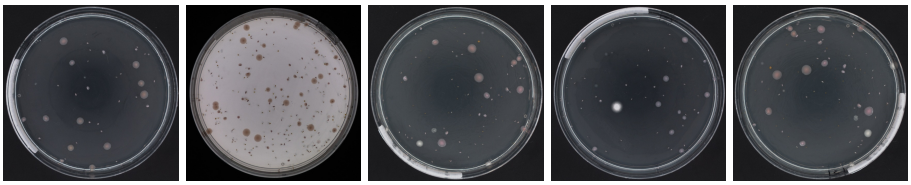
Demonstration of Synoptics Dataset V2

A.1 Synoptics Dataset V2



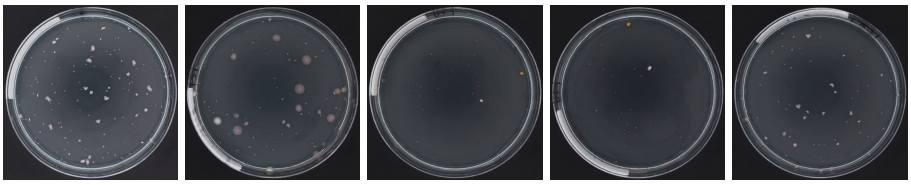
(a) 1.jpg. (b) 2.jpg. (c) 3.jpg. (d) 4.jpg. (e) 5.jpg.

Fig. A.1 Demonstration of Synoptics Dataset V2 images 1 - 5.



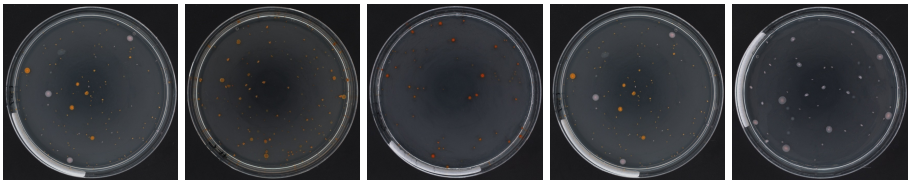
(a) 6.jpg. (b) 7.jpg. (c) 8.jpg. (d) 9.jpg. (e) 10.jpg.

Fig. A.2 Demonstration of Synoptics Dataset V2 images 6 - 10.



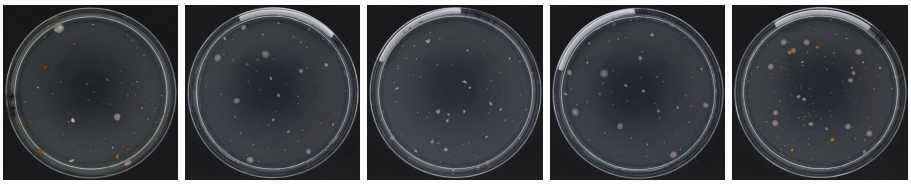
(a) 11.jpg. (b) 12.jpg. (c) 13.jpg. (d) 14.jpg. (e) 15.jpg.

Fig. A.3 Demonstration of Synoptics Dataset V2 images 11 - 15.



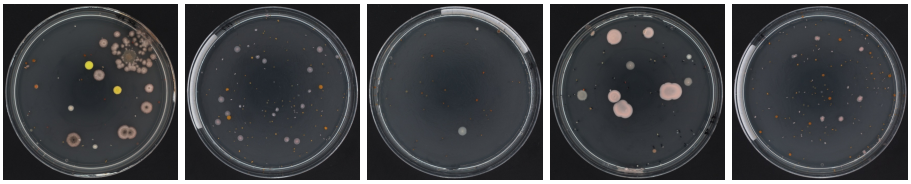
(a) 16.jpg. (b) 17.jpg. (c) 18.jpg. (d) 19.jpg. (e) 20.jpg.

Fig. A.4 Demonstration of Synoptics Dataset V2 images 16 - 20.



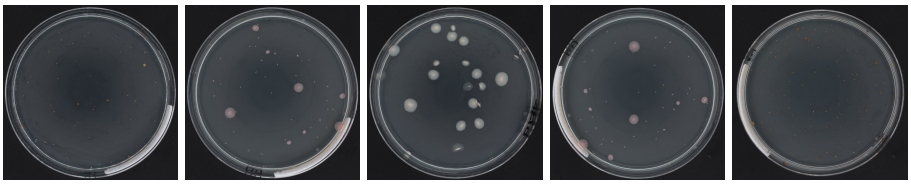
(a) 21.jpg. (b) 22.jpg. (c) 23.jpg. (d) 24.jpg. (e) 25.jpg.

Fig. A.5 Demonstration of Synoptics Dataset V2 images 21 - 25.



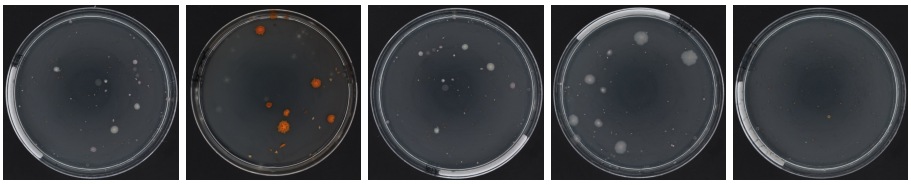
(a) 26.jpg. (b) 27.jpg. (c) 28.jpg. (d) 29.jpg. (e) 30.jpg.

Fig. A.6 Demonstration of Synoptics Dataset V2 images 26 - 30.



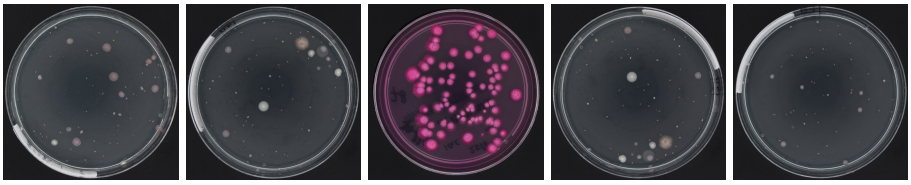
(a) 31.jpg. (b) 32.jpg. (c) 33.jpg. (d) 34.jpg. (e) 35.jpg.

Fig. A.7 Demonstration of Synoptics Dataset V2 images 31 - 35.



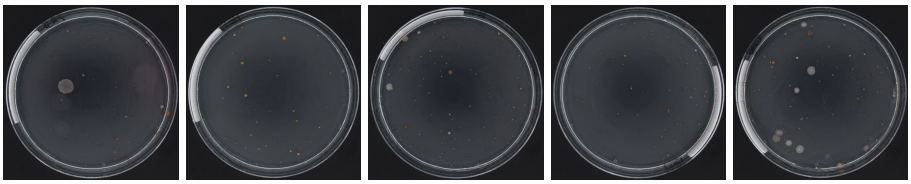
(a) 36.jpg. (b) 37.jpg. (c) 38.jpg. (d) 39.jpg. (e) 40.jpg.

Fig. A.8 Demonstration of Synoptics Dataset V2 images 36 - 40.



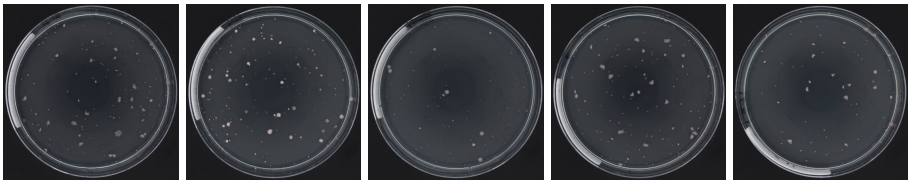
(a) 41.jpg. (b) 42.jpg. (c) 43.jpg. (d) 44.jpg. (e) 45.jpg.

