Deep Learning for the Identification and Quantification of Wheat Disease

MEGAN CHARLI LONG

A thesis presented for the degree of
Doctor of Philosophy

University of East Anglia, UK
John Innes Centre, UK
September 2022

Abstract

Wheat is hugely important across the globe, providing food and nutrients for millions of people and livestock. Like all crops, it struggles with pressure from multiple diseases, which need to be controlled to preserve both yield and quality. Breeding new varieties with resistance to important diseases is a long process that takes many years and requires trained pathologists to manually score thousands of plots for disease levels. Automating the disease scoring process would free up time for pathologists to work on other important tasks. It also has the potential to improve the accuracy of scoring through multiple applications and eliminating human error.

Here we present a dataset of wheat images taken in real growth situations, including field conditions, with five categories: healthy plants and four foliar diseases, yellow rust, brown rust, powdery mildew and Septoria leaf blotch. This dataset was used to train deep learning models to identify and classify the diseases. We collect a quantification dataset of yellow rust images and performed experiments with different score categories to train various models. Finally, we carry out experiments with simulated data for determining the viability of deep learning models for disease quantification.

In this thesis we find that deep learning models are capable of classifying complex field images, with accuracies of over 97%. We identify limitations in the data collection for quantification of wheat diseases in the field and provide a method for determining ideal dataset size. These results show the viability of deep learning models for quantifying disease and determine some of the challenges which need to be overcome to develop an automated method for use in the field.

Table of Contents

Associated Publications

This thesis contains material from the published works below:

Long, M. (2023). Using machine learning to identify and diagnose crop disease *in* Advances in crop disease detection and decision support systems: Instant Insights Guide (Publication date: Unknown) *and* Advances in sensor technology for sustainable crop production (Publication date: 20th February 2023). Burleigh Dodds Science Publishing

Contribution: Planned and drafted entire manuscript. Completed work for case study.

Long, M. *et al.* (2022). 'Classification of wheat diseases using deep learning networks with field and glasshouse images', *Plant Pathology*, 72(3), pp. 536-547. Available at: https://doi.org/10.1111/ppa.13684

Contribution: Designed experiments, alongside JKB, RJM and MH, carried out all experiments. Performed all analyses and produced all figures. Planned and drafted the manuscript, followed by refining with contributions from all authors

List of Tables

List of Figures

List of Abbreviations

Here is a list of the abbreviations used within this thesis, along with their meanings. This does not include abbreviations contained in references or appendices.

AHDB	Agriculture and Horticulture Development Board
API	Application programming interface
BBSRC	Biotechnology and Biological Sciences Research Council
BLS	Bacterial leaf streak
BYDV	Barley yellow dwarf virus
CNN	Convolutional neural network
CPU	Cental processing unit
DUS	Disctinctiveness, uniformity and stability
GOV	Government
GPU	Graphics processing unit
HPC	High performance computing
iCASE	Industrial cooperative awards in science and engineering
ILSVRC	ImageNet large scale visual recognition challenge
IPP	Inventory photos plus K (app)
JIC	The John Innes Centre
KWS	Seed breeding company (Klein Wanzlebener Saatzucht)
NIAB	National Institute of Agricultural Botany
NPIF	National productivity investment fund
PIL	Python image library
POI	Probability of infection
QOI	Quinone outside inhibitor fungicide
QTL	Quantitative trait loci
RAGT	Seed breeding company
RGB	Red, green and blue
SDHI	Succinate dehydrogenase inhibitor fungicide
UK	United kingdom
UUID	Unique universal identifier

VCU	Value for cultivation and use
VGG	Visual geometry group
VPN	Virtual private network

Acknowledgements

My studentship was supported by a BBSRC-NPIF iCASE studentship with KWS UK Ltd, Limagrain UK Ltd and RAGT Seeds Ltd as industrial partners.

I would like to thank my primary and secondary supervisors, James Brown, and Richard Morris, for their continued support throughout this project. As a physicist with very little biology knowledge, James has been so helpful with his mini lessons and directions to useful information and references to help me learn what I needed. Thank you to Richard for always being a sounding board as I've planned my experiments and interpreted the results. The continued support from both of you as I struggled during the pandemic and with family illness meant so much and helped me get back on track to finish this thesis.

Thank you also to the other member of my supervisory team from the JIC, Matthew Hartley. Matthews technical help in the first couple of years was invaluable in helping me learn about deep learning and plan my experiments.

I would also like to thank my supervisors from the three companies associated with this PhD: Ruth Bryan (RAGT), Nicholas Bird (KWS) and Simon Berry (Limagrain). Thank you especially with all your help collecting images for my datasets. Without your knowledge of when and where to photograph and your help with image collection, I would not have been able to produce any of the results I have. A special thanks also to Paul Fenwick and Rachel Goddard (Limagrain) for getting involved with the project and helping with image collection and the test pathologist experiment.

Thank you to Douglas Brown for being the most hard-working summer student I've ever come across and for following my (often unnecessarily convoluted) instructions to a tee. Thank you also to Patrick Seed for allowing me to visit and photograph his Septoria infected plants at Nottingham University.

Thank you to the members of the Morris Dancers for providing coding help when I needed it and for making the effort to include me in meetings since I've continued to work from home. Thank you also to the members of the Brown and Nicholson groups who, although I didn't spend a great deal of time with them, were always so enthusiastic and welcoming at any meetings or group events.

A very special thank you to my family (Nikki, Carl and Alex Long) and fiancé (Ashley Robinson) for always bigging me up and providing endless encouragement throughout. And finally, thank you to my dog Drax for being my fluffy work buddy!

Megan Long
John Innes Centre, Norwich
September 2022

Chapter 1 Introduction/ Literature Review

1.1 Wheat and its diseases

1.1.1 The importance of wheat

Wheat is a hugely important staple crop, which provides food for humans and livestock worldwide (Shewry, 2009). It is consumed across the globe and cultivated over a huge range from 67°N in Scandinavia and Russia to 45°S in Argentina, including elevated regions in the tropics and subtropics (Feldman, 1995). In 2020 over 760 million tonnes of wheat were produced worldwide, making it the third most produced crop globally (FAOstat https://www.fao.org/faostat/en/#data/QCL). Due to the growing demands of a rising population, the production of wheat and other grain crops has tripled since 1960 and is expected to continue rising (Godfray *et al*., 2010).

For humans, wheat is a major source of starch and energy, as well as providing many other health benefits. It provides protein, vitamins, dietary fibre, and phytochemicals (Shewry and Hey, 2015). In the UK especially, wheat is an extensive source of dietary fibre, which can help reduce the risk of type 2 diabetes, cardio-vascular disease, and some cancers. Due to westernisation of the diet, global demand for wheat is increasing due to its gluten proteins which are used in the production of processed foods (Day *et al*., 2006). Gluten provides valuable viscoelastic functional qualities to dough (Shewry *et al*., 2002), for producing consumables such as bread, noodles and pasta.

Some modern day wheat species are diploid (having two sets of chromosomes), however many are polyploid having four sets of chromosomes (tetraploid) or six sets (hexaploid) (Hancock, 2004). The most commonly produced species of wheat is *Triticum aestivum* (hexaploid), sometimes known as bread wheat, which makes up approximately 95% of the global wheat production. In the UK, this makes up the very great majority of wheat grown. The second most cultivated species is *Triticum*

durum (tetraploid), or durum wheat, used for pasta, which makes up the majority of the remaining wheat production. There are a few other species cultivated in only small areas; einkorn (diploid *T. monococcum var. monococcum*), emmer (tetraploid *T. turgidum var. dicoccum*), and spelt (*T. aestivum var. spelta*) (Shewry and Hey, 2015)

The vast majority of wheat grown in the UK is winter wheat, which is sown early autumn and harvested in the spring and summer. It requires a period of cold, vernalisation, in order to produce seed. The rest of the wheat grown in the UK is spring wheat, which is sown late winter and is harvested late summer.

1.1.2 Wheat diseases

As the population of the world continues to grow, one of the biggest challenges facing food security is crop disease (Strange and Scott, 2005). Wheat, like all crops, can be subject to yield losses thanks to pests and diseases, which have been reported to consume over 20% of the world's wheat crop annually (Savary *et al*., 2019). Disease can have devastating effects on the yield of a crop, in some cases causing major losses where food quality is concerned. Not only is this a problem on a global scale, but it also has effects on a local scale for individual farmers. In poorer areas of the world, farming is the main or only source of income for many families

For any farmer, being able to detect and identify diseases on their plants is hugely important for the mitigation of potential losses. The problem with this, however, is that identifying crop diseases often requires specialist knowledge which is not always readily available to all farmers and can be expensive. Even with specialist knowledge, there are multiple factors that make it even more challenging. Many diseases appear with similar symptoms, meaning they are easily confused with one another. Furthermore, it is not uncommon for multiple diseases to be present at any one time, making the task of distinguishing them even more difficult. For this reason, it is important to know about the ailments, their symptoms and when and how to treat them. In this thesis we focus on foliar diseases (diseases affecting the leaves) for wheat plants. These diseases were chosen because, at the time of planning

the work in this thesis, they were considered to be of particular importance in the UK.

The first disease we will discuss is yellow rust, also called stripe rust, caused by the basidiomycete fungus *Puccinia striiformis* f.sp. *tritici (Pst)* (Liu and Hambleton, 2010). This is an important disease across the UK, especially in the East. It can cause yield losses of 40-50% in susceptible wheat varieties and affect grain quality, however resistant varieties and fungicide sprays can mitigate the losses so they are usually small (AHDB, 2020). Yellow rust rated by AHDB as having high importance for variety recommendation and therefore breeding. Although it's the most important disease in dryer parts of England (the east), it's easier to control with fungicides than some other diseases, e.g., Septoria (see below).

Yellow rust infection can occur whenever there is green leaf material, and the conditions are favourable. In the UK this is usually in the spring, when temperatures are in the range of 2-15°C and humidity levels are high, however some strains are capable of surviving in higher temperatures, up to 23°C. When the temperature is lower, below 2°C,t the fungus lies dormant as mycelium until favourable conditions return and spores can be produced. Mild winters can mean that the epidemic starts much earlier in the season, thus lasting much longer. Spores are spread by contact between the leaves or by the wind.

The earliest yellow rust infections in the season appear as randomly scattered yellow/ orange uredinial pustules on younger leaves. These pustules contain asexual urediniospores, which are produced on wheat and infect wheat, thus generating annual epidemics. As the leaves age, the pustules form stripes, giving way to the most recognisable symptoms of the disease. yellow rust gets to the later stages of its life cycle, the distinctive orange spores fall from the leaf leaving necrotic lesions with black telia (AHDB, 2020). These black telia are often pulvinate to oblong in shape and range from approximately 0.2 – 0.7mm in length and 0.1mm in width (Chen *et al*., 2014). Figure 1.1 shows yellow rust symptoms on leaves.

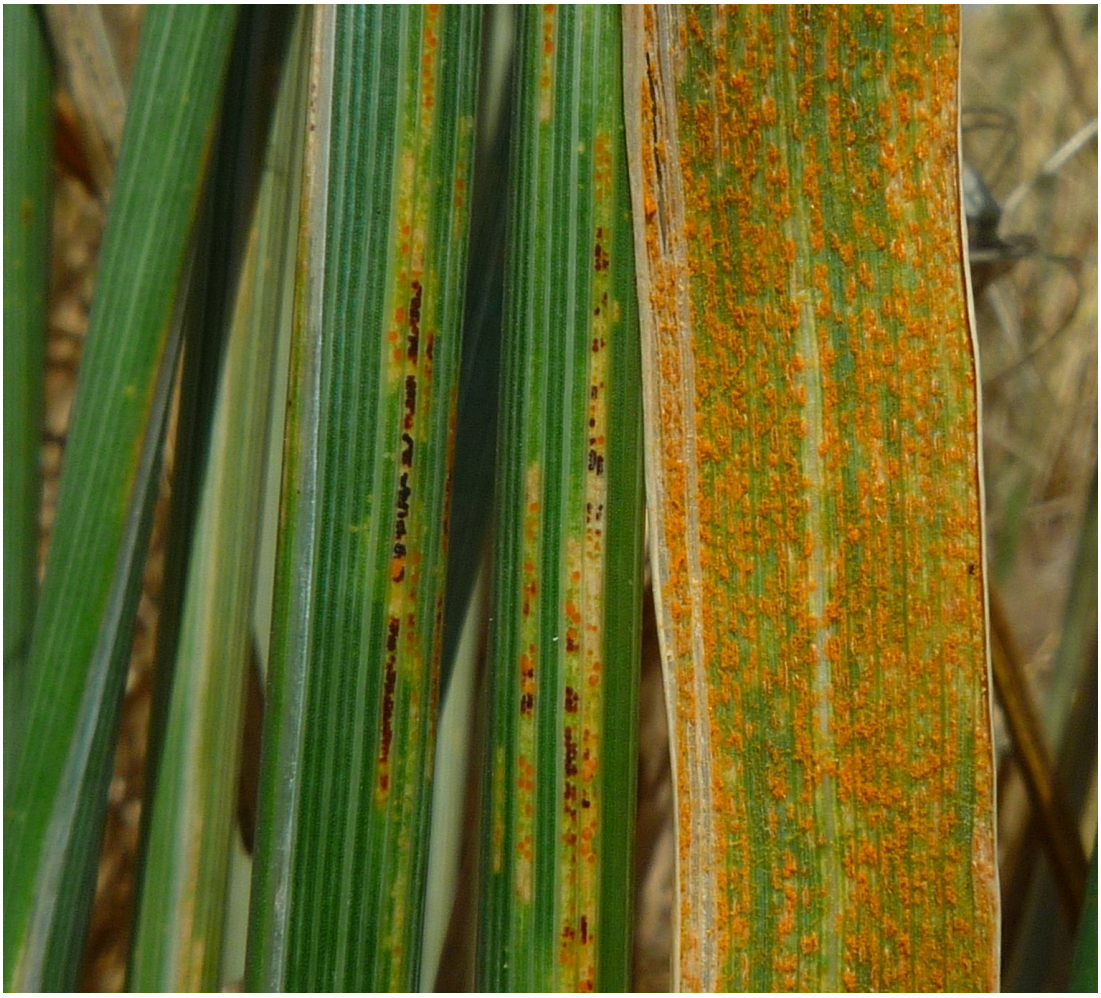


Figure 1.1: An example of yellow rust symptoms including black telia on the left leaves and orange uredinia on the right. From https://www6.versailles-grignon.inrae.fr/bioger/pages-perso/Suffert-Frederic © Frédéric Suffert, INRAE

Telia produce teliospores, the resting stage of rust fungi. Teliospores produce basidiospores, which infect alternate hosts, a deciduous, evergreen shrub called the barberry species, on which sexual reproduction occurs. This happens from Iran to China but sexual reproduction is not yet known to occur in Europe.

The genotypes of *Pst* which have appeared in Europe since 2011 originated from east Asia (Hovmøller *et al*., 2016) and have a much higher capacity for telia production than the only clone of *Pst* which was present in northern Europe until 2010 (Ali *et al*., 2011).

The next disease we include in our work is Septoria tritici blotch, also known as Septoria leaf blotch or simply Septoria, caused by the ascomycete fungus *Zymoseptoria tritici* (formerly *Mycosphaerella graminicola*) (Hardwick et al. 2001). It is thought to be the most important and damaging disease for winter wheat in the UK (AHDB, 2020). It is rated as having very high importance for variety recommendation and thus breeding, partly because of its potential for causing yield losses and partly because of the difficulty of controlling it with fungicide (because *Zymoseptoria tritici* is now largely resistant to azole, QOI and SDHI fungicides). It causes infections across the whole country, with areas where rainfall is higher being most at risk. Much like yellow rust, yield losses can be high (up to 50%) and the quality of the grain affected. Mitigating the losses largely depends on the development of resistant varieties and dryer weather throughout the summer months.