Fig. A.9 Demonstration of Synoptics Dataset V2 images 41 - 45.



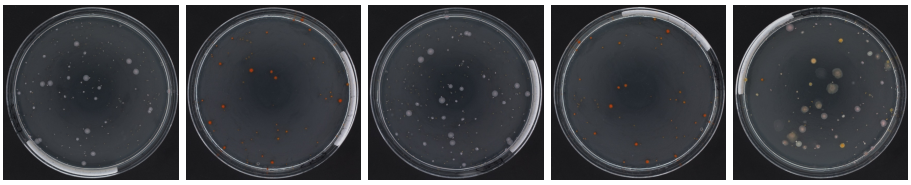
(a) 46.jpg. (b) 47.jpg. (c) 48.jpg. (d) 49.jpg. (e) 50.jpg.

Fig. A.10 Demonstration of Synoptics Dataset V2 images 46 - 50.



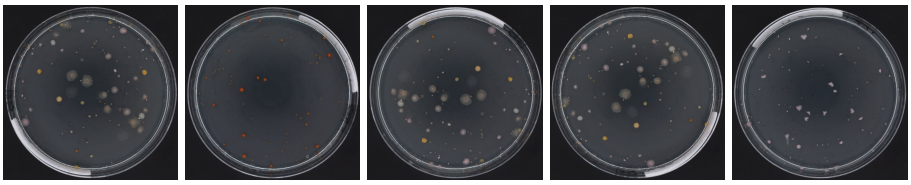
(a) 51.jpg. (b) 52.jpg. (c) 53.jpg. (d) 54.jpg. (e) 55.jpg.

Fig. A.11 Demonstration of Synoptics Dataset V2 images 51 - 55.



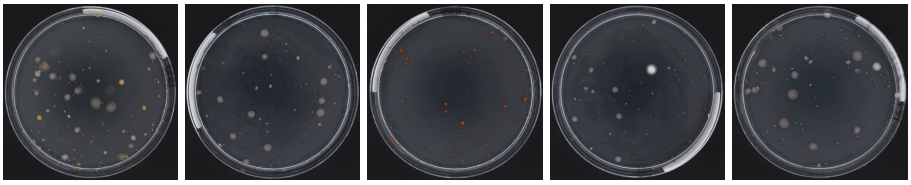
(a) 56.jpg. (b) 57.jpg. (c) 58.jpg. (d) 59.jpg. (e) 60.jpg.

Fig. A.12 Demonstration of Synoptics Dataset V2 images 56 - 60.



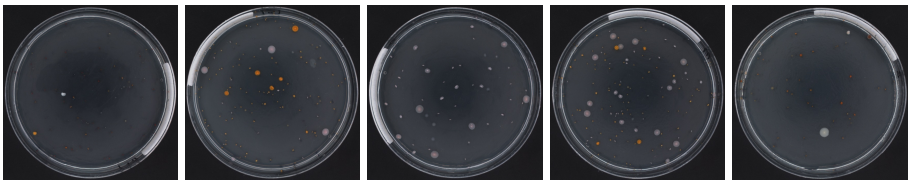
(a) 61.jpg. (b) 62.jpg. (c) 63.jpg. (d) 64.jpg. (e) 65.jpg.

Fig. A.13 Demonstration of Synoptics Dataset V2 images 61 - 65.



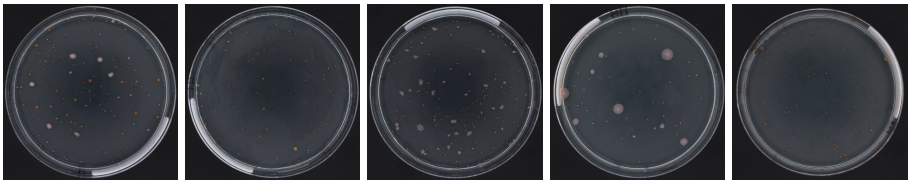
(a) 66.jpg. (b) 67.jpg. (c) 68.jpg. (d) 69.jpg. (e) 70.jpg.

Fig. A.14 Demonstration of Synoptics Dataset V2 images 66 - 70.



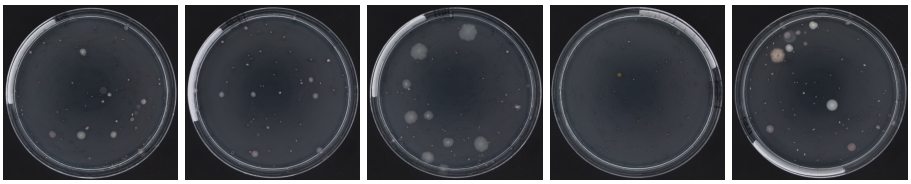
(a) 71.jpg. (b) 72.jpg. (c) 73.jpg. (d) 74.jpg. (e) 75.jpg.

Fig. A.15 Demonstration of Synoptics Dataset V2 images 71 - 75.



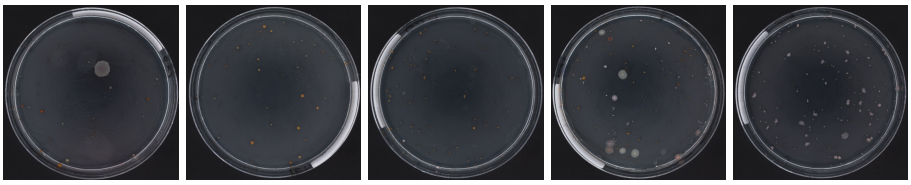
(a) 76.jpg. (b) 77.jpg. (c) 78.jpg. (d) 79.jpg. (e) 80.jpg.

Fig. A.16 Demonstration of Synoptics Dataset V2 images 76 - 80.



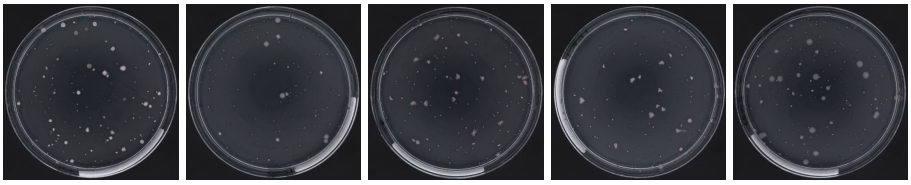
(a) 81.jpg. (b) 82.jpg. (c) 83.jpg. (d) 84.jpg. (e) 85.jpg.

Fig. A.17 Demonstration of Synoptics Dataset V2 images 81 - 85.



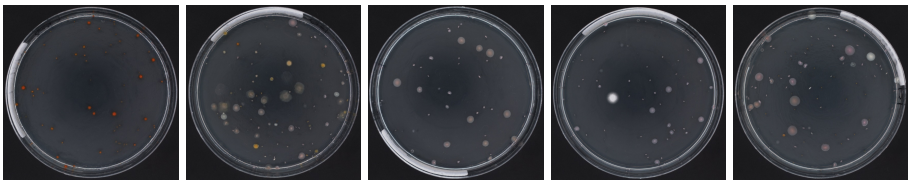
(a) 86.jpg. (b) 87.jpg. (c) 88.jpg. (d) 89.jpg. (e) 90.jpg.

Fig. A.18 Demonstration of Synoptics Dataset V2 images 86 - 90.



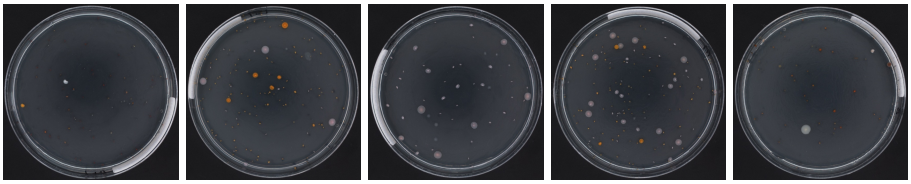
(a) 91.jpg. (b) 92.jpg. (c) 93.jpg. (d) 94.jpg. (e) 95.jpg.

Fig. A.19 Demonstration of Synoptics Dataset V2 images 91 - 95.



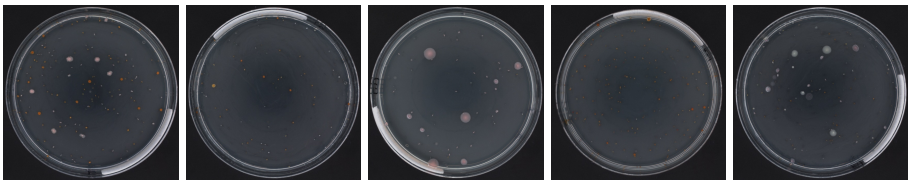
(a) 96.jpg. (b) 97.jpg. (c) 89.jpg. (d) 99.jpg. (e) 100.jpg.

Fig. A.20 Demonstration of Synoptics Dataset V2 images 96 - 100.



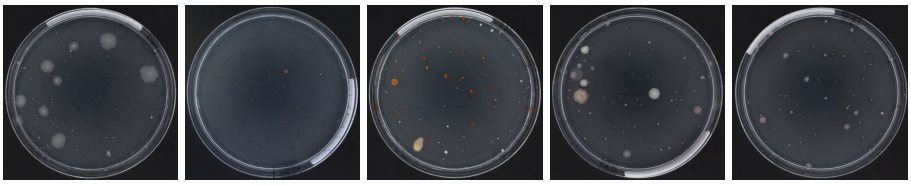
(a) 101.jpg. (b) 102.jpg. (c) 103.jpg. (d) 104.jpg. (e) 105.jpg.

Fig. A.21 Demonstration of Synoptics Dataset V2 images 101 - 105.



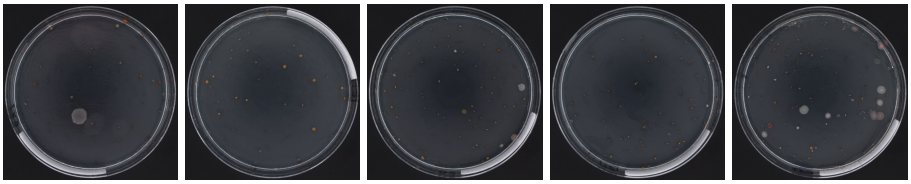
(a) 106.jpg. (b) 107.jpg. (c) 108.jpg. (d) 109.jpg. (e) 110.jpg.

Fig. A.22 Demonstration of Synoptics Dataset V2 images 106 - 110.



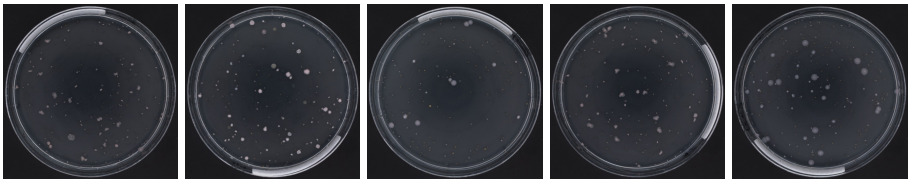
(a) 111.jpg. (b) 112.jpg. (c) 113.jpg. (d) 114.jpg. (e) 115.jpg.

Fig. A.23 Demonstration of Synoptics Dataset V2 images 111 - 115.



(a) 116.jpg. (b) 117.jpg. (c) 118.jpg. (d) 119.jpg. (e) 120.jpg.

Fig. A.24 Demonstration of Synoptics Dataset V2 images 116 - 120.



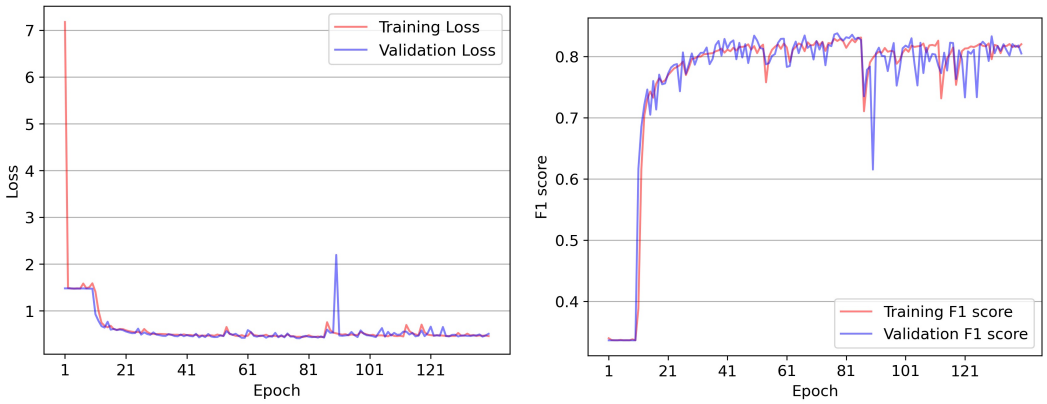
(a) 121.jpg. (b) 122.jpg. (c) 123.jpg. (d) 124.jpg. (e) 125.jpg.

Fig. A.25 Demonstration of Synoptics Dataset V2 images 121 - 125.

Appendix B

Supplementary material for counting by cardinality classification

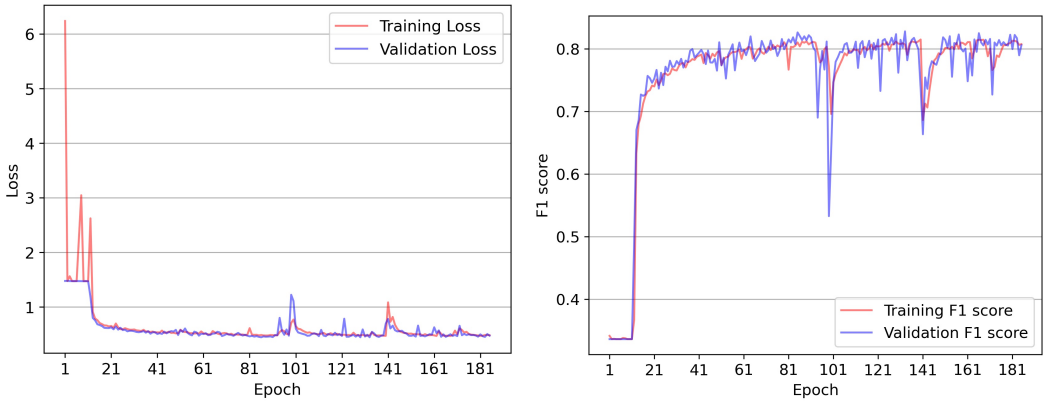
B.1 Results evaluated on other Microbia data sets



(a) Loss curves.