Figure 1.2: Mature Septoria lesions with small black pycnidia. From https://www.apsnet.org/edcenter/disandpath/fungalasco/pdlessons/Pages/Septoria.aspx © American Phytopathology Society

Septoria lesions appear in long oval shapes restricted by the leaf veins, resulting in a rectangular lesion shape. Immature lesions appear as brown necrotic tissue. As they mature, small black fruiting bodies, called pycnidia, form, see Figure 1.2. These pycnidia are smaller than the telia of yellow rust, approximately 0.06 – 0.2mm in diameter, and are almost spherical in shape (Tiley, Foster and Bailey, 2018). Higher levels of infection can mean that lesions join together to form large areas of the leaf that are necrotic, brown tissue.

Septoria lies dormant during the winter, predominantly as pseudothecia (sexual fruiting bodies) and some pycnidia (asexual fruiting bodies). The pseudothecia sexually produce ascospores, which are wind-born, and pycnidiospores are produced asexually by pycnidia in the epidemic phase, which are transferred through rain splash in infected lower leaves. In the spring and summer, when the majority of Septoria infection happens due to optimal temperatures of 15-20°C, pycnidiospores are responsible for the majority of the spread of disease. Following infection, symptoms appear following a latent period of two – four weeks.

Yellow rust and Septoria can cause problems for a pathologist throughout the scoring process due to their similar appearance at certain stages in their life cycle (Brown 2021). Before sporulation, both diseases form lesions with yellow or pale brown areas of necrosis on the infected leaf. In most of Europe, the uredinial stage of yellow rust is formed a few weeks earlier than the pycnidial stage of Septoria. These two stages are clearly visible different, however at the later stages of yellow rust infection when the black telia are formed, it can be difficult to distinguish from mature Septoria lesions that are formed at approximately the same time (Schirrmann *et al.*, 2021). The yellow rust telia are superficially similar to Septoria pycnidia, thus presenting problems even for experienced pathologists when they have limited time to look closely at each plot. This can lead to mistakes when assessing wheat varieties and treatment options.

To further complicate the problem, another important wheat disease, brown rust caused by *Puccinia triticina* (Goyeau et al. 2006; Bolton et al. 2008), produces orange/brown pustules on the leaves of wheat plants. Brown rust is rated as medium

importance nationally, but this is an average rating. It has high importance in south-east England (south of Cambridge and east of Bournemouth) and is unimportant in northern England and Scotland.

In its earliest stages, the brown rust pustules appear in a way similar to early yellow rust infection, with a lighter orange colour, thus leading to the two diseases frequently being confused with one another. Later in the season, as the infection develops, the colour of the brown rust pustules is often darker, and the pustules continue to be randomly dispersed across the leaf, see Figure 1.3. At this stage, it is easier to distinguish the two rust diseases.

Figure 1.3: Brown rust. Brown rust symptoms can often be confused with yellow rust at certain stages of its life cycle. https://ahdb.org.uk/brownrust © AHDB

Brown rust is active at 7-25°C, a wider temperature range than yellow rust and requires surface moisture on leaves for spore germination (AHDB, 2020). Spores are

spread by wind and infection in the UK often occurs mid-late summer, due to the optimum temperature and humidity levels.

The final foliar disease to mention is powdery mildew, which is an important disease in many parts of the world and is caused by *Blumeria graminis* (Dubin and Duveiller, 2011). Mildew mainly appears with distinctive white, fluffy pustules, which are easy to distinguish from the three other diseases we have discussed, Figure 1.4. That being said, no list of wheat diseases would be complete without its inclusion.

Figure 1.4: Powdery mildew infection. https://eldersrural.com.au/news/powdery-mildew-control-in-wheat/ © Elders

Mildew of winter and spring wheat is mostly well controlled by breeding (Brown and Wulff, 2022). Currently rated by AHDB as medium importance but will be upgraded to high importance from 2023, because it is now resistant to most fungicides, therefore it is important for the moderate to high durable resistance found in all current varieties to be maintained. Only one variety with a rating of 3 (on a 1-9 scale, lower numbers represent high susceptibility and high numbers represent high

resistance) has been released in the last 40 years (Leeds in 2017, on which mildew was difficult to control).

Powdery mildew requires living plant tissue to grow. When this is not available, it survives as mycelium in host plants. When conditions are right for germination, the mycelium produce conidia which infect living host plants and are dispersed through wind. Infection can happen over a wide temperature range (5-30°C), with 15°C being the optimal temperature. Rain or other free water inhibits germination, however high humidity is required. Following infection there is a latent period of a week before symptoms are shown. Typical mildew infection begins with the white fluffy pustules mentioned above. Towards the end of the season black spore cases (cleistothecia) develop within the pustules (AHDB 2020).

The four diseases described above are not the only diseases which affect wheat. Here we will mention a selection of other diseases which we have decided not to include in our work:

- Septoria nodorum (*Parastagonospora nodorum*) was important until about 1990 but has now been almost completely controlled by breeding (Cowger *et al*., 2020).
- *Fusarium* head blight: UK varieties are susceptible but environmental conditions are unusually not conducive to disease development. Hence rated by AHDB as medium importance. However, important to breeding companies for markers in areas of Europe with warm, humid summers, especially central Europe (southern Germany, Switzerland, Austria, Hungary, Romania). Fusarium head blight mainly affects the ears of wheat, rarely causing severe symptoms on leaves.
- Tan spot (*Pyrenophora tritici-repentis* or *Drechslera tritici-repentis*) is common in the UK and is not thought to cause significant yield losses.
- Stem rust (*Puccinia graminis*) occurs sporadically (Saunders, Pretorius and Hovmøller, 2019) and is a serious disease in hotter, dryer areas such as East Africa and Southwest Asia. Recent outbreaks of wheat stem rust in Europe have caused yield losses in Sicily but not in northern Europe.

- Barley yellow dwarf virus (BYDV), transmitted by aphids, was controlled until 2019 by neonicotinoid insecticides, which are now banned (European Commission, no date). It was considered locally unimportant at the time of planning the work in this thesis, so was not included, however its importance has since risen. BYDV causes areas of necrosis at the leaf tip, which can sometimes be confused with Septoria, however it is more important on barley and oats than on wheat.

1.2 Breeding for disease resistance

1.2.1 The breeding process

Resistance to common diseases is a trait that is desirable amongst the farming community. Often this resistance needs to be bred into the crop over a long period of time. Over multiple breeding seasons, varieties can be selected based on their levels of disease resistance. However, this is not the only desirable trait, and as such multiple traits are measured during the breeding process. In the UK, as there is not one standout trait which is important over all others, breeding for wheat is a balancing act between desirable traits such as yield, disease resistance and grain quality. This is in contrast to locations in Europe, for example, where fusarium resistance is a standout trait which all varieties need to strive for, see previous section.

The breeding process for new wheat varieties takes many years, moving from millions of seeds all the way to only one or two varieties which will be produced commercially. The pedigree method is very common for use in wheat breeding, however is very resource and labour intensive (Baenziger, 2016). It allows selection among individual plants and whole families at every inbreeding generation (Rutkoski, Krause and Sorrells, 2022) . This method begins by making many crosses between lines which contain desirable traits, based on knowledge and genomics. Each plant contains thousands of genes, so selecting the plants for crossing is a complex process. A team of individuals will then make the crosses by emasculating

the female plant and fertilising using pollen from the male. From this process, often millions of seeds are collected for planting as the F_2 population in the subsequent year. Each new generation is planted in subsequent years.

The F_2 population is planted in single plant rows, so that individual plants within families can be selected and caried forward to the next generation (F_3). Generations F_3 and F_4 are planted as families in rows. Visual selection takes place for every generation, taking only the lines which appear to be showing the best traits through to the next year. Generations F_5 – F_7 are often planted as both treated and untreated yield plots, where they can be tested for different traits. It takes approximately 5-6 years for the lines to meet the desired level of homozygosity.

By the eighth year, the number of varieties has decreased significantly to approximately 10 lines, which can now be taken forward for national list trials. For a variety to be marketed commercially, they must be added to the national lists (GOV https://www.gov.uk/guidance/national-lists-of-agricultural-and-vegetable-crops). These trials take place over two years at various approved locations. The varieties which are added to the national lists must demonstrate that they pass distinctiveness, uniformity, and stability (DUS) and value for cultivation and use (VCU) testing. DUS tests need to prove that the varieties differ from any other variety already available within the species, that the distinctive characteristics are produced uniformly and that these characteristics are stable (do not change) over subsequent breeding periods. VCU tests need to show that the variety has satisfactory value for cultivation and use (GOV https://www.gov.uk/guidance/vcu-protocols-and-procedures-for-testing-agricultural-crops). Once they have passed these tests, they can be listed on the national lists and sold commercially. Usually only one or two varieties makes from the initial cross make it to this point.

The final end goal for any newly bred variety, however, is inclusion on the recommended lists. The recommended lists are publications which provide potential consumers with information about traits such as the quality, yield and resistance to individual diseases to aide in choosing varieties. The recommended varieties are considered to have the potential to provide a consistent economic benefit to the UK cereals or oilseeds industry (AHDB https://ahdb.org.uk/knowledge-library/using-the-

recommended-lists-for-cereals-and-oilseeds-rl). The varieties are chosen for recommendation based on information and data gathered from many trials.

The pedigree method can take 9 years to get a variety listed on the national lists, and even longer to be selected for the recommended lists. For this reason, there are a couple of methods used by breeders to speed up the process: single seed descent and double haploid production. In single seed descent, a single seed is taken from each plant in the earlier generations and used to grow the next generation. This happens in the glasshouse and multiple generations can be produced in the same year, thus reducing the number of years required to reach the desired level of homozygosity. The downside of this is that it is more expensive than the normal pedigree method.

In double haploid production a haploid plant is created by either intergeneric crosses (with maize or corn) or by another culture. The chromosomes of the haploid plant are doubled, producing a double haploid plant which is completely homozygous. A single seed is then taken to form the next generation. As with single seed descent, this method reduces the time taken to reach the desired level of homozygosity, however it is an even more expensive method.

Often, due to only using single seeds for the first few generations, when either of these methods are used, the same crosses are used with the pedigree method also. This is in case any information is lost throughout the sped-up process.

1.2.2 Scoring disease levels

During various stages of the breeding process, plants need to be scored for the amount of disease present in order to assess the resistance or susceptibility to certain diseases. The lines with high susceptibility are removed from consideration, while those with high resistance are put forward to the next stage. For help making these decisions, the plants are scored using a scale which represents how much of each disease is present.

Infection (%)	Yellow rust	Septoria tritici	Score
0	No infection observed		1
0.1	One stripe per tiller	One lesion per 10 tillers	2
1	Two stripes per leaf	Two small lesions per tiller	3
5	Most tillers infected but some top leaves uninfected	Small lesions beginning to form areas of dead tissue across width of leaf	4
10	All leaves infected but leaves appear green overall	Two lower leaves – large areas of diseased tissue, some covering a third of the leaf	5
25	Leaves appear half infected and half green		6
50	Leaves appear more infected than green		7
75	Very little green leaf tissue left		8
100	Leaves dead (no green tissue left)		9

Figure 1.5: In blue: Guidance for scoring yellow rust and Septoria used for rating in the recommended lists (https://ahdb.org.uk/recommended-lists-disease-ratings © AHDB). In green: The score associated with each level of infection.

A widely used scale used for scoring disease is a scale of 1-9 developed by NIAB. This scale gives a score between 1-9 based on the amount of disease present on a plot, from zero disease present (most resistant) to no green tissue left (most susceptible). Figure 1.5 shows the guidance for scoring yellow rust and Septoria of wheat taken from the AHDB recommended list guidance (AHDB https://ahdb.org.uk/recommended-lists-disease-ratings). For brown rust and mildew the same separations of infection are used for different categories. Most breeding companies give a score of 1 to the most resistant plots (0% infection) and a score of 9 to the most susceptible (100% infection). The recommended lists use the same

score categories to rate their varieties, however they give score 1 to the most susceptible plots and score 9 to the most resistant. In this thesis we follow the method used by breeding companies.

Although the final varieties put forward for the recommended lists are scored into these score categories, over the course of the breeding process some breeding companies use variations of this method for their own plants. In some cases, the categories are expanded to have decimal values, or half values in order to give better differentiation between the levels of disease. Otherwise, come companies give a percentage score, which can then be transformed into a score based on the above figure.

Scoring is a time-consuming process, often made more difficult by the variation in disease symptoms, similarity between different diseases and the presents of multiple, simultaneous infections. Currently scoring needs to be undertaken by an experienced pathologist, which can be expensive, especially with non-local trials where the pathologist is required to travel. Often, in these cases, non-local trials are only scored once per season.

It would be beneficial to produce an automated method to aide scoring for breeders. An automated method would allow pathologists to use their time for other important tasks, while the automated scoring model could be used by any person, perhaps as a mobile application. It could also be taken out multiple times, even in non-local trials, where a person local to the trial could be hired for this purpose. This would be more efficient in managing time and cost. It would also remove the difference in scores by different pathologists, and eliminate errors made due to tiredness etc.

In this thesis we investigate deep learning as a possible tool for automating the scoring process. We begin with identification of wheat diseases in field conditions, before investigating models for quantifying the amount of disease present.

1.3 Deep learning for crop disease detection

In recent years, deep learning models have become a key player in the role of detection and identification of crop diseases. If you are not familiar with machine learning, refer to chapter 2 for an introduction to the concepts described here. Convolutional neural networks (CNN) (LeCun *et al.*, 1999) are a type of deep learning network which have become popular for image classification of plant diseases (Boulent *et al.*, 2019). Many studies utilise pre-defined CNN structures for their work. A few examples that occur over and over again throughout the literature are AlexNet (Krizhevsky *et al.*, 2012), GoogLeNet (Szegedy *et al.*, 2014) and Inception (Szegedy *et al.*, 2016), however there are plenty of others which provide a starting point for almost all of the studies we will discuss. These pre-defined networks are all CNNs with different numbers of layers and additional features to aid with feature extraction. See chapter 2 for description of deep learning components and the training process.

A good place to start here is with the Plant Village dataset (Hughes and Salathe, 2016). This collection of almost 88,000 images taken in controlled conditions was the first openly available dataset of crop disease images. The dataset contains 38 categories, each corresponding to a plant-disease pair. Each image contains a single diseased or healthy leaf taken from the plant and placed on a neutral background and photographed under different lighting conditions. Many studies use the whole or part of this dataset with their work (Mohanty, Hughes and Salathé, 2016; Amara, Bouaziz and Algergawy, 2017; Brahimi, Boukhalfa and Moussaoui, 2017; Ferentinos, 2018; Rangarajan, Purushothaman and Ramesh, 2018; Zhang, Huang and Zhang, 2018; Saleem *et al.*, 2020).

Mohanty *et al.*, (2016) aimed to show the viability of deep learning networks for classification of a range of different diseases. They performed the first deep learning experiments using the Plant Village dataset with two different pre-trained networks: AlexNet and GoogLeNet. Through training these two networks both from scratch and using transfer learning, with a range of image processing techniques and train-test splits (the split of data between training and testing), they returned near perfect

classification accuracy of 99.34% by using a pre-trained GoogLeNet, full colour images and an 80-20 train-test split.