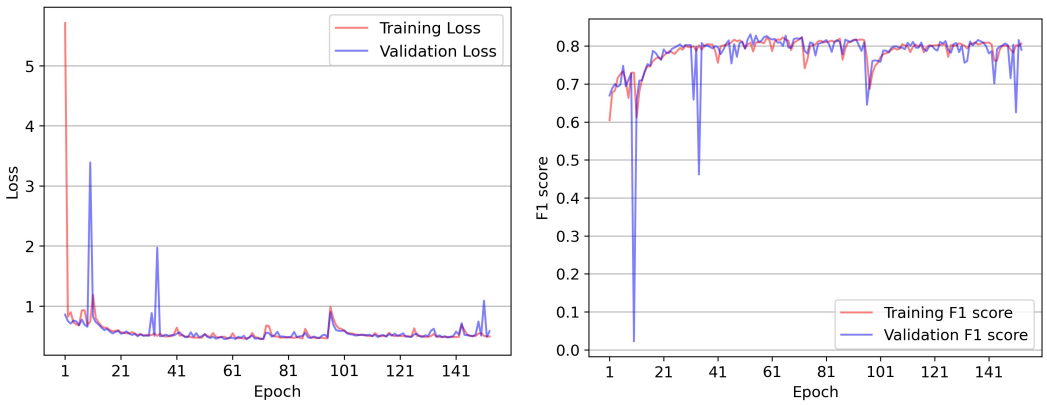
(b) F1 score curves.

Fig. B.1 Loss value and F1 score throughout the training process obtained from MicrobiaS2 data set.



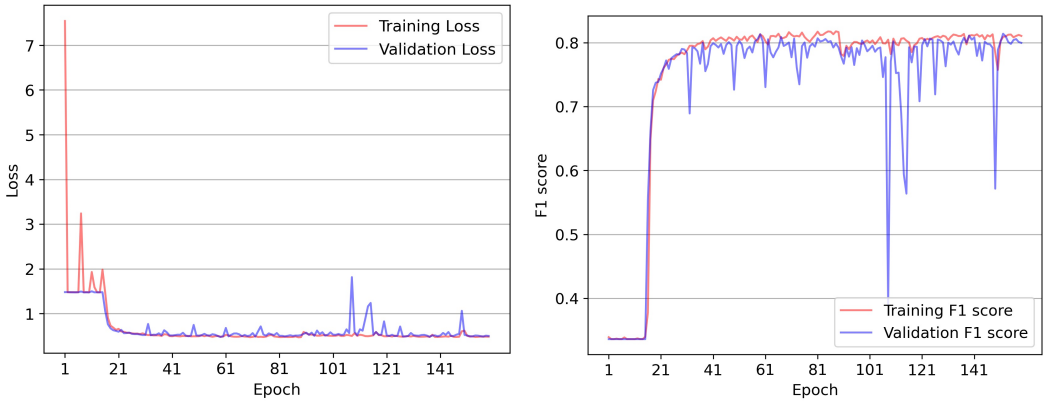
(a) Loss curves. (b) F1 score curves.

Fig. B.2 Loss value and F1 score throughout the training process obtained from MicrobiaS3 data set.



(a) Loss curves. (b) F1 score curves.

Fig. B.3 Loss value and F1 score throughout the training process obtained from MicrobiaS4 data set.



(a) Loss curves.

(b) F1 score curves.

Fig. B.4 Loss value and F1 score throughout the training process obtained from MicrobiaS5 data set.

Table B.1 Classification results evaluated on MicrobiaS2 validation set.

Class Name	Precision	Recall	F1 score
One-colony	0.95	0.98	0.96
Two-colonies	0.84	0.81	0.83
Three-colonies	0.66	0.68	0.67
Four-colonies	0.47	0.53	0.50
Five-colonies	0.50	0.14	0.22
Six-colonies	0.60	0.64	0.62
Outlier	0.83	0.90	0.86

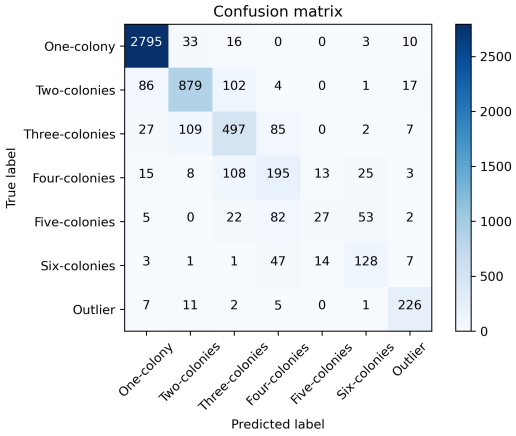


Fig. B.5 Confusion Matrix from MicrobiaS2 validation results.

Table B.2 Classification results evaluated on MicrobiaS3 validation set.

Class Name	Precision	Recall	F1 score
One-colony	0.94	0.98	0.96
Two-colonies	0.82	0.84	0.83
Three-colonies	0.70	0.72	0.71
Four-colonies	0.50	0.46	0.48
Five-colonies	0.33	0.31	0.32
Six-colonies	0.80	0.32	0.46
Outlier	0.85	0.80	0.82

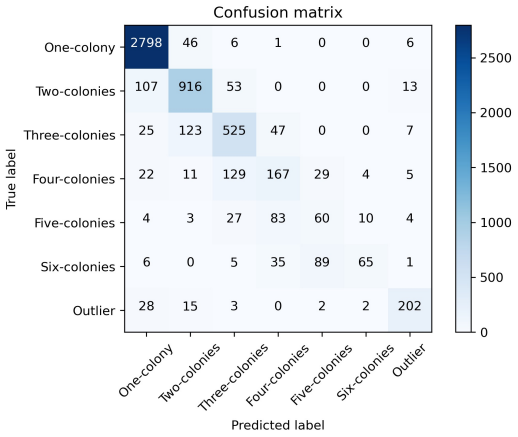


Fig. B.6 Confusion Matrix from MicrobiaS3 validation results.

Table B.3 Classification results evaluated on MicrobiaS4 validation set.

Class Name	Precision	Recall	F1 score
One-colony	0.95	0.97	0.96
Two-colonies	0.83	0.83	0.83
Three-colonies	0.71	0.65	0.68
Four-colonies	0.47	0.54	0.50
Five-colonies	0.37	0.24	0.29
Six-colonies	0.63	0.67	0.65
Outlier	0.85	0.80	0.82

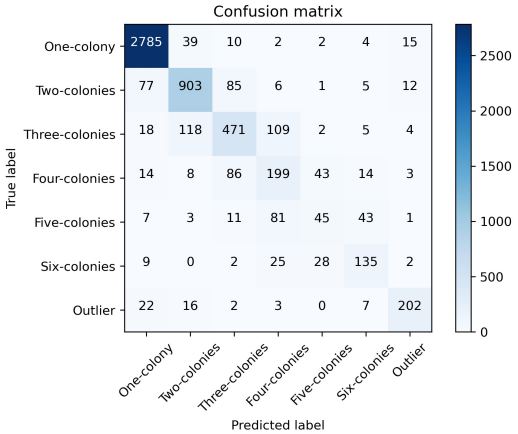


Fig. B.7 Confusion Matrix from MicrobiaS4 validation results.

Table B.4 Classification results evaluated on MicrobiaS5 validation set.

Class Name	Precision	Recall	F1 score
One-colony	0.94	0.97	0.96
Two-colonies	0.80	0.83	0.81
Three-colonies	0.60	0.69	0.64
Four-colonies	0.48	0.36	0.41
Five-colonies	0.36	0.23	0.28
Six-colonies	0.68	0.52	0.59
Outlier	0.87	0.82	0.84

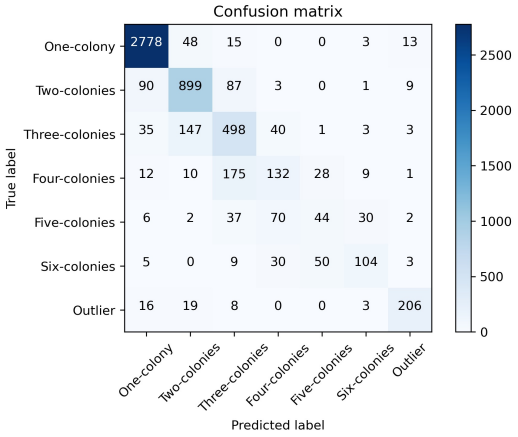
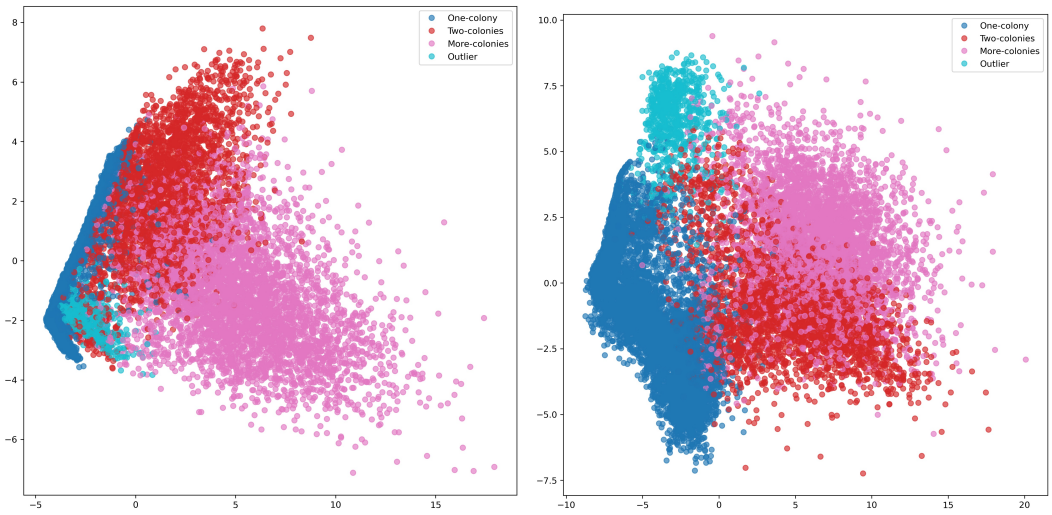


Fig. B.8 Confusion Matrix from MicrobiaS5 validation results.

B.2 Network layer outputs visualisation for the model trained on MicrobiaS1C1 data set

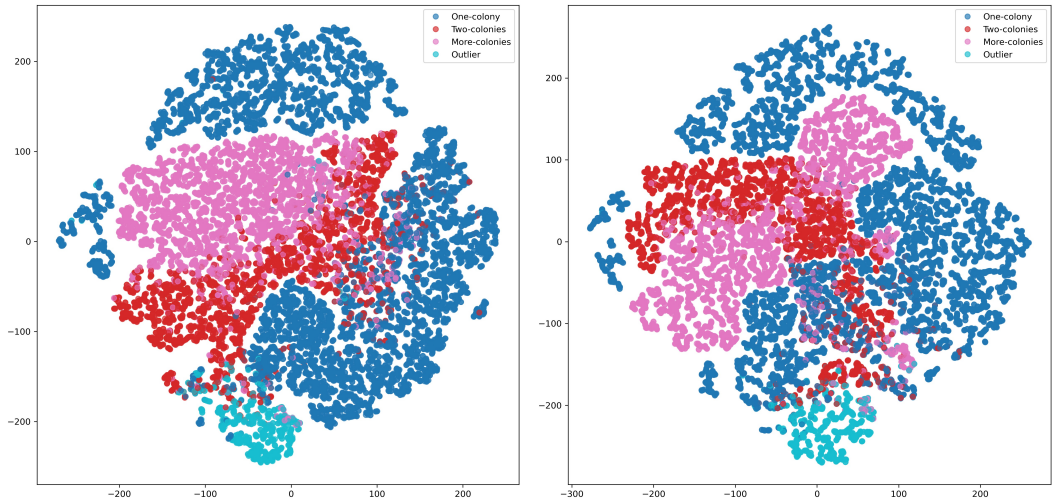


(a) The second to last network layer output. (b) The last network layer output.

Fig. B.9 Visualisation of the last two network layer outputs from the model evaluated on MicrobiaS1C1 training set with dimensionality reduced by PCA.

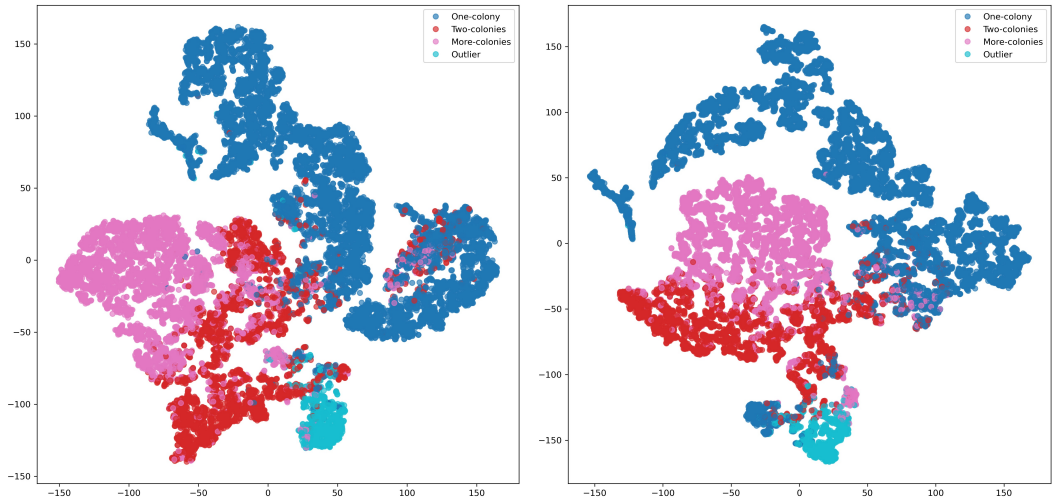
(a) The second to last network layer output. (b) The last network layer output.

Fig. B.10 Visualisation of the last two network layer outputs from the model evaluated on MicrobiaS1C1 training set with dimensionality reduced by t-SNE of 2 perplexity.