Much like Mohanty et al., Brahimi *et al.,* (2017) also used transfer learning with the two pre-trained networks AlexNet and GoogLeNet. In this study however, rather than using the entire Plant Village dataset, a subset of images containing only diseased tomato leaves was used. Both studies utilized the networks by transfer learning and by training from scratch in an attempt to compare the results from both methods. In the same way as in the work of Mohanty et al., the best results gained in Brahimi et al.'s work were of extremely high accuracy, reaching 99.18% accuracy in classifying tomato diseases. Again, this result came from the use of GoogLeNet with pre-training, although they do not specify the train-test split.

Another study that made use of a subset of the Plant Village dataset is that of Amara *et al.,* (2017). They used only the banana leaf images in their work with the LeNet (Lecun *et al.*, 1998) architecture. Although using a previously defined network, they did not use a pre-trained version, rather the architecture was trained from scratch with the banana leaf images. They used a range of train-test splits with both coloured and grayscale images. It was shown that the networks that used coloured images always outperform those without, thus showing the importance of colour information for the problem. Using a train-test split of 80-20, the network achieved an accuracy of 98.61%, another extremely promising result.

Too *et al.,* (2019) took the whole of the Plant Village dataset and evaluated the performance of multiple pre-trained networks in classifying the diseases. They used transfer learning with some fine-tuning of VGG16, Inception V4, Resnet with 50, 101 and 152 layers and DenseNets (Huang *et al.*, 2018). DenseNets was the best performer having gained an almost perfect accuracy of 99.75%.

This almost perfect accuracy is a common occurrence in studies which use only images from the Plant Village dataset. Although comprehensive in that it covers a wide range of diseases and plant species, the images within are not representative of those which would be found in real growth situations. They contain images of leaves taken from the plant and placed on a plain background, thus eliminating any

background information, which obviously would not be the case in the field. The high accuracies gained in these studies are impressive, however it is unknown how any of the models would perform when confronted with real field data.

Ferentinos, (2018) demonstrated the issues of the Plant Village dataset for field use in their work. They made use of multiple pre-trained networks within his study; AlexNet, AlexNetOWTBn (Krizhevsky, 2014), GoogLeNet, Overfeat (Sermanet *et al.*, 2013) and VGGNet. The dataset used contains images taken from Plant Village as 'lab condition' images and was supplemented with more images taken in the field. This resulted in a dataset of 87,848 images sorted into 58 classes, some that contained just lab conditions, others that contained just field conditions and some with both. The most successful architecture in this study was the VGG network, which gained an accuracy of 99.53% on unseen images. Due to the presence of both lab condition and field condition images within the dataset used, Ferentinos (2018) experimented with training on laboratory condition images and testing on field condition images and vice versa. The accuracy of classification in these experiments was significantly lower than with the mixture of images for training. Training on field images and testing on laboratory images resulted in an accuracy of 65.69%, whereas the other way around resulted in an accuracy of only 33.27%. These figures emphasize the importance of including all relevant conditions within a training set for use in practice.

Although the Plant Village dataset is used regularly throughout the literature, there are plenty of studies which make use of data acquired elsewhere. Sladojevic *et al.*, (2016) created a large dataset of images (over 30,000 in 15 classes) by taking pictures from internet searches. The dataset included a class for just healthy leaves and also a class with just background images. The reason for this was to train their network to differentiate leaves from their surroundings. The network used for this study was the pre-trained CaffeNet (Jia *et al.*, 2014) model. Using this method, they gained a classification accuracy on their dataset of 96.3%. They concluded that the accuracy for individual categories was slightly lower on the classes which contained fewer images. Another thing to note about this study is how the images were collected. As they were taken straight from the internet, it is possible that some of the images have been wrongly classified which would have affected the accuracy of the network.

Barbedo, (2018) used a dataset of images collected in both field and controlled conditions. The size of the dataset was relatively small, only 1383 images in total, spread over 56 disease-crop categories. This meant that there were only a small number of samples for each class. They performed experiments by training a model to classify the images both as is, and with the background removed. The results varied over all the categories, with no distinctive positives or negatives recorded for removing the background information. This was probably because the low number of training images was not enough for the model to make accurate representations from the data, whether the background was included or not.

A study by Lu *et al.*, (2017) used a relatively small dataset of rice disease images (500 images) to train CNNs inspired by LeNet and AlexNet architectures. Although they did not use the actual networks for either training from scratch or transfer learning, they did create a very similar network to those already defined. The accuracy gained for this network was 95%; while still a very encouraging result, this is slightly lower than many of the results discussed before. A reason for this could be to do with the size of the dataset used; with only 500 images spanning 10 categories, it could be hard for the network to learn all the characteristics present in each of the categories.

Alongside their own network modelled on a combination of AlexNet and GoogLeNet, Liu *et al.*, (2018) utilized four pre-trained networks on their apple leaf disease identification problem; AlexNet, GoogLeNet, VGGNet and ResNet .They compared their network results to those obtained through transfer learning with the pre-trained networks and found that their model outperforms the known networks. The final accuracy recorded for their network was 97.62%, a percentage point higher than the next best performer VGGNet. Many studies make use of pre-defined networks; however, Liu *et al.* show that in some cases, defining a new network will gain a better performance. Often new networks will be inspired by one or several of the widely known networks (like in Lu *et al.*, (2017)), but this might be the best way to get all the best components for tackling the problem.

The train-test split is important for ensuring a network has enough data to learn from, while also having enough to for evaluating its performance. It is also important to include validation where possible. Often the validation is incorporated into the train part of the split when described in the literature.

Oppenheim *et al.*, (2018) experimented with different train-test splits to find the best combination for their work detecting potato tuber disease. Their dataset contained 2465 images of disease lesions cropped from whole potato images, with four diseased and one uninfected category. They found that, unsurprisingly, more training data increased accuracy. The model that performed best on the test data used a 90-10 train-test split and gained an accuracy of 95.8%. Many studies elect to stick to an 80-20 split in the training and test data, in this case the higher amount of training images may improve training, but the lower amount of test images may not have contained enough images to fully show the performance of the network considering the size of the original dataset. A 90-10 split may be more suited to a larger dataset where the test set would contain more images. This would not be an issue, of course, with a significantly large dataset containing, for example, 100,000's of images, where each subset would contain ample data.

At the time the classification work in this thesis was carried out, there was little to no research which utilised a large, multi-category dataset of field images for any disease. In recent years, the Plant Village dataset is still a popular tool for testing new deep learning architectures and training methods (Kulkarni *et al.*, 2021; Albattah *et al.*, 2022; Pandian *et al.*, 2022). We have encountered one study by Haque *et al.*, (2022) where a dataset collection effort similar to our own was conducted. In this study, they collect images in field of maize plants with three diseases and a healthy category. Data augmentation methods are used to combat imbalance between the number of images per category.

The studies discussed above have shown the great potential for deep learning to be used for crop disease detection. The Plant Village dataset was a breakthrough in the field, which has seen multiple networks classify its images with incredibly high accuracy. Furthermore, other works have used more complex images while still gaining promising results. The problem with these works is that they are not likely to

be viable for use in the field. Due to the time investment often required for collecting field data, many of the datasets used are relatively small. This means that it is unlikely that the range of variable conditions which would occur in the field are represented within the training data. Moreover, in some cases only two classes are used, one disease and healthy. Here, when confronted with a disease which wasn't included, a model would likely classify it as the disease it was trained on, which could cause problems with applying the wrong control measures. There is a lot of room for expanding these techniques for use with more comprehensive field datasets containing more diseases and crop types.

1.4 Deep learning for crop disease severity assessment

When compared with the research for using deep learning for crop disease detection, little research has been conducted for determining the severity of crop disease using deep learning. Much of the research into this problem has been conducted since the planning of our quantification experiments took place in early 2020, so plans were guided mainly by our own experience and knowledge.

The studies for this section can be split into two main groups. The first involves the collection of data which is then labelled with a severity score or class. These images are then used to train a deep learning model to classify them into the pre-defined scores. One of the earliest studies for severity assessment was by Wang *et al*., (2017), who took apple black leaf rot images from the Plant Village dataset and had botanists assign a class to each; : healthy stage, early stage, middle stage, or end stage. They end up with just over 100 images per category, which their model is able to classify with an accuracy of 90.4%. This is a good starting point, although they highlight the need for collecting more data, across more severity categories in more versatile conditions.

In a different approach, Mi *et al.*, (2020) collected 5242 images of wheat leaf images in the field, with various levels of stripe rust infection. The images were divided into 6 levels of infection. In this study, prior to network training the images were cropped

to form a rectangle around the leaf, so little background information remained. The result was a fairly well-balanced dataset, with over 600 images per score category. After training, their model was able to classify images from their test set with 97.99% accuracy. This study provided encouraging results, however it would be ideal for a model to not require cropping images prior to usage. If this model were deployed in the field, there would be a significant time investment to get the data into the right format for use.

The second group of studies require more data preparation. Images are annotated in a process called semantic segmentation, where each pixel is assigned a class label. In the case of crop disease severity, these classes could be healthy tissue, disease tissue and background information. A model then learns to perform this segmentation themselves, and from there calculate a percentage of infection.

An earlier example of this method was by Lin *et al*., (2019). In their work with cucumber powdery mildew, they collect a small dataset of 50 leaf images taken in controlled conditions. Every image is manually annotated with the segmentation of the disease lesions. 30 of the images are augmented and used to make the train dataset. They determine that their model achieves a "satisfactory" segmentation accuracy when tested on the other 20 images and conclude that their method is feasible in practice. The main limitation with this study is the dataset size. Although the data is augmented to produce more training samples, there are only 30 initial samples to augment, meaning that there will be little natural variation in the images.

Chen *et al*., (2021) collect a dataset of rice bacterial leaf streak (BLS) images, taken in the field. They use LabelMe software (Russell, Torralba and Murphy, 2008) to assign a class of BLS lesion, rice leaf or background to each pixel in the images. The percentages of infection were sorted into five disease score categories. Their model was able to accurately segment the disease lesions and classify the images into the corresponding score category with at least 89% over all classes. It is important to note that in this study, despite using realistic field data, the data is all collected on a single day in a single location. Therefore, the range of conditions and amount of data needs to be built upon for deployment in the field.

In a similar experiment, Divyanth *et al*., (2023) used a segmentation model to identify and then calculate the severity of three corn diseases. Through their segmentation methods and model training, they were able to identify and quantify the severity of the diseases to a high accuracy. The inclusion of multiple diseases is a useful tool for working towards an automated method. In many cases, multiple infections can be present at once. It would be useful to build on this work with a larger dataset for training.
Both types of study discussed in this section show the early stages of this problem. They show the promise of using deep learning for quantifying the amount of disease present using two different methods. The first, classifying labelled images into severity classes, shows promise using carefully curated or cropped images. It would be useful to be able to use images that are not taken in controlled conditions or require editing by the user before use.

The second method using semantic segmentation also produces good results with small datasets of images. In each case, the segmentation process performs with relatively good accuracy, meaning they can approximate the amount of disease present fairll well. The accuracy of these models could likely be increased with further training data.

All the studies discussed in this section use single leaf images, whether in controlled or field conditions, to train their models. While this is a promising start, a breeder will not be scoring disease on a single plant very often. During the breeding process, they are required to score full plots. To be able to automate this process, any model will need to be trained using full plot images.

1.5 Thesis overview

In this thesis we will investigate the potential for deep learning models for wheat disease identification and scoring given complex images taken in realistic growth conditions. This will provide a foundation for eventually producing an automated

method to aid breeders and farmers in tasks which usually require specialist knowledge to achieve.

In chapter two we give an introduction to deep learning. Here we provide descriptions of the individual aspects of the deep learning models used in our work as well as the process of training a deep learning model as a whole. We also outline the challenges posed by data collection and the data requirements for using deep learning to classify and quantify crop diseases.
In our third chapter we produce deep learning models for the classification of wheat diseases. We collect a dataset of diseased wheat leaf images taken in complex, realistic growth conditions, which is used to train a deep learning model to classify four diseases (and a healthy category). We show that deep learning models are capable of handling complex images and can classify them with high accuracies. The performance of our model is compared to human participants, revealing it can perform at least as well as pathologists on image data. We perform an experiment to verify that the correct information is being used by the model to drive classifications.

Breeders have identified a need for automating the scoring process to save time and remove differentiation between different scorers. The fourth chapter explores deep learning models for quantifying the amount of disease present. A dataset of scored yellow rust plots is collected and used to train deep learning models. We highlight limitations in the collection of field data for this problem. We perform experiments with simulated data which show that classification based on infection level is possible given sufficient data. This work provides the means for evaluating the best experimental design for a disease quantification problem.

In our final chapter we discuss the results of our work, along with the limitations we discovered and possible ways these could be overcome. The potential next steps for taking this work closer to an automated disease scoring system are also outlined.

Chapter 2 Introduction to Deep Learning

The use of deep learning is a constant theme throughout this thesis. For this reason, we provide an introduction, within this methods chapter, to the various elements of deep learning for crop disease detection and quantification that we have used.

The two main libraries we use for creating and training our deep learning models are Keras (Chollet and others, 2015) and Tensorflow (Abadi *et al*., 2015). Keras is an interface (API) for Tensorflow, allowing for a quick and easy implementation of a deep learning model that relieves the user from many of the complexities of Tensorflow. A deep learning model in Keras can be written in a few, relatively simple, lines of code and is much more user-friendly than Tensorflow. Tensorflow is a comprehensive, open-source library for machine learning. Where Keras is useful for learning and understanding deep learning models, and quick experimentation with model architectures, Tensorflow has a wider range of capabilities for real world applications.

2.1 Machine Learning

To understand deep learning and its purpose in this project, it would first be wise to introduce the entire topic of machine learning, of which deep learning is a type. Machine learning is a field that focuses on using algorithms that 'learn' to perform a task. The following section will describe some of the aspects of a machine learning problem, which are important for the work in this thesis.

2.1.1 What is classification?

In machine learning, classification is an algorithm which predicts a class label associated with the input data. For example, in this thesis we are using a model to

predict the disease present in an image. A classification model is trained using labelled data, in our case images labelled with a disease class,

2.1.2 How to assess quality

As with any scientific problem, it is important to be able to assess the quality of the results. For classification, the main metrics for this are accuracy, precision, and recall, the latter two often being combined to give an F1 score.

Accuracy is defined as the percentage of correct predictions out of all predictions made. In a perfect scenario, where the model had learned enough to correctly predict every classification, the accuracy would be 100%, however in practice, a 100% accuracy is rarely (if ever) achievable. The accuracy which would be deemed acceptable often depends on the particular problem at hand and the accuracy of a human completing the same task. In some cases, a lower accuracy may be accepted where the model is able to compete in another manner, for example speed or man hours. Accuracy is the main metric we look at in this thesis due to the replicable nature of the work.

In some classification problems, in addition to accuracy, it is useful to know the measure of false positive and false negatives for the model. This would be of particular importance for example in a medical field false classifications in disease diagnoses could have critical repercussions. This is where precision and recall come in.

Precision is a measure of how many of the classification predictions for a certain class were correct. So, in a medical sense, the number of positive diagnoses that were actually positive cases. Obviously, it wouldn't be ideal to diagnose a person as having a condition when they don't actually have it, as this would result in emotional trauma and unnecessary treatments and tests. In this work, the precision would be the number of predictions for each class that were correct for that class. Falsely predicting the wrong disease could result in the wrong fungicides being deployed, so

the precision for each class needs to be high. Precision is calculated by the number of true positives over the sum of true and false positives.

Recall is a measure of how many actual pieces of data from a certain class were correctly identified. In this work, that would be the number of correctly classified images for one class out of all the images of that class. Although the recall should ideally be high, it is not quite as important in this case due to the replicability of our model. In a field there would be 1000's of plants allowing for several classifications to be performed to build a full picture of the disease situation in the field. In a medical field it is more important, missing a positive diagnosis would mean that vital treatment would not be given. Recall is calculated by the number of correctly identified positives out of all positive pieces of data.

In machine learning, an F1 score is often used as a measure of quality, combining the precision and recall over the whole model for all categories. We used the macro averaged F1 scores for our work, which is the unweighted mean of F1 scores for each class. The F1 score for each individual category was calculated using Equation 1, then an average F1 score was calculated for the entire model.

Equation 1

$$F1 = 2 \times \frac{(precision \times recall)}{(precision + recall)}$$

2.1.3 What is overfitting?

A common issue in machine learning problems, and one that needs to be mitigated during training, is the problem of overfitting. Overfitting occurs when a machine learning model learns specifics about the training data and so is not able to perform accurately on new data. This means that the model would be able to make extremely accurate predictions on the training data, but essentially be useless in the field. When working correctly, a model will learn general features about the training data which it is able to apply to new data it has never seen before.

There are multiple ways to try and combat overfitting when training a model, the first and most obvious being the use of validation data.

2.1.4 What is validation?

Validation is the process of evaluating the performance of a machine learning model. A validation set of a data is used during the training of a machine learning network to help ensure that the model does not overfit to the training data.

In this thesis we use a hold-out validation set, which is separate to the test set. After each iteration through the training data, the model makes predictions on the validation data to see how it performs on new information. If the model is learning too much about the training samples, it will not perform well on the validation set as it will not be able to generalise to this data. This information allows us to see when the model overfits and use this to determine how many iterations to train the final model for prior to evaluating on the test set (completely new, unseen data).

2.2 Deep Learning Network Components

In this section we will introduce the different elements of the deep learning models that we create throughout this thesis. The descriptions contained within this section are sufficient for the reader to have a good understanding of the work in this thesis, however they are not fully comprehensive. More information can be found in the papers cited as well as various books including (but not limited to):

- Deep Learning with Python – Francois Chollet
- Deep Learning – Aaron Courville, Ian Goodfellow, Yoshua Bengio

2.2.1 Layers

Before jumping into the individual components that make up our deep learning models, it would first be beneficial to describe the overall topology of a neural network.

A neural network is a type of machine learning or deep learning model which learns to perform a task in a way that is inspired by the human brain. It is made up of interconnected neurons (see section 2.2.2) which are arranged in a layer formation. Every network begins with an input layer. This is the layer which takes the raw data into the model (for example image data). At the end of each network there is an output layer. This layer provides predictions about the data, for example classifications.

In between the input and output layer there are hidden layers. Hidden layers perform computations to transform the input data into something the output layer can use to make its predictions. The number of layers can be anything from zero upwards depending on the experimental design. For a simple problem with little available data for training, fewer hidden layers may be used, however for more complex problems with lots of data there may be more. This can be experimented with during the training process.

2.2.2 Neurons

A neuron takes input signals, either from neurons in the previous layer or from input data, performs a function and then sends output signals to the neurons in the following layer. Each input signal has an associated weight, which is a learnable parameter which is adjusted throughout training. The weight dictates the importance of the preceding signal on the entire network. At the beginning of training, the values

for the weights are randomly initialized. Neurons are present in all layers of a deep learning network and are the nodes through which data flows.

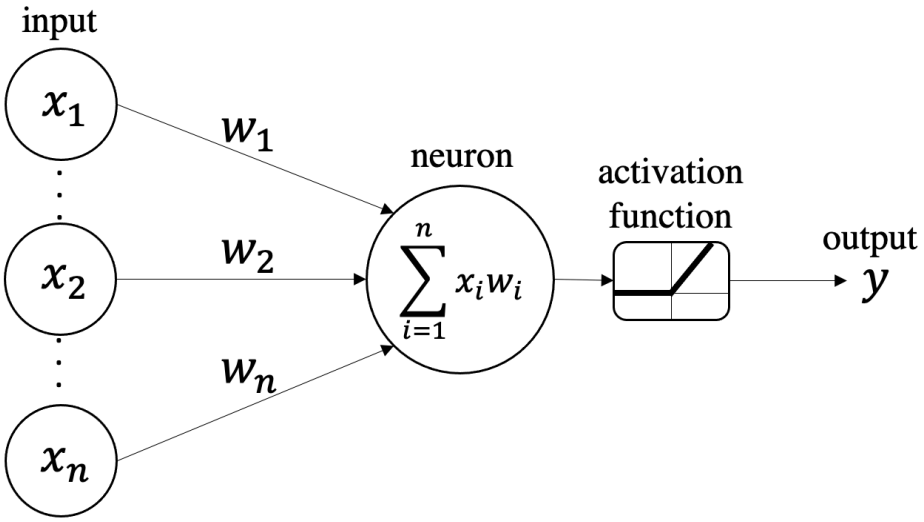


Figure 2.1: A neuron receives an input signals and weights. The signals are multiplied by their weights and summed before being passed to an activation function to produce an output signal.

Once the neuron has received the input signals and their associated weights from all neurons, n, in the preceding layer, the signals, x_i, are multiplied by their weights, w_i, and summed before being passed to an activation function. The activation function calculates the output of the neuron, see Figure 2.1. The activation function is chosen for each layer and is applied to all the neurons in that layer.

In the models we create in this thesis, we use two different activation functions. For the input layer and all layers before the output, a rectified linear unit (ReLU) function is used, see Figure 2.2, which sets all negative values to 0. This activation function is commonly used in neural networks due to their computational simplicity, meaning faster training, and linear behaviour for positive values, which allows for easier optimisation of the model. Networks which utilise the ReLU function often yield better performance than networks trained with other activation functions (Glorot *et al.*, 2011).

For the output layer which gives the predictions a softmax function is used, see Figure 2.3. It converts the input of the function to a probability distribution over the number of classes, where the values are non-zero and add up to one. Here, a higher input value would produce a higher probability.

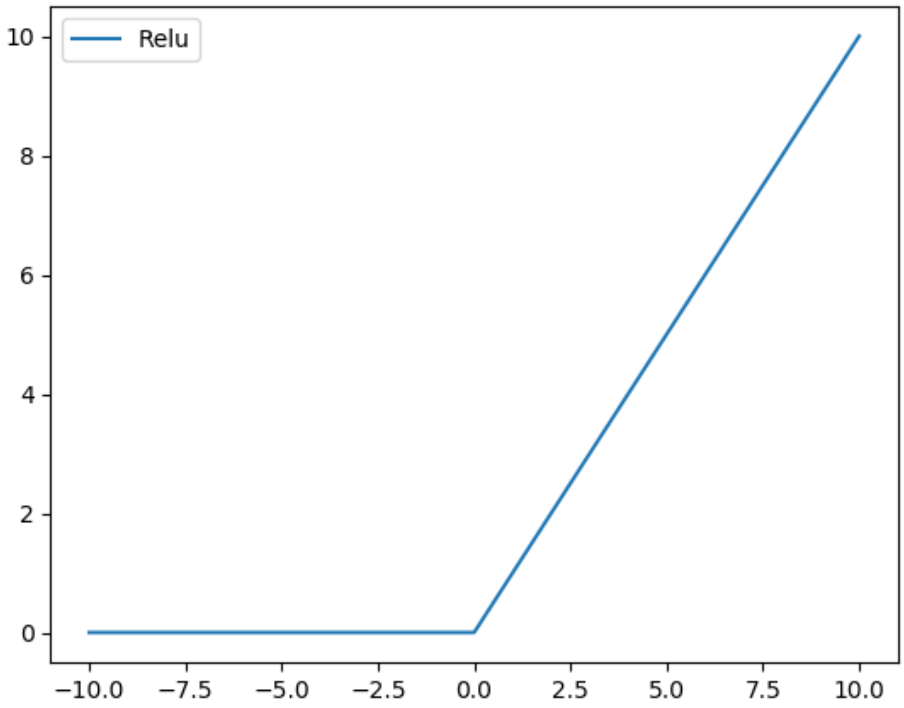


Figure 2.2: Rectified linear unit (ReLU) function. Sets all negative inputs to 0.

Each layer in a deep learning network is made up of multiple neurons, which are connected to a selection of the neurons in the previous and following layers. This selection depends on the type of layer used. There can be any number of neurons in a layer. A small number will mean a small number of learnable weights, and therefore a limit on the amount the network can learn. Conversely, too many neurons will overcomplicate the problem and increase the computing power required for the calculations. It is best to use the fewest number of neurons possible to complete the task, without having too few. This number depends on several factors such as the complexity of the data, and the types of activation function used, and can be determined by experimenting with different values.

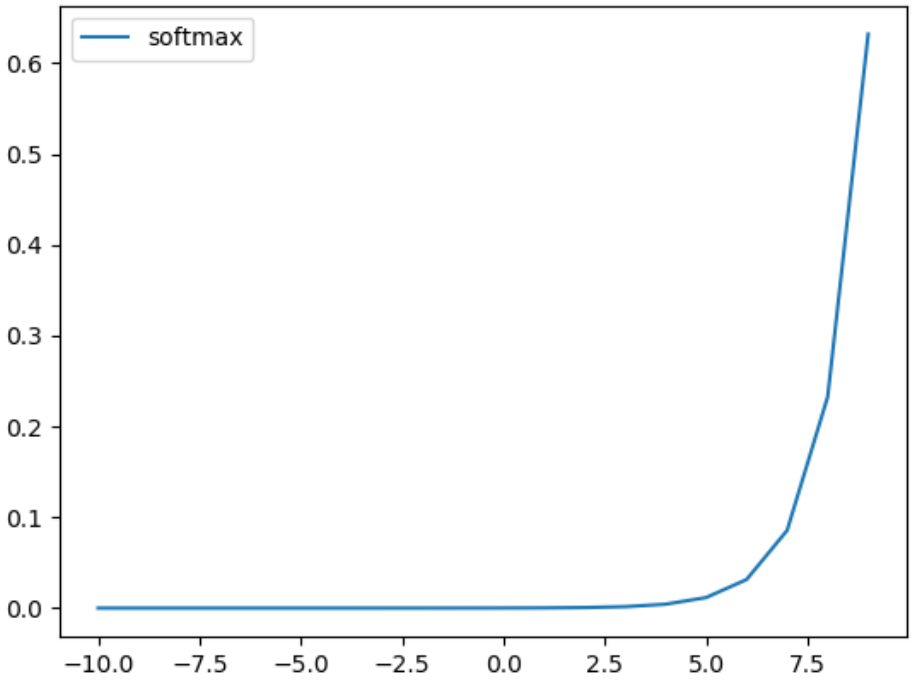


Figure 2.3: Softmax function. A higher input produces a higher probability.

2.2.3 Optimising weights

After each run through the input data (epoch), the output is predictions for the classifications of the data. A loss function is used to measure how far these predictions are from the expected output. It takes the predicted results and computes a distance score for how far away they are from the true results. The loss is the result which shows how well the network performed on the previous data. Throughout training, the network needs to update its weights in order to minimise the loss.

In this work we use categorical cross-entropy as the loss function. This can be calculated by Equation 2 where S is the number of samples, N is the number of classes, t is the true distribution for a sample of data (this will be a vector zeros of length N, with a 1 at the position corresponding to the correct class) and p is the predicted distribution produced by the output activation function.

Equation 2

$$L(t,p) = -\sum_{i=1}^{S}\sum_{j=1}^{N} t_{ij} log(p_{ij})$$

The nature of neurons in a deep learning network means that information is only passed forwards through the model. To be able to update the weights after each iteration, information somehow needs to be passed back through the model. This process is called back-propagation and is how the network fine-tunes the weights. The model uses information about the loss function and an optimizer to calculate the new values for the weights. An optimizer is a function used to minimise the loss function. There are many different optimizers, three which are commonly used are Adagrad (Lydia and Francis, 2019), RMSProp (Hinton and Tieleman, 2012) and Adam (Kingma and Ba, 2017).

2.2.4 Hyperparameters

In addition to these weights that are optimised through the training process, a number of other factors are important that need to be chosen prior to training – these are hyperparameters. When developing a deep learning model for a specific purpose, it always involves tuning the network hyperparameters to find the configuration which gains the optimal results. Examples of hyperparameters are:

- network architecture – the number, size, and type of layers in the model
- batch size – how many images the model works through before updating its parameters
- learning rate – the magnitude of change to the model weights during training
- number of training epochs – how many times the model works through every piece of training data
- optimizer – used to update the weights to reduce the loss function. The loss function is how the network measures its performance. It computes how far

the predictions of the network are from the true labels of the images and gives a loss score. The more accurate the predictions, the lower the loss. The choice of hyperparameters can greatly affect the convergence rate and overall performance of a network.

2.2.5 Convolutional neural network

Convolutional Neural networks (CNNs) (LeCun *et al.*, 1999) are widely used in a multitude of different computer vision tasks. They have proven to perform well on many image classification problems, including the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky *et al.*, 2015).

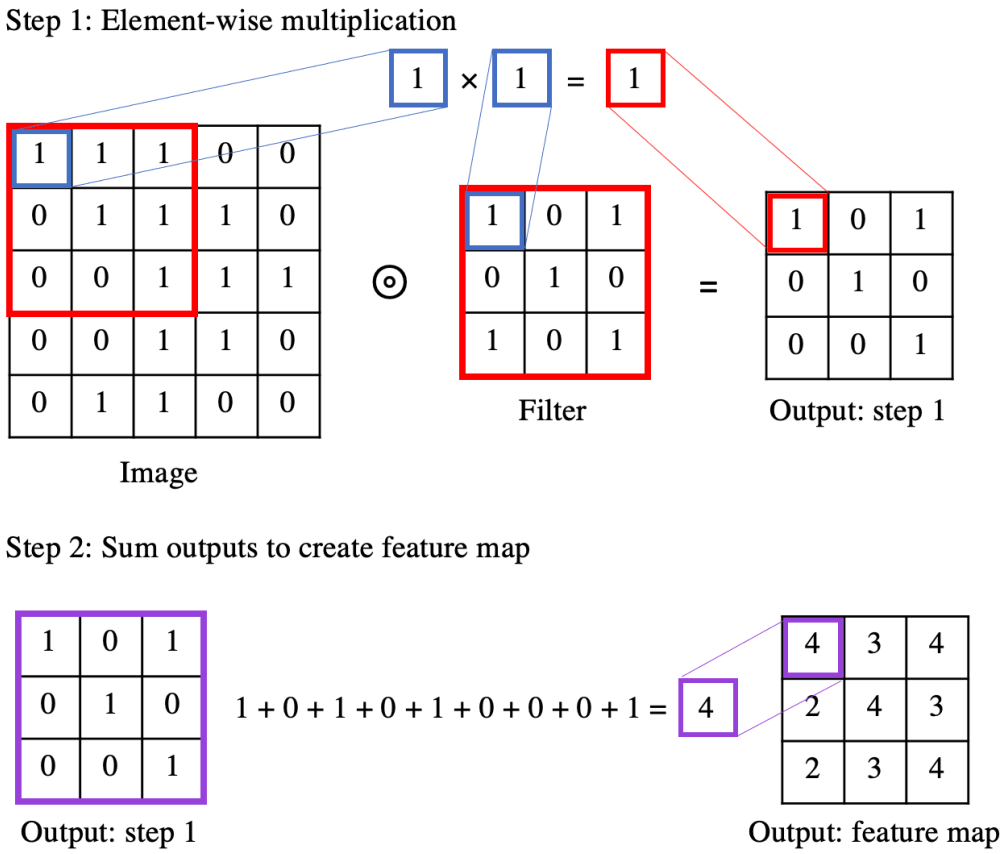


Figure 2.4: Example of how a feature map is produced using a filter in a convolutional layer. Step 1 is element-wise multiplication, where each element in the feature map is multiplied by the value in the image in the same position. Step 2 sums all the outputs from element-wise multiplication and adds the resultant value to the output feature map. The filter is then moved to the next position in the image and the process is repeated.