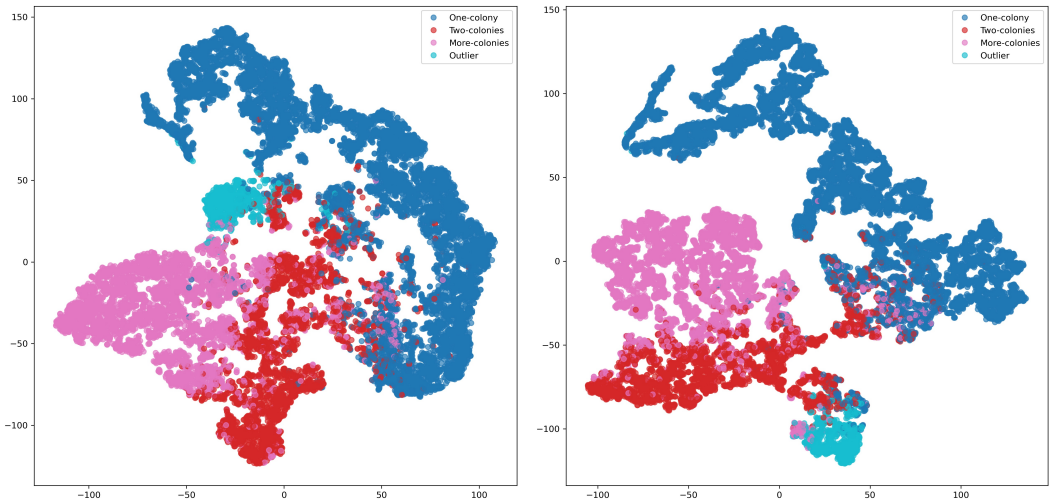
(a) The second to last network layer output. (b) The last network layer output.

Fig. B.11 Visualisation of the last two network layer outputs from the model evaluated on MicrobiaS1C1 training set with dimensionality reduced by t-SNE of 5 perplexity.



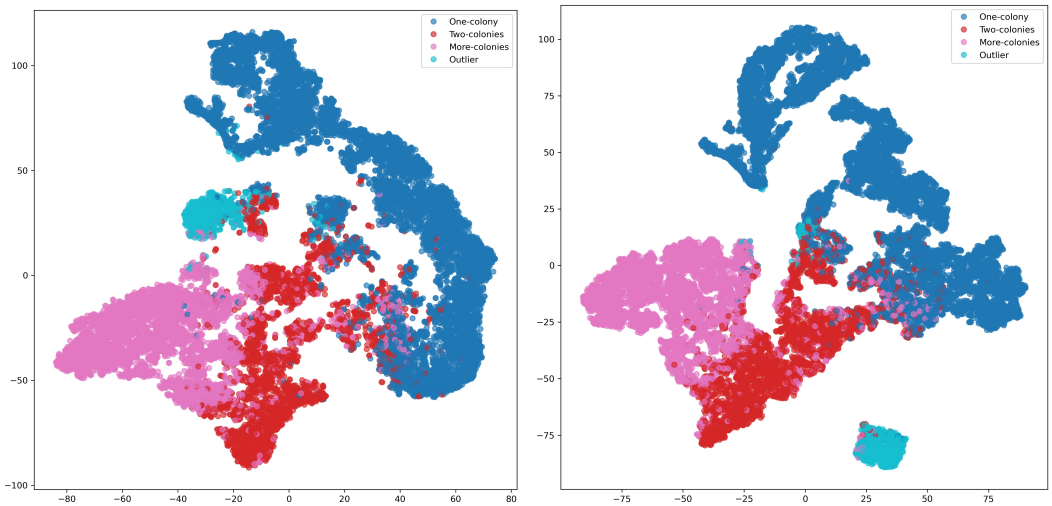
(a) The second to last network layer output. (b) The last network layer output.

Fig. B.12 Visualisation of the last two network layer outputs from the model evaluated on MicrobiaS1C1 training set with dimensionality reduced by t-SNE of 30 perplexity.



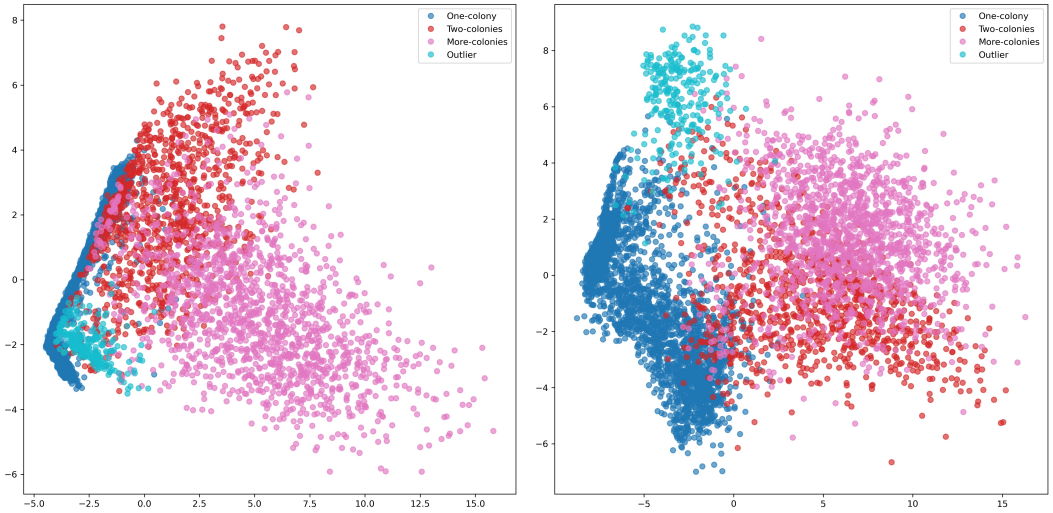
(a) The second to last network layer output. (b) The last network layer output.

Fig. B.13 Visualisation of the last two network layer outputs from the model evaluated on MicrobiaS1C1 training set with dimensionality reduced by t-SNE of 50 perplexity.



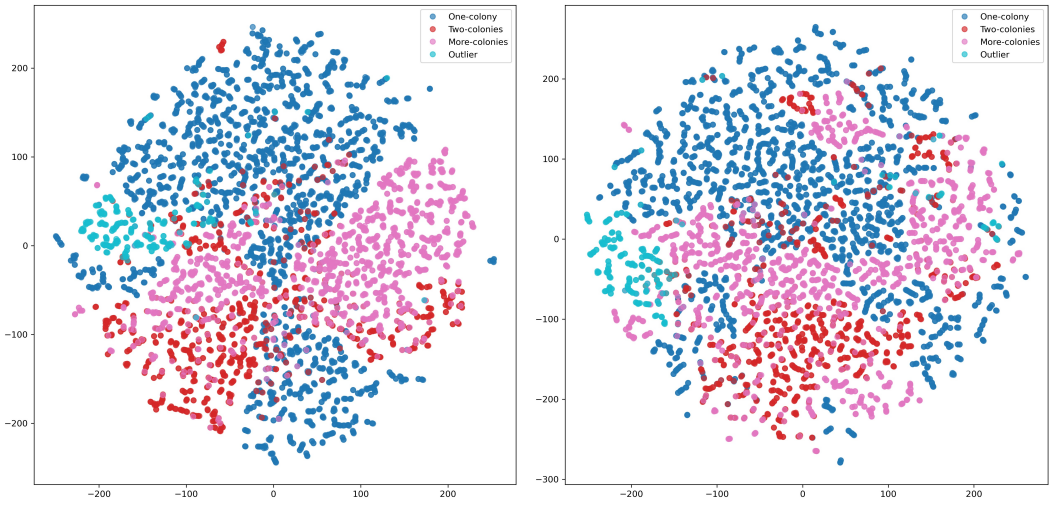
(a) The second to last network layer output. (b) The last network layer output.