An image is made up of pixels, each of which has a value corresponding to the colour of the pixel. For a grayscale image, each pixel is denoted by a single number between 0 and 255. For an RGB image each pixel has three values, one for each of the three channels: red, green, and blue. A convolutional layer uses filters to create multiple feature maps from an input image.

The filter slides over the input image and performs element wise multiplication at each point. The outputs are then summed to generate the input for the feature map. The filter then slides to the next position on the input image and repeats this process. This process is shown in Figure 2.4 giving an example of a filter's resulting feature map on a given input image. The feature maps generated after each layer then become the input for the following layer. The number of filters, and the size of the filters are hyperparameters which can be set during the definition of the model architecture. Our figures show a 3x3 filter, but larger filters are also used in many models.

2.2.6 Fully connected layers

A fully connected layer (also called a dense layer) is a layer where all its neurons are connected to all the neurons in the preceding layer. This is different to a convolutional layer, where the neurons in one layer are only connected to a selection of neurons in the following layer. Fully connected layers are used as part of the classification part of the model. First the data is flattened to be one dimensional as this is the format needed to provide the classifications. Fully connected layers followed by a softmax activation function make the classifier, which provides the predictions.

2.2.7 Batch Normalisation

Batch normalisation (batch norm) allows a model to use normalised data throughout the training process (Ioffe and Szegedy, 2015). Much like the input data, which is

normalised prior to being fed through the network, see section 2.3.1, a batch norm layer normalises the outputs of the neurons from the previous layer prior to the activation function being applied. It does this for each batch of data. Throughout training, as the weights of the model are updated, it is possible for one weight to become significantly larger than the other weights. Deep learning networks do not handle large values well as they stop the model from converging. This is where batch norm comes in to normalise the data between layers. The addition of these layers makes training more stable and decreases the learning time due to the standardised data they produce.

For each batch of data, batch norm takes the outputs from the previous layer's neurons (O_i) and calculates the mean (μ) and standard deviation (σ) using Equation 3 and Equation 4 respectively where m is the number of neurons in the previous layer.

Equation 3

$$\mu_i = \frac{1}{m}\sum O_i$$

Equation 4

$$\sigma_i = \sqrt{\frac{1}{(m-1)}\sum (O_i - \mu)^2}$$

The outputs are then normalised using Equation 5.

Equation 5

$$\hat{O}_i = \frac{O_i - \mu_i}{\sigma_i}$$

For data to be normalised the mean needs to be scaled and shifted to zero and the standard deviation to one. This is done using Equation 6 and the batch norm layer's two trainable parameters γ and β, which are learned over the epochs like the neuron's weights and are different for each batch norm layer.

Equation 6

$$bn_{output} = \gamma \hat{O}_i + \beta$$

Much like the weights throughout the network, γ and β are adjusted throughout training to find the optimal values to produce the best predictions.

2.2.8 Max Pooling

Max pooling layers are used to down sample the size of the feature maps, so reduce their dimensions, produced after a convolutional layer. This reduces the number of parameters that the network needs to learn and therefore the computational cost of the model. It is also used as a method to combat overfitting resulting from using only convolutional layers. Over time, a CNN without any max pooling would start to associate the presence of features with a specific location. Adding max pooling to reduce the dimensionality means that the filters used to produce the feature maps will be looking at a larger portion of the input. This in turn means that the network will learn to be less dependent on the location of specific features.

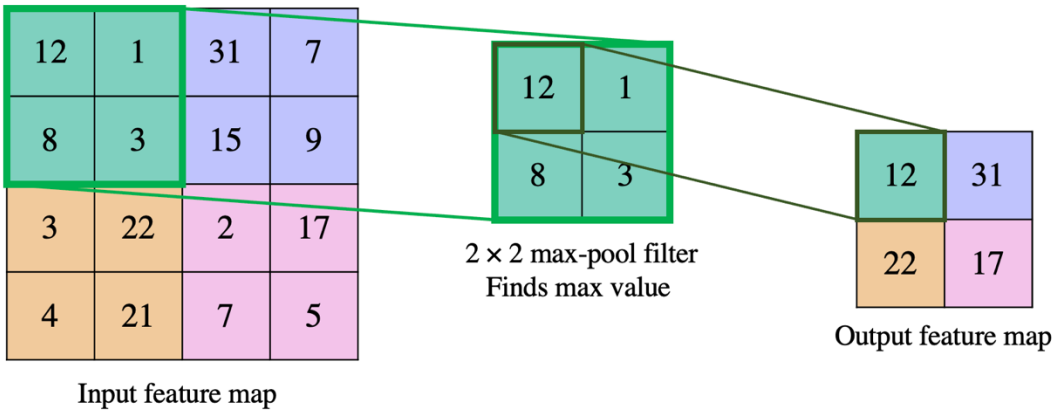


Figure 2.5: Example of max-pooling down sampling an input feature map. In each position the filter selects the maximum value to add to the output feature map. After this is complete at one position, the filter moves 2 pixels across or down to the next position and repeats the process.

The way a max pooling layer works is similar to a convolutional layer, using filters to perform an operation on the input feature map. Where it differs is that instead of

performing element wise multiplication, the filter instead picks out the maximum value in the receptive field. Most commonly, the filter size used by a max pooling layer is 2 x 2, which moves across the input feature map with a stride 2. The stride of a layer defines how much a filter moves across an image. In a convolutional layer it is often set as 1, meaning the filter moves by one pixel to the next location. A stride of 2 for a max pooling layer down samples the feature maps by a factor of 2. Figure 2.5 shows an example of the max pooling operation over an input feature map.

2.2.9 Dropout

Dropout works by randomly dropping (setting to zero) a number of output features of the layer throughout training (Srivastava *et al*., 2014). For example, if a layer would usually return a vector [1.2, 0.3, 1.1, 0.9, 0.8], after applying dropout there would randomly be zeros in a number of locations in the vector: [1.2, 0, 0, 0.9, 0.8]. The percentage of features which are dropped is set by the dropout rate, usually 0.5 in the networks used within this thesis.

This method helps to reduce overfitting by introducing noise into the results. Without the noise, the network may start to learn coincidental patterns which aren't significant. The noise breaks up these patterns so that the model can't remember them, thus only learning significant features.

2.2.10 Residual connections

In a deep learning network, each layer is built on top of the one before. This means that the output of one layer is only available as input to the following layer. Hence, if any information is lost along the way, for example due to a layer not having enough neurons to handle all the information available to it, then it cannot be regained at a later stage – it is lost for good. The loss of information in this way is called a representational bottleneck (Chollet, 2017a).

Backpropagation propagates a signal from the output layer of a network back through previous layers to the earlier layers, this is how a neural network is trained. The more layers that are stacked upon one another, i.e., the deeper the network, the more likely that the signal being propagated can become weaker or disappear entirely. If this happens then the network can no longer be trained. This is the problem of vanishing gradients (Hochreiter, 1998).

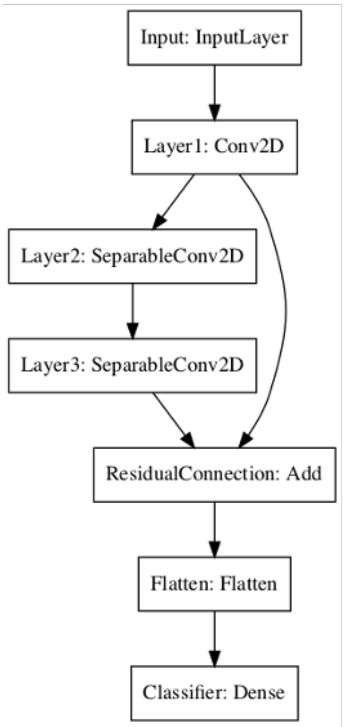


Figure 2.6: A short network with a residual connection. The output of layer 1 is used as input for layer 2 and also added to the output of layer 3.

Residual connections (He *et al*., 2016) provide a way to help tackle both representational bottlenecks and vanishing gradients. They provide a way for information from the output of earlier layers to be fed into later layers, thus limiting the amount of information lost. Figure 2.6 shows an example of a very short network with a residual connection. Here, it shows that the output of Layer1 is not only being used as input to Layer2 but is also being added to the output of Layer3. This creates a kind of shortcut for the earlier outputs to be included in the later layers. Residual connections are often used in deep neural networks with many layers to ensure that as much information is being maintained throughout the training process, and we experiment with several models that utilize them in this thesis.

2.2.11 Depthwise-separable convolutions

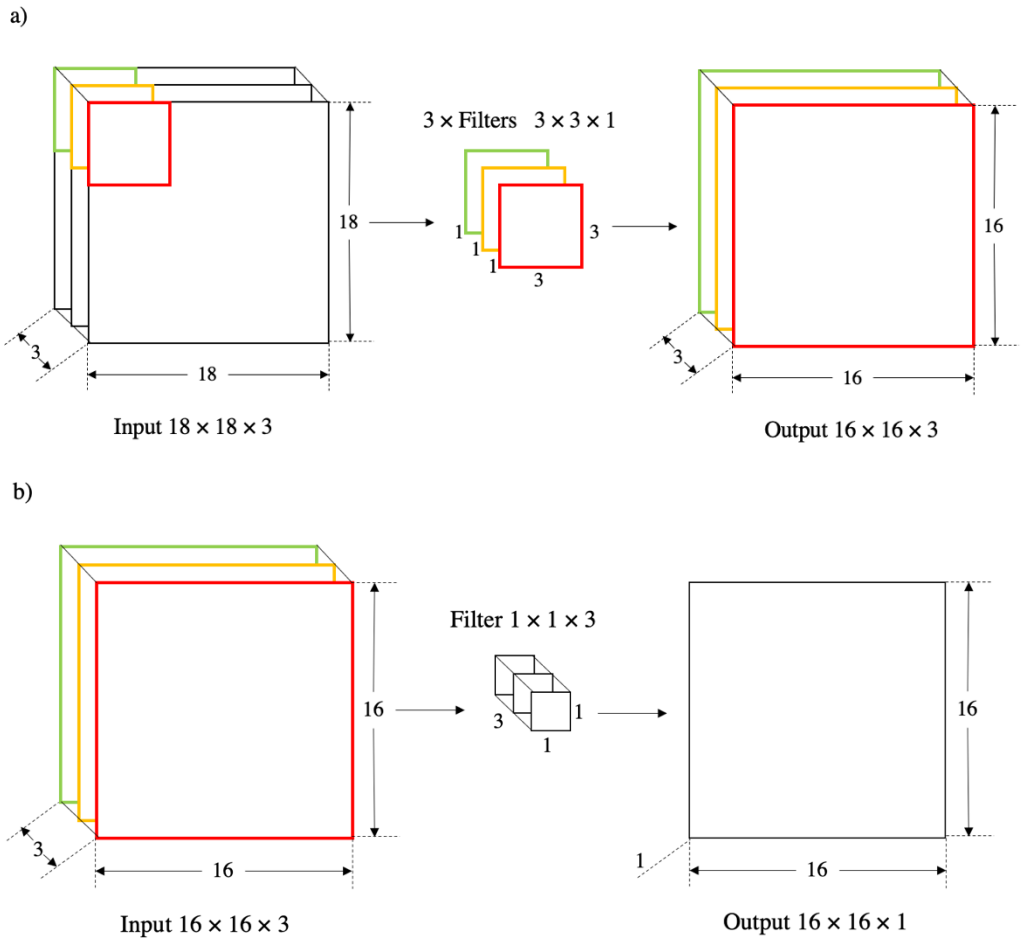


Figure 2.7: An example of a depthwise-separable convolution. a) shows the spatial convolution and b) shows the pointwise convolution. See text for full explanation.

Depthwise separable convolutional layers provide a less computationally demanding alternative to normal convolutional layers (Chollet, 2017b). These layers reduce the number of trainable parameters in a network, making them faster to run than using conventional convolutional networks. As well as reducing computational cost and time, the use of these layers can often increase the final accuracy gained on a problem.

Where the filters in a convolutional layer perform a convolution on all channels at once, the depthwise separable convolutional layer performs a spatial convolution on each individual channel from the input separately. Following this it performs a 1x1

pointwise convolution on all channels together. Figure 2.7 shows an example of how a depthwise separable convolution works. In Figure 2.7 a), the input is an 18x18 image with three channels. Three single channel 3x3 filters are used to perform spatial convolution on each of the channels of the input, so resulting in one channel of the output image for each of the filters. The output will then be a 16x16 image with three channels.

Figure 2.7 b) shows the pointwise convolution part of the operations. A 1x1 convolution is performed over the output 16x16x3 image, meaning that it iterates over every pixel. The 1x1 filter has three channels to match the number of channels in the image. The resulting image has the same height and width (16x16) but only one channel. The number of channels can be increased in the same manner as for a normal convolutional layer, by using multiple 1x1x3 filters to gain multiple 16x16x1 output feature maps.

2.2.12 Residual Attention Network

Attention modules are used within CNNs to get the network to give more attention to the important information in the data and disregard the unimportant background information. A residual Attention Network is built by stacking Attention Modules which generate attention-aware features (Wang *et al.*, 2017b). We use a residual attention network in our quantification chapter in the hope that it would be able to focus on the amount of disease present without being distracted by the complex background information.

The attention module is made up of two parts, the trunk branch, and the mask branch. The trunk branch uses residual units to perform feature processing to gain meaningful information about the input. With input x the output of the trunk branch is $T(x)$. The mask branch uses a bottom-up top-down structure to softly weight output features, to estimate the importance of the features, with the goal of improving trunk branch features. The bottom-up step down-samples the image with max pooling to gain information about the whole image, while the top-down step combines this global information with the original feature maps by up-sampling

(interpolation) to keep the output feature map the same size as the input feature map. The output of this branch is a learned mask $M(x)$ which is the same size as the input feature maps. It acts as a gate for passing information through the network. A simplified example is a mask that blocks out blue colour, thus eliminating the background sky information. The output of the attention module H is given by Equation 7 where i is the range of spatial information and c is the channel index.

Equation 7

$$H_{i,c}(x) = M_{i,c}(x) * T_{i,c}(x)$$

Figure 2.8 shows the architecture of the residual attention network used in Wang *et al*. (2017) which is used as the basis of our experiments with a residual attention model in Chapter 4.