Fig. B.14 Visualisation of the last two network layer outputs from the model evaluated on MicrobiaS1C1 training set with dimensionality reduced by t-SNE of 100 perplexity.



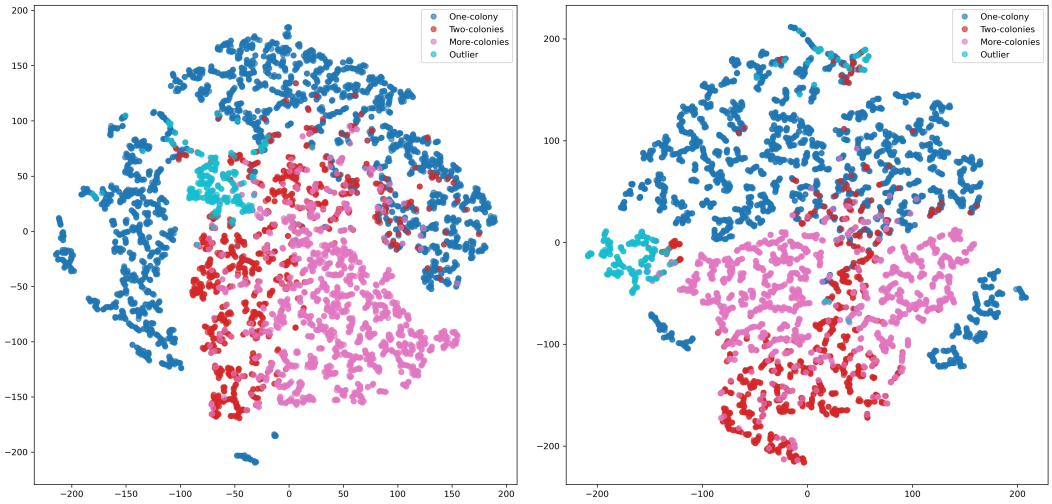
(a) The second to last network layer output. (b) The last network layer output.

Fig. B.15 Visualisation of the last two network layer outputs from the model evaluated on MicrobiaS1C1 validation set with dimensionality reduced by PCA.



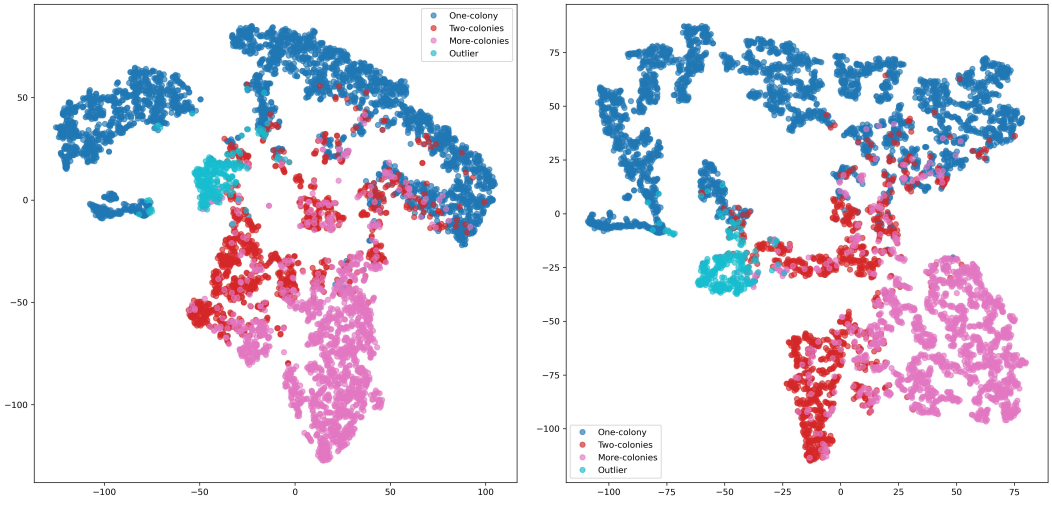
(a) The second to last network layer output. (b) The last network layer output.

Fig. B.16 Visualisation of the last two network layer outputs from the model evaluated on MicrobiaS1C1 validation set with dimensionality reduced by t-SNE of 2 perplexity.



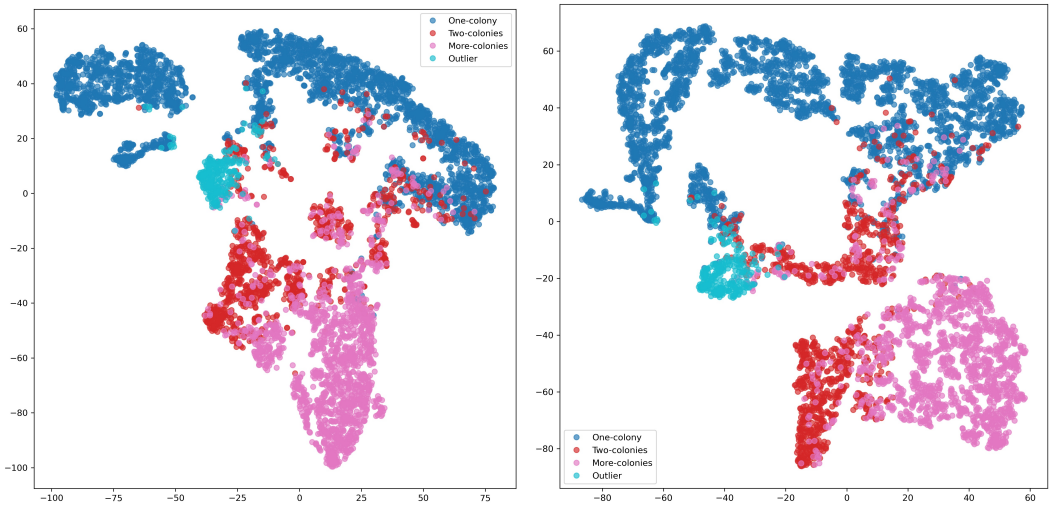
(a) The second to last network layer output. (b) The last network layer output.

Fig. B.17 Visualisation of the last two network layer outputs from the model evaluated on MicrobiaS1C1 validation set with dimensionality reduced by t-SNE of 5 perplexity.



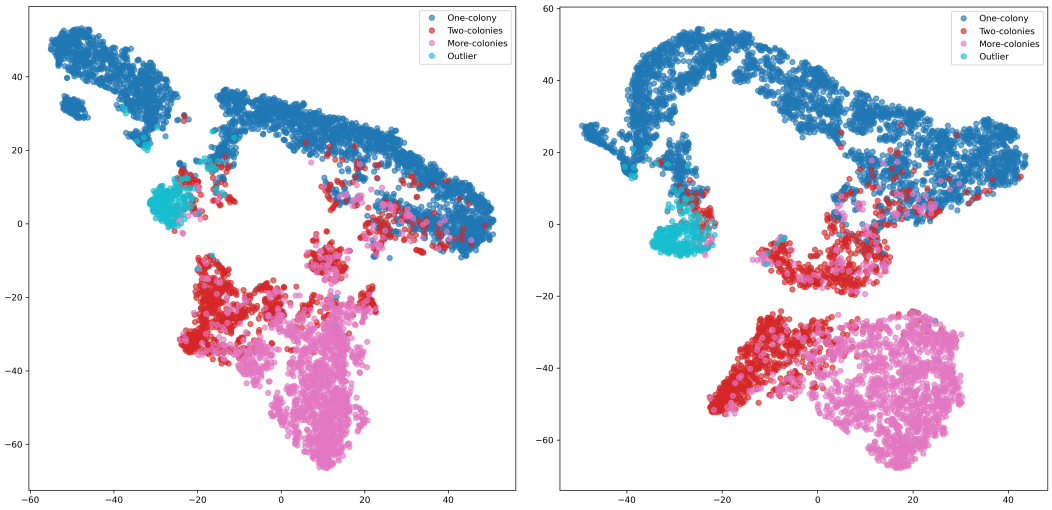
(a) The second to last network layer output. (b) The last network layer output.