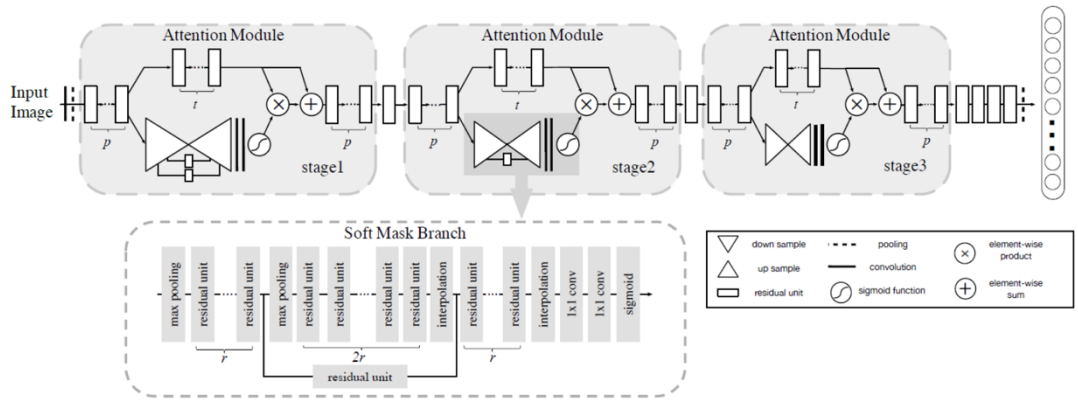


Figure 2.8: An example residual attention model. Here *p*, *r* and *t* are hyperparameters. *p* is the number of pre-processing residual units before splitting into trunk branch and mask branch. *t* is the number of residual units in the trunk branch and *r* is the number of residual units between the adjacent pooling layer in the mask branch (Wang *et al*., 2017, p4).

2.3 Basics of Training a Deep Learning Model

2.3.1 Data requirements

One of the biggest challenges for the successful application of machine learning techniques for the identification of plant diseases (or any image classification task for that matter) is the availability of data. The majority of these methods require large datasets of labelled or annotated images, which can be time-consuming to collect and process. For example, with plant disease detection, it is necessary to have a large number of images for each disease for each plant species that is being modelled. In the case of disease quantification, there needs to be enough images for every score or disease severity level.

One of the most famous, and largest, datasets used for image analysis with deep learning is the ImageNet dataset (Deng *et al.*, 2009). This dataset was created for use with object recognition software. The full dataset contains more than 14 million images with over 20,000 categories, however a smaller subset of this has been used in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky *et al.*, 2015). This challenge ran annually between 2010 and 2017, encouraging participants to develop and improve computer vision techniques for image classification and object recognition. Multiple winning networks created for this competition over the years are now used as the starting point for hundreds of deep learning problems, including the problem of crop disease detection.

Collecting a dataset of images for use with any deep learning problem is not quite as easy as simply gathering as many images as possible by any means. It is important to ensure that the dataset contains appropriate information for the required use case. The rest of this section will discuss the factors to consider when preparing a dataset for plant disease recognition and classification. These factors include range of conditions, controlled versus uncontrolled capture conditions, image quality issues, the number of images required and labelling and annotation requirements.

The most widely known, and one of the only openly available datasets used for the recognition of plant diseases, is the Plant Village dataset (Hughes and Salathe, 2016). This is a collection of almost 88,000 images taken in controlled conditions with 38 categories, each corresponding to a plant-disease pair. Each image contains a single diseased or healthy leaf taken from the plant and placed on a neutral background and photographed under different lighting conditions. While this dataset

was ground-breaking in the field of plant disease detection when it was first created, the use of controlled conditions in the photos means that it is not comprehensive enough to be useful for an automated system in field conditions.

The Plant Village dataset was useful for demonstrating the potential of deep learning methods for the classification of plant diseases; however, in order to create a model that will be useful in realistic growth conditions, it is now important to collect datasets which accurately represent those conditions. PlantDoc (Singh *et al.*, 2020) is a dataset created to cover many of the diseases present in the Plant Village dataset, but the images instead cover real field conditions. Here, images were downloaded from the internet and checked by members of the team before being added to the new dataset. This resulted in almost 3000 images, spanning 27 of the categories from Plant Village (any classes with fewer than 50 samples were removed for this dataset). This is a step in the right direction, but there is a distinct possibility of mis-classified samples within the dataset due to them being taken from internet searches. Also, with it still being a relatively small dataset spanning a lot of classes, there is still a high chance that not all conditions are being covered.

For studies that are looking at building a model for a certain crop, as for the work within this thesis, it is unlikely that there are already datasets openly ready and available for use. This means that, for each case, there will be a large collection operation required prior to any numerical experiments. The result is usually a relatively small dataset with few categories (in some cases only two: diseased or healthy). While these can be useful for the problem at hand, there is still some way to go with generating a larger dataset to be used in a wider variety of cases.

The collection of a dataset which sufficiently covers each category can be a time-consuming task, often requiring the specialist knowledge of an expert pathologist and multiple volunteers to take the pictures. Furthermore, it is not simply a case of capturing a large number of images for each category, but also including a representative range of conditions. If the model is to be used for identifying diseases in the field, then the range of typical conditions that could be encountered in the field need to be represented. This includes:

- The variation in crop varieties/ species – for example different leaf colours or sizes
- State of the crop – seedling, adult, flowering, mature with seed
- Stage and severity of the disease – early to late infection, mild to severe symptoms (particularly important for disease quantification)
- Weather and lighting conditions – Full sun, sun, and cloud, overcast, rain, etc.
- Background information – this needs to be consistent throughout the dataset. Having one class with different background information to the rest (e.g., glasshouse instead of field) will cause issues in training
- Image qualities – Focus, depth of field, range of angles

The main point to remember when creating a dataset for deep learning is that the conditions present need to be consistent between classes. Any class (category) containing conditions which are not present in the others, for example one class having sky in the background whereas no other does, will cause the network to learn the wrong information about that class and classify it based on the presence of sky, rather than the disease information.

Another factor to consider if working with real condition images is the diversity of background information, which might contribute negatively to the training process by distracting from the features that are of interest. If the images are collected in a field, for example, this may not be too much of an issue as the field conditions are likely to be relatively uniform. However, if the images are of a plant species which grows in various wild locations, then a vast array of background information can be expected. Where possible, the full range of diverse background conditions should be represented in images across all classes.

The number of images is also important. The number of images to aim for per category will depend on the complexity of the problem at hand. A simpler problem, for example a binary classification problem of healthy or diseased, will require fewer images than a classification problem with multiple diseases with similar symptoms. However, the general rule of thumb with deep learning datasets is the more data, the

better (ideally hundreds, if not thousands or even tens of thousands, of images per category in our opinion). The more images the network has to learn from, the better its performance is likely to be. It is also best if the data is relatively well balanced, with a similar number of images in each category, so the network does not learn a bias towards one class due to it having significantly more training samples than the others.

One technique to increase the number of images in a dataset where it is not possible to collect more is data augmentation. Augmenting the data involves performing multiple transformations on each image to add new samples to the dataset (Perez and Wang, 2017). For example, an image may be mirrored, flipped horizontally or vertically, rotated, or shifted to create tens of new images from a single sample. The main drawback of this is that there is no actual new data created, just variations of existing data. This means that the original dataset still needs to contain enough variation so that the network can learn enough to form predictions. There are other methods for working with smaller datasets, however where possible it is always better to collect more data.

After collecting all available data, it will then need to be labelled and collated into a full dataset. For best results, a pathologist will need to label each image with the correct category, either as the images are taken, or by going through all data and assigning categories later. This of course can be incredibly time consuming and can result in misclassifications within the dataset if a pathologist is not available. In cases where different visualisation techniques are being used with the dataset, it may also be necessary to annotate the data with further information (e.g., a bounding box around a disease lesion). This often has to be done manually on each image and is a huge undertaking, requiring many hours of work and in some cases specialised knowledge.

Once all labelling and annotation is complete, the data can be sorted into the train, validation and test sets and go through pre-processing prior to being used with a deep learning model. It is important to separate the data into the three subsets before any pre-processing occurs to ensure that data leakage does not affect the model. If pre-processing is done on all the data together, then it will have access to knowledge

about the dataset as a whole which will influence the training. Data leakage can happen when the model gains information about the test or validation set during training, which means it can make more accurate predictions which will not be a true representation of the model's ability on unseen data.

Deep learning networks take floating-point tensors as their input, so any data used for training or evaluating a model should be formatted as such. In computer science a tensor is a multi-dimensional array which stores data of a specific type. With image classification, the images are often stored in the dataset as .JPEG or .PNG files. Pre-processing is the method of converting these image files into floating-point tensors prior to feeding them into a network. First the images are decoded into their RGB pixel grids, where each pixel has a value between 0 and 255 for each of the three colour channels. Following this, the data is normalised by dividing each value by 255 to get all data in the range 0 – 1, giving a dataset for each image in the format A[i][j][k]. Normalising is important because feeding larger or heterogeneous values into a network can stop the model from converging.

2.3.2 Model Training

The deep learning networks that are used for the identification of crop diseases are often a type of CNN trained to perform their task by image analysis and classification. CNNs are used because of their strong ability to extract useful features from the images (Yamashita *et al.*, 2018). Each network has an input layer where the data (in this case images) is fed into the network and an output layer, which is where the predictions are given. Between these are a number of hidden layers which perform feature extraction. The number, size, and type of layers in a deep learning network is referred to as the model architecture.

Feature extraction is the process where the network learns features from the images that are relevant for the predictions. Earlier hidden layers, closer to the input layer, learn low level features, for example lines and edges. As the images are progressed through the hidden layers, the extracted features become increasingly complex.

Figure 2.9 shows a simplified representation of the deep learning workflow for image analysis. Feature selection prior to deep learning approaches was done manually by experts with domain knowledge and was a highly time-consuming

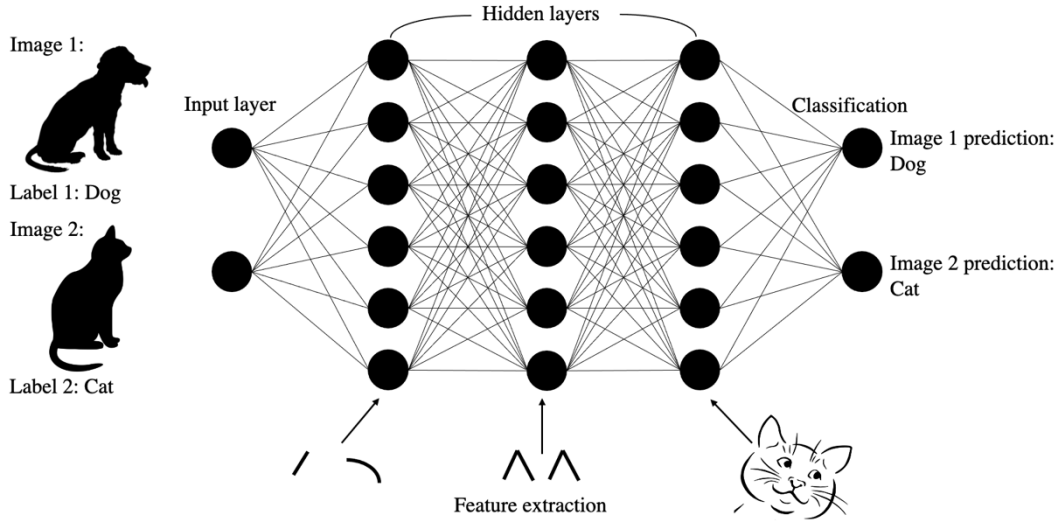


Figure 2.9: A simplified depiction of a deep learning model for the classification of dog and cat images. The model takes images at input, has multiple hidden layers for feature extraction, and produces classification predictions as output.

undertaking. An advantage of the manual approach is that the features are often meaningful features that humans can relate to, such as number of eyes, number of legs, etc. The features inherent to deep learning networks are learnt during the training phase by adjusting the weights in the network to increase the performance. This automation brings huge advantages in terms of saving time and not relying on expert domain knowledge but has the disadvantages that the features are often not easy to extract and interpret. Feature selection is thus part of the overall network weight optimisation process.

Training a network for image analysis requires a large dataset of images to work with, ideally tens of thousands of images depending on the problem. The image dataset is split into smaller datasets, usually train, validation, and test sets, however some studies use only a train set and test set. In cases where there is enough training data, having a separate validation set is usually the best method. However, if the dataset used is smaller and having a separate validation set would make the train and/or test set too small, then studies often use other validation techniques such as

K-fold validation and iterated K-fold validation and shuffling (Refaeilzadeh, Tang and Liu, 2009). The split of the dataset between these sets varies over different studies, but the bulk of the images is always contained within the train set, with a smaller amount in the validation and test sets.

When training a model, the train set of images is fed into the network in small batches (e.g., 64 images at a time). Once all the images have been through once, this is known as an epoch. The network will be trained for a certain number of epochs, as defined by the programmer. The number of epochs should be sufficient to reach convergence during the optimisation process to deliver the best performance. Training for too many epochs can waste computational resources and lead to overfitting (see below). This number is often picked by taking an educated guess based on previous research and experiments in similar fields. Many studies will try multiple experiments with different numbers of epochs before finding the number that yields the best results for their data.

Between every epoch, the current network parameters are evaluated against the validation set to ensure that the training process is not overfitting to the data in the train set. If it were overfitting, then it would be learning features that are specific to the images in the train set and would not be able to generalise to new data of the same type once training is complete, as would be found in the test set. For example, if the network were looking for a certain feature from the train data which did not appear in the validation data, such as the presence of different soil colours, the validation set would highlight a discrepancy in the performance. The network parameters would then be readjusted to ensure that it is not using that soil information from the train set for classifying that disease, but rather the disease information on the plant instead. Ideally, the train, validation and test tests would all contain both colours of soil.

Throughout training, the network is constantly adjusting its internal parameters after each batch and epoch to allow it to better make predictions about the images. As it learns, the accuracy of the predictions increases until it reaches a peak at the end of training. At this point, a new, trained instance of the network is defined and trained for the same number of epochs that returned the peak accuracy. It is trained on all

available training data, the train and validation sets combined. Following this, the network can be evaluated on the test set of images. This is a set of images of the same kind as contained in the train and validation sets, but that the network has never seen before. This shows how the trained network performs on brand new images to give a final accuracy rating. The whole process from start to finish can take a long time, from hours to days, to even months. This depends on multiple factors, such as the computing power available, the size of the network and the size of the dataset. For example, a model with 10 convolutional layers would likely take double the time to train than a model with 5 convolutional layers would using the same data. Furthermore, training the same model using one or many GPUs would be significantly faster (for example hours or days) than training using only CPUs (for example tens of days or months).

2.3.3 Transfer Learning

A lot of studies begin their experiments by using transfer learning with their datasets. This is a method that takes the knowledge learned by a previously trained network and applies it to the new problem. The main advantage of this is that it is relatively quick compared to training a deep learning network from scratch. Some examples of networks often used for transfer learning are AlexNet (Krizhevsky, Sutskever and Hinton, 2012), GoogLeNet (Szegedy *et al.*, 2014), VGGNet (Simonyan and Zisserman, 2015), ResNet (He *et al.*, 2016), Inception V4 (Szegedy, Ioffe, *et al.*, 2016) and MobileNet (Howard *et al.*, 2017).

Transfer learning uses a network which has been fully trained on a large dataset (often the ImageNet dataset described in section 2.2.2). The pre-trained network is divided into two parts; the convolutional base, which is the part that performs feature extraction on the images, and the fully connected classifier, which forms predictions about the images. Depending on the method used, all or parts of the network are repurposed for the new dataset. In some cases, only the network structure is used and is retrained for the new problem without using the pre-trained knowledge. We use two different methods of transfer learning over the course of this PhD, which will

each be described in the following sub-sections. These both use the same pre-trained models and fully connected classifier, however they are trained in different ways.