Fig. B.18 Visualisation of the last two network layer outputs from the model evaluated on MicrobiaS1C1 validation set with dimensionality reduced by t-SNE of 30 perplexity.



(a) The second to last network layer output. (b) The last network layer output.

Fig. B.19 Visualisation of the last two network layer outputs from the model evaluated on MicrobiaS1C1 validation set with dimensionality reduced by t-SNE of 50 perplexity.



(a) The second to last network layer output. (b) The last network layer output.

Fig. B.20 Visualisation of the last two network layer outputs from the model evaluated on MicrobiaS1C1 validation set with dimensionality reduced by t-SNE of 100 perplexity.

Appendix C

Supplementary material for counting by density estimation

C.1 Results of SAFECount's cross-category prediction

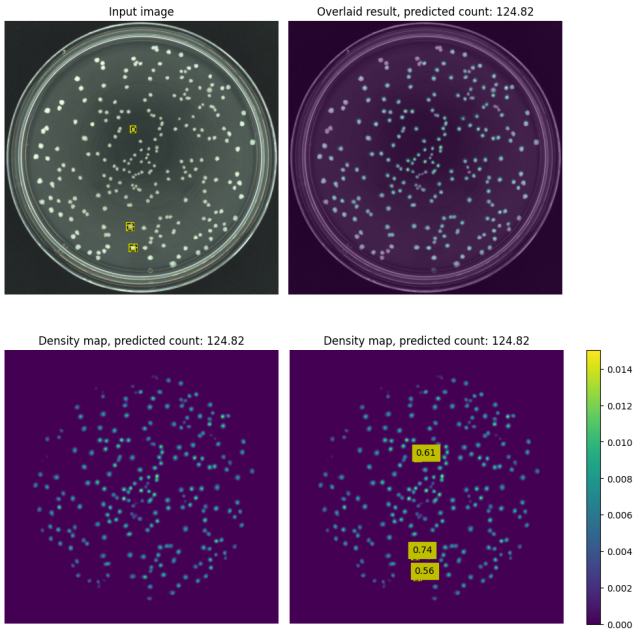


Fig. C.1 Illustration of SAFECount's prediction on Fig. 6.6a. Predicted count and ground truth count are 124.82 and 228 respectively.

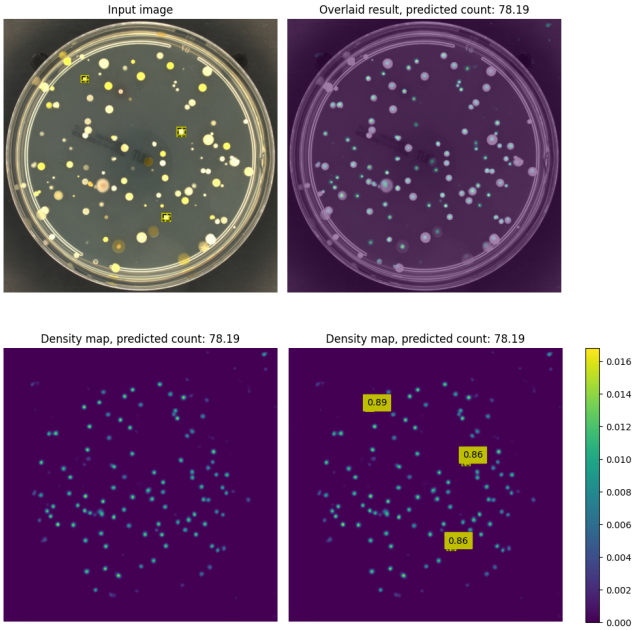


Fig. C.2 Illustration of SAFECount's prediction on Fig. 6.6b. Predicted count and ground truth count are 78.19 and 124 respectively.

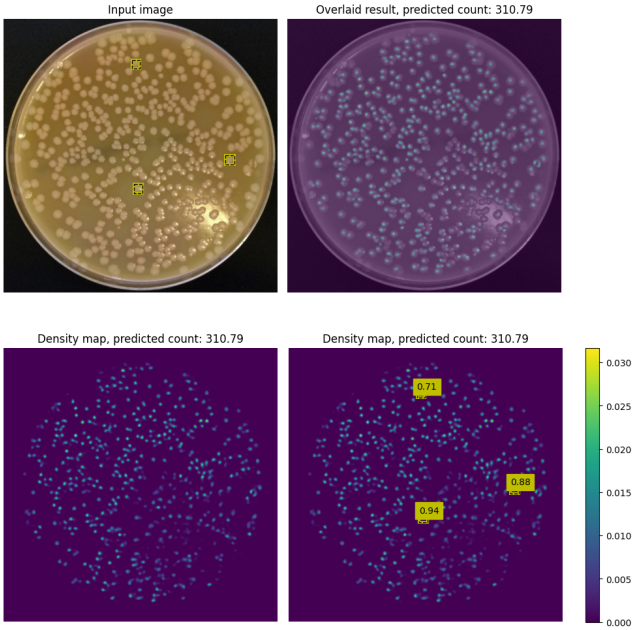


Fig. C.3 Illustration of SAFECount's prediction on Fig. 6.6c. Predicted count and ground truth count are 310.79 and 529 respectively.

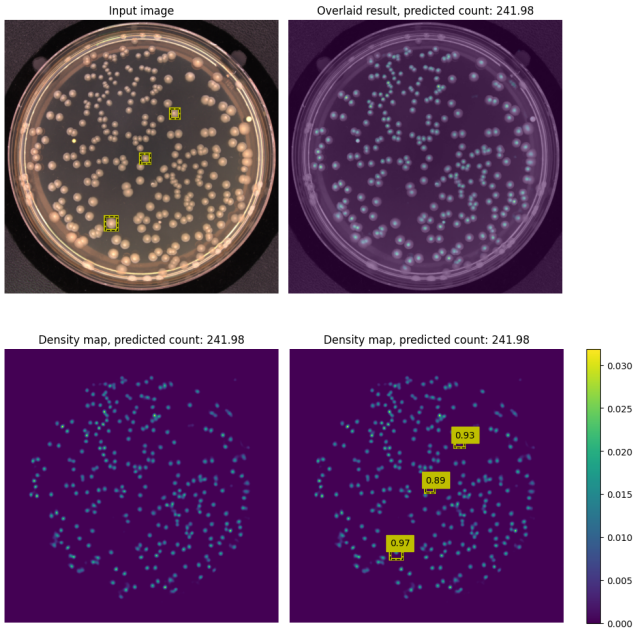


Fig. C.4 Illustration of SAFECount's prediction on Fig. 6.6d. Predicted count and ground truth count are 241.98 and 302 respectively.