2.3.3.1 Method 1

The first method of transfer learning that we will describe takes the convolutional base of the pre-trained network, complete with learned weights from having been trained on a larger dataset. The original fully connected classifier is discarded. This is because the features learned by the convolutional base are often more general and able to be reused, but the classifier is likely to have learned representations which are much more specific to the original training data.

We feed the data for the new problem (here for wheat disease classification) through the convolutional base a single time, allowing the convolutional base to perform feature extraction. The extracted features are then used to train a newly defined, fully-connected classifier network. This method is much quicker and cheaper to run than training the entire model from scratch (see method 2), as the convolutional base, which is the most computationally expensive part of the network, only needs to be run once.

2.3.3.2 Method 2

The second method we use takes the entire pre-trained network, without separating it into the convolutional base and fully-connected classifier. In this case, we freeze the weights for the convolutional portion of the network. This means that they stay the same throughout training and can no longer be changed as the model learns. The fully-connected classifier portion of the network, however, does learn through training and its weights and parameters are constantly updated as it learns.

The new data is sent through the entire network for multiple training epochs, much like training a full model from scratch, until a final accuracy is reached. This is a much more time consuming and expensive method, however it allows the user to

retrieve information from the individual layers in the convolutional base. This information can be used for analysing the performance of the model by generating area importance heatmaps (Selvaraju *et al*., 2020), for example.

Chapter 3 Classification of Wheat Diseases

The first step to producing an automated deep learning model for scoring wheat disease in the field is to ensure that the defining features of the disease can be detected and classified from realistic field data. If this is not possible then there is no way that it would be able to identify and then quantify the amount of disease present. It is important to use data collected in real growth situations for training a model as this will reflect the conditions that would be encountered in the field. Including this kind of data in a training dataset will allow the model to learn to ignore background features. Furthermore, the disease information can be affected by the environment around it. For example, the lighting conditions that would be found in the field are different to those that would be experienced in controlled laboratory conditions, therefore the appearance of the disease symptoms will be different in each situation due to the variance in light.

For any model to be useful in real field situations, it would ideally need to perform at a level of accuracy and speed at least equal to the human pathologists who would usually be classifying these diseases. However, a slightly lower accuracy than human classification, several percentage points for example, would also be acceptable as our model could be taken out multiple times to build a clearer picture of the infections in a certain location. We explore the human and model accuracies for our problem in section 3.1.5. This could be done by anyone and wouldn't require the time of a trained pathologist to do so.

In this chapter we aimed to produce a viable deep learning model for wheat disease detection in the field. We hypothesized that convolutional neural networks (CNNs) would be able to handle the complex images and be able to classify the different diseases. We collected a large, comprehensive dataset of images taken in realistic growth conditions, including four of the most important wheat foliar diseases in the UK (see section 1.1.2). We experimented with multiple deep learning architectures to find the configuration that classifies the images into their disease category with the highest accuracy. Ideally, any model would need to perform at least as well as

human pathologists to be useful in the field, so we compared our best performing model's classification power against five expert pathologists with backgrounds in identifying these wheat diseases.

3.1 Methods

3.1.1 Collection of a dataset of wheat disease images in realistic growth conditions

Training a network to classify images requires a large dataset of images sorted into categories. A dataset that could be used to train a network to identify and classify wheat diseases needed to be collected. For this work we aimed to define and train a deep learning model to classify images containing a single disease using images taken in realistic growth conditions, including complex background information.

The diseases we decided to include in our dataset were Septoria, yellow rust, brown rust, and mildew. We also included a healthy category to ensure that any trained model would not classify healthy leaves as having a disease due to a lack of any other option. The locations for photography were chosen by my supervisor, James Brown (in particular locations in Norfolk), and pathologists from RAGT, Limagrain and KWS. The majority of field photographs were taken in plant breeders' trials containing thousands of wheat lines. A minority were taken in farmers' fields containing diverse commercial varieties, which were not identified or marked in any way for the image collection. These locations provided plenty of variation in the appearance of the wheat over different varieties, as well as a range of susceptibility to the diseases. This was important to collect a variety of infection levels for the dataset.

Photos were taken over a three-week period in diverse weather conditions. However, due to a lack of mildew in field situations thanks to breeders having good control over mildew in the UK (Brown, 2015), many mildew images were taken in

glasshouse conditions. The images were taken from diverse lines from genetic and plant breeding experiments. Each photography location was identified by a pathologist as having only one disease present during the photography window and the resulting photographs were all labelled with that disease. This was possible due to many of the photography locations being breeding trials where other diseases were controlled with fungicides. Other locations were chosen because they were known to have high levels of one disease and low levels of other diseases, reducing the risk of mixing diseases. We were able to make logical decisions for photographing in farmers' fields based on the part of the country and what part of the growth season it was. Different diseases can be more prevalent in some areas and at certain points in the season.

It would be ideal for any model trained to identify wheat diseases to be deployed on a mobile device, as this way any person would be able to take it out to the field without having to rely on a trained pathologist. To replicate the capture conditions for deployment on a mobile device, the majority of the photographs were taken using various iOS and Android smartphones, with a range of camera resolutions. Some images were also taken using a low-resolution setting of a digital camera. The resultant images had resolutions that ranged from 6 – 16 Megapixels.

Photography volunteers were given instructions for capturing their images to ensure that the data was collected consistently over the whole dataset. They were told to collect images of leaves still attached to the plant, where the main focus was from one up to five leaves all infected with the same disease (or healthy). The images were to contain normal background information and be taken at a range of distances and angles to emulate the different capture styles that would be used by the future user. The volunteers were also provided with a selection of sample images to allow them to see visually what the images we needed would look like.

We were conscious of the need to collect images which accurately represented the range of conditions that would typically be found in realistic growth environments. All weather and light conditions which could affect the appearance of the diseases needed to be included, for example sun, cloud, and rain. We also had to make sure

that we captured variation in the wheat plants that we imaged, including leaf colour and growth stage of the plants. Additionally, it is important to be able to identify these diseases through all stages of their life cycle and over all possible symptoms, so we needed to include the different stages and symptoms in the images we collected. This was achieved by visiting some of the sites multiple times and photographing each category across the entire growth period to include early, middle, and late infections. Making many visits to photography locations for each category allowed us to collect a good range of variation in symptoms. While trying to capture as diverse a range of conditions in our dataset as possible, we had to ensure that this range was as consistent as possible over all of the categories to ensure that our models did not learn something about a single category due to the absence of a certain attribute in all other categories. Information about the location, disease, weather conditions and date were noted for each collection session.

As with the other categories, the mildew was imaged over the entire growth period, and where possible, with different weather conditions. However, naturally as many images were captured in glasshouses, there was a difference in the background information included in these images when compared to the field images.

We wanted to make sure that the dataset included as much useful information (disease and leaf tissue) with as few misclassifications as possible. Every image was loaded individually using preview and checked for quality control purposes. Due to the way we classified the images initially, by labelling all images from one location with a single disease (or healthy), there was the potential of other diseases forming multiple infections on the plants in some of the images. In some cases, other diseases weren't entirely controlled by fungicides, so some images were taken of the wrong disease in error. During quality control, any image that contained a disease other than the one that matched the initial label was removed. This included images with multiple diseases present.

As with any task carried out by humans, there was also the possibility of human error in the photographing. This meant that some images contained no useful information about the disease (for example an accidental image taken which contains

predominantly soil and little to no leaf information), were too blurry to be included or the useful leaf and disease information was covered by a hand or shoe etc. All of these images were removed, however we kept images that had additional features (like fingers, boots, or line markers) as long as the relevant disease information was not obstructed. This was done to ensure that the images were as representative as possible of the type that would be taken when the model is taken out on a mobile device in future. We cannot assume that every image taken by every person using the model would be perfect without additional features or manageable amounts of blur. All images that passed quality control were assigned a label corresponding to the disease present.

We took every precaution to ensure that the images were correctly classified prior to being included in the dataset, however due to the nature of the problem there will naturally be some misclassifications included. The way the images were labelled as a group depending on where they were photographed and then being quality controlled by a single person means that some misclassifications would have slipped through. In an ideal world, the data would be checked by multiple pathologists before using to train any models to ensure that as many images were labelled correctly.

At this point, we sorted the images into a train, validation and test set ready for use with our deep learning models. Each category was shuffled prior to being divided into the three sets. This meant that images from each location were likely to be spread across the three sets, ensuring that a model was trained and tested on the same variety of conditions. We split our dataset as follows: 60% of images in the train set and 20% of images in each the validation and test sets. Every category was divided in these proportions. We felt that our dataset was of sufficient size for this split to contain enough variation across all three subsets. The exact numbers contained in each category for each set of data is discussed in section 3.2.1.

3.1.2 Transfer learning with our dataset

Our goal was to develop and train deep learning networks to classify images as Septoria, yellow rust, brown rust, mildew, or no disease from images taken in realistic growth conditions. We decided that the best place to start was by utilising pre-trained networks through transfer learning. By using these models which have already been trained on a larger dataset, we were able to extract features from our dataset much quicker than training a model from scratch. This meant that we could get some results early in the process which would tell us whether or not deep learning networks could handle a complex dataset like ours without us having to do any pre-processing to our images prior to training.

For the experiments in this chapter, we used keras version 2.2.0 (Chollet and others, 2015) to define, train and evaluate all the deep learning models in python 3.5.1. We experimented with four pre-trained networks (MobileNet (Howard *et al*., 2017), InceptionV3 (Szegedy *et al*., 2015), VGG16 (Simonyan and Zisserman, 2015) and Xception (Chollet, 2017b)) to extract features from our dataset. These four models were chosen as they were available within keras and could be imported with the pre-trained weights using only a few lines of code. One further model, Resnet50 (He *et al*., 2016), was tried, however the code repeatedly failed, so it was removed from our experiments.
We used method 1 for transfer learning, see section 2.2.3.1. We removed the part of the pre-trained network which provides classifications and sent our images once through the part of the network which extracts features, the pre-trained convolutional base. Having extracted the features, we defined a short classifier network which took the extracted features as input for training. The architecture of the classifier network was the same for all four pre-trained networks (see Figure 3.1) and used an RMSProp optimizer (Hinton and Tieleman, 2012). For each model we use classification accuracy and F1 score (Goutte and Gaussier, 2005) to evaluate performance.

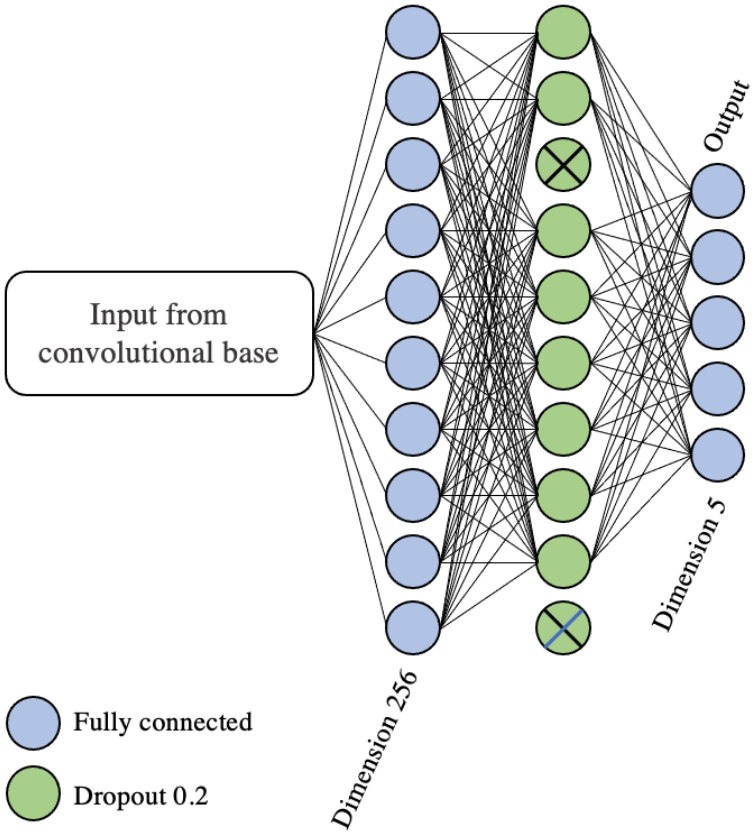


Figure 3.1: An artistic representation of the architecture of our fully connected classifier network.

Our first experiments used input image sizes of (150,150) or (128,128) depending on the individual network requirements for input size. Batch size 32 was used for training the classifier network. Due to the complexity of our images, we also experimented with a larger batch size and input image sizes. We multiplied each component by four for every model, giving a batch size of 128 and input image sizes of (512,512) or (600,600).

3.1.3 Defining our own model architectures

The results from transfer learning suggested that deep learning networks can deal with complex images such as those taken for our dataset, under real field conditions,

for detecting wheat diseases. The results are above the baseline metrics that we calculated based on our data.

To find a baseline for our models, we used the zero rule, which is widely used in the machine learning community. This rule states that if the classifier always chooses the most common class (the class with the most samples), then it will be right $x\%$ of the time, where x is the percentage of the whole dataset contained within the most common class. In our case this is the Septoria class, containing 36.84% of the test data. Therefore, our zero-rule baseline for classification accuracy is 36.84%.

Table 3.1: The percentage of the full dataset contained within each individual category.

Category	**Percentage of dataset**
Brown rust	13.06%
Healthy	11.86%
Mildew	15.35%
Septoria	36.84%
Yellow Rust	22.89%
Total	**100%**

Another baseline metric we used is that of the weighted random guess. If our dataset were perfectly balanced then a random guess would have a one in five chance of being correct, so would have a 20% accuracy. As our dataset is imbalanced, we use a weighted random guess which squares the percentages (in decimal format e.g., 30% as 0.3) of the dataset for each category and adds them together. Table 3.1 shows the percentage of the full dataset contained within each category. For our dataset, the accuracy baseline for a weighted random guess is 24.28%. Any model with an accuracy above this point is adding value, however, to be useful as a predictor is also needs to be above the zero-rule baseline.

Although the accuracies of the pre-trained network are relatively high, they were not as high as would be ideal for use in the field. We estimate that a trained pathologist

undertaking the same task in the field would be able to classify the diseases with accuracy percentages of 95% or higher. We decided to explore other network architectures to see if we could optimize the classification accuracy. We experimented with multiple deep learning architectures, based on convolutional neural networks (CNNs), where we changed the number of layers and the types of layers used to find the combination which provided the highest classification accuracy on the test data. We also experimented by changing the hyperparameters (such as input image size, batch size, learning rate, number of epochs) to further optimise the performance of our models.

Each model was trained, validated, and tested using the method described in section 2.2.2 All networks were trained using the Norwich Bioscience Institute's high-performance computing (HPC) facilities. The pre-trained networks used central processing units (CPUs) on the HPC clusters, whilst our own networks were trained using graphics processing units (GPUs) to decrease training time.

3.1.4 Masking images

While it is important for a trained model to be able to classify the images with high accuracy, it is no use if the classifications are being made based off of the wrong information. In our case, we want the model to be using the leaf and disease information to drive classification, and not any arbitrary background features which may be present in the dataset.

As an experiment to determine whether our model was using the correct information for its classifications, we used black masks to cover the important information (disease lesions and leaf tissue) in a selection of images from our test dataset. A rectangular mask was placed over the foreground diseased (or healthy) leaves in images from each category. We theorised that with a mask covering this information, a deep learning model would have trouble to correctly classify the images and would instead have to 'guess' rather than make an informed decision. This would result in a drop in classification accuracy for the masked image dataset when compared with

the same images with no mask. The masks were added using the editor functions in the Preview app, available on apple computers. The number of masked images created for each category is described in section 3.2.5.

We sent the newly masked images and their original counterparts through the trained network to gain predictions for each set. Confusion matrices were used to compare the results.

3.1.5 An experiment to evaluate our model against human participants

Table 3.2: Sample of network results on test data to show how images were selected for experiment with human participants

Filename	**Prediction score**	**Network prediction**	**Correct/ Incorrect**
Mildew1936	35.7	YellowRust	Incorrect
Healthy1598	36.1	Mildew	Incorrect
Yrust240	40.9	BrownRust	Incorrect
⋮	⋮	⋮	⋮
Brust173	39.8	BrownRust	Correct
Yrust3027	57.4	YellowRust	Correct
Mildew1282	60.1	Mildew	Correct
Healthy1976	63.9	Healthy	Correct
Mildew1121	66.0	Mildew	Correct
Healthy1744	68.9	Healthy	Correct

We designed an experiment which allowed us to evaluate the performance of our model against multiple trained pathologists with backgrounds in working with these diseases on wheat. The experiment took a small subset of images (describe in the next paragraph) from the original dataset and gained classification accuracy results

for our model and five pathologist participants. The smaller dataset used for this experiment contained images taken from the test set of images. It contained 999 images, which is about a quarter of the size of the original test set. This number represents a reasonable compromise between getting a good statistical assessment on the performance of the pathologists and the network and the number of images that could realistically be classified by a human in a few hours.

To choose the images for this experiment we first took all the images from the test set which were incorrectly classified by the network. We did this to see if there was any correlation between the images that the network struggled to classify and those that the pathologists struggled with. From there, we ordered the rest of the images by the network's prediction scores and added every fourth image to the dataset, see Table 3.2. This ensured that the dataset contained a selection of images that, from the network's perspective, had varying degrees of difficulty in classification and were well distributed across the five categories. The images were shuffled randomly to ensure that there was no way to know which disease would appear next when presented to the participants.

Table 3.3: Tag numbers for each category in qtagger

Category	**Tag**
Brown Rust	1
Yellow Rust	2
Septoria	3
Mildew	4
Healthy	5

We used a tagging system, qtagger (Hartley, no date) (link in reference), created by supervisory team member, Matthew Hartley, which allowed us to present the images in our smaller set to each of the five pathologist participants in the same order. The tagging system worked by loading each image individually onto the screen. The

participants then assigned a tag to the image corresponding to the class it contained, and this action caused the next image to be loaded, replacing the previous. Each category was assigned a number between one and five (see Table 3.3) so that, instead of having to find the right category name each time, the participants only needed to enter the number which represented the category they had chosen. This meant that the tag could be entered quickly without much additional effort. Qtagger is open source and freely available. It is implemented in python and uses the Qt framework as a graphical interface and the dtool library for data management.

The system made a record of the tags made by each participant, and the time taken between clicks. We could then compare the classifications for each participant with those of the network and the true labels for each category, and also get a rough idea of the time taken for each participant to classify every image in the dataset. We used confusion matrices to display the classification results for each participant and our model.

Initially, we presented this experiment to the pathologist participants as individuals, however we conducted a second experiment where the group were allowed to discuss their classifications and even remove images that they were not able to agree upon. For this second experiment, we removed all images that all five pathologists classified correctly in the first experiment. This was done to save time, as we theorised that there was a high chance that they would classify these images the same a second time and working as a group was likely to take longer than individual classification.

3.2 Results

3.2.1 Dataset collection

To evaluate the potential of automated disease detection from realistic field images with complex background information, we collected images that reflected the

conditions found in typical growth situations. With the help of volunteers, we collected over 20,000 images (before quality control) across the five categories. Table 3.4 gives information about the photography locations, approximate number of images collected and weather information where available. In all categories other than mildew, all of the images were taken in outdoor field conditions, except for a very small amount in the healthy category that were taken in glasshouses. Approximately 80% of mildew images were taken in glasshouse conditions due to a lack of available mildew-infected field plots.

The dataset contains images of wheat leaves with either Septoria, yellow rust, brown rust, mildew, or no disease, taken in real growth conditions. The photos include complex backgrounds, as would be encountered in the field. We took care to include many different conditions which are found in the field, firstly, so the model can work in all situations and secondly, to reduce the risk of the model learning features that are not related to the disease. The different conditions Ie camera position, weather, lights, wheat varieties including shades of green of leaves, age of plant and the life cycle stage of any disease present. There were similarly diverse conditions in the photos of mildew on Iheat in glasshouses, however the background information in many of the glasshouse pictures was different than for those images taken in the field. Figure 3.2 and Figure 3.3 show example images for each category in field conditions, as well as an example image for mildew taken in glass house conditions.

After image collection was complete, the images were quality controlled and the final dataset contained 19,172 images, spread across all five categories. Each category was split so that 60% of images were placed into the train set, and 20% went into each the validation and test set. Table 3.5 shows the distribution of images in our dataset overall and through the three subsets.

Table 3.4: Information about the collection of images, including disease category, location, company, date, and photographer

Disease	Approx number	Location	Weather	Company	Date	Notes	Collected by
Healthy	450	Newton, Cambridge	sun/ cloud	KWS	14/03/2019	Seedling	Megan Long
Healthy	450	Newton, Cambridge		KWS	14/03/2019	glasshouse	Megan Long
Mildew	550	Newton, Cambridge		KWS	14/03/2019	glasshouse	Megan Long
Yellow rust	550	Newton, Cambridge	sun/ cloud	KWS	01/05/2019	teenage' plants	Megan Long
Healthy	250	Newton, Cambridge	sun/ cloud	KWS	01/05/2019	teenage' plants	Megan Long
Septoria	1550	Sutton Bonnington,Nottingham	sun/ cloud	RAGT - student	14/05/2019		Megan Long
Yellow rust	850	Ickleton, Cambridge	cloud/rain	RAGT	04/06/2019		Megan Long
Septoria	400	Sutton Bonnington,Nottingham	rain	RAGT - student	11/06/2019		Megan Long
Brown rust	1050	Woolpit, Suffolk	cloud/rain	Limagrain	12/06/2019		Megan Long
Mildew	650	JIC		JIC	21/06/2019	glasshouse	James Brown
Septoria	300	Sutton Bonnington,Nottingham	sun/ cloud	RAGT - student	21/06/2019		Patrick Seed
Yellow rust	1250	Rothwell, Lincolnshire	sun/ cloud	Limagrain	18/06/2019		Megan Long
Mildew	650	Rothwell, Lincolnshire	sun/ cloud	Limagrain	18/06/2019		Megan Long
Septoria	80	Unknown location Ireland		Teagasc			Ewen Mullins
Brown rust	850	Cambridge	cloud	NIAB	25/06/2019		Megan Long
Septoria	350	Rothwell, Lincolnshire		Limagrain	18/06/2019		Edward Flatman
Healthy	1250	Morley Farm, Norfolk	sun/ cloud	JIC	27/06/2019		Megan Long
Mildew	1050	JIC		JIC	28/06/2019	glasshouse	Megan Long
Brown rust	850	Thriplow, Cambridge		KWS	01/07/2019		Douglas Brown
Septoria	1050	Sutton Bonnington,Nottingham		RAGT - student	03/07/2019		Douglas Brown
Yellow rust	1350	Rothwell, Lincolnshire		Limagrain	02/07/2019		Douglas Brown
Septoria	1250	Morley Farm, Norfolk		JIC	04/07/2019		Douglas Brown
Septoria	550	Thriplow, Cambridge		KWS	16/07/2019		Douglas Brown
Septoria	1650	Sutton Bonnington,Nottingham		RAGT - student	15/07/2019		Douglas Brown
Yellow rust	450	Thriplow, Cambridge		KWS	16/07/2019		Douglas Brown
Mildew	650	JIC		JIC	18/07/2019		Jade
Septoria	400	Rothwell, Lincolnshire		Limagrain	11/07/2019		Edward Flatman
Yellow rust	300	Rothwell, Lincolnshire		Limagrain	23/07/2019		Simon Berry, Petros Zafeiriou

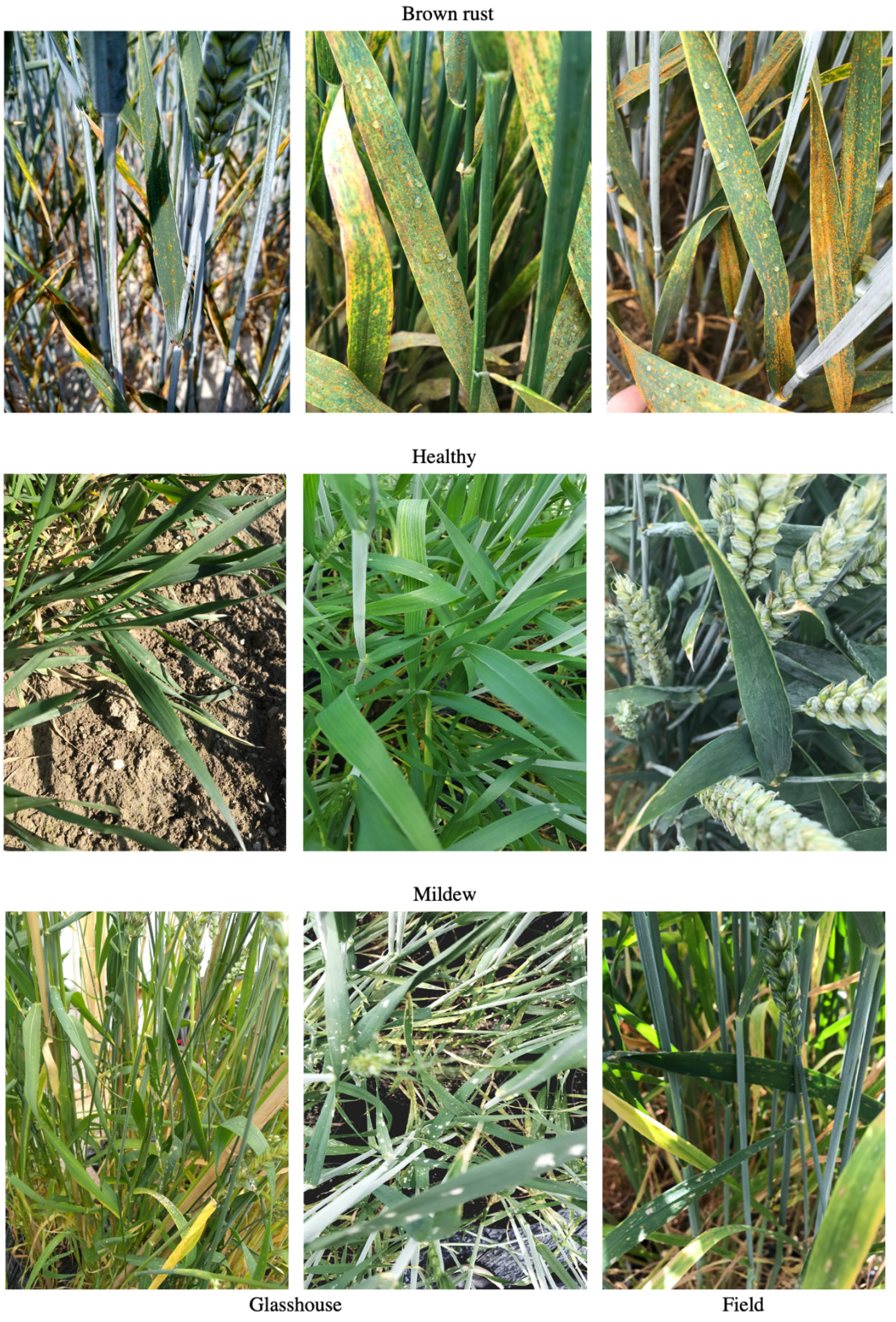


Figure 3.2 Example images from our dataset showing some different conditions and levels of infection. (Part 1)

Figure 3.3: Example images from our dataset showing some different conditions and levels of infection. (Part 2)

Table 3.5: Distribution of images across all five categories for the full dataset, and the three subsets: train, validation, and test.

Category	**Full Dataset**	**Train Set**	**Validation Set**	**Test Set**
Brown Rust	2503	1501	501	501
Healthy	2274	1364	455	455
Mildew	2943	1765	589	589
Septoria	7063	4237	1413	1413
Yellow Rust	4389	2633	878	878
TOTAL	**19172**	**11500**	**3836**	**3836**

3.2.2 Pre-trained models

Our first experiments used pre-trained models MobileNet, InceptionV3, VGG16 and Xception for feature extraction, before training a newly defined classifier network to classify the images in our dataset. The classifier network that we used had the same architecture for each of the pre-trained models. It consisted of one fully connected layer with 256 neurons, dropout (Srivastava *et al*., 2014) and the fully connected output later which provided the predictions. We used a learning rate of $1x10^{-4}$ and a batch size of 128. Table 3.6 provides individual information for each of the four models about the input image sizes and number of training epochs we used for training on all available data prior to testing.

Table 3.6: Input sizes and number of training epochs used for each of the four pre-trained networks

Models used	**Input image size for feature extraction (pixels)**	**Number of training epochs**
InceptionV3	600 x 600	20
MobileNet	532 x 532	15
VGG16	532 x 532	30
Xception	600 x 600	15

Table 3.7: Classification accuracy and F1 score for each of the pre-trained networks

Models Used	**Classification accuracy on test dataset**	**F1 Score**
MobileNet	91.46%	0.90
InceptionV3	91.41%	0.91
VGG16	85.16%	0.83
Xception	89.87%	0.89

Table 3.7 shows the classification accuracies and F1 scores for each of the pre-trained models used for this experiment. These results are significantly higher than our weighted random guess and zero rule baselines, proving that deep learning models are capable of handling complex data, such as the images in our dataset. Following this, we decided to experiment with our own model architectures with the aim of finding the network that provides the highest classification accuracy.

3.2.3 Training new model architectures

We did a collection of experiments with different numbers and types of layers. First, we started by using the train and validation step to try different architectures and hyperparameters, then the models which seemed to perform best on the validation set were evaluated on the test set. Table 3.8 shows a summary of our experiments and the different hyperparameter values used, the highlighted rows show the models which were sent for testing. All initial training was conducted using the HPC cluster, not the GPU nodes.

Table 3.8: Summary of network experiments, validation accuracy results and approximate train time

Model	Input size	Batch size	Epochs	Learning rate	Approx. train time	Peak validation acc.
1.1	(256,256)	32	30	1.00E-05	1 day	76.92%
1.2	(512,512)	128	30	1.00E-05	17 days	78.90%
2.1	(256,256)	128	50	1.00E-04	6 days	91.79%
2.2	(512,512)	128	50	1.00E-04	18 days	92.26%
2.3	(256,256)	128	75	1.00E-04	8 days	93.12%
2.4	(256,256)	8	75	1.00E-04	8 days	96.76%
2.5	(256,256)	128	125	1.00E-04	12 days	94.84%
2.6	(512,512)	32	50	1.00E-04	16 days	97.55%
2.7	(512,512)	32	20	1.00E-04	9 days	96.75%
2.8	(256,256)	8	200	1.00E-04	28 days	97.11%
2.9	(512,512)	32	100	1.00E-04	38 days	98.04%
2.10	(256,256)	8	400	1.00E-04	60 days	97.55%
3.1	(512,512)	8	20	1.00E-04	9 days	95.31%
3.2	(512,512)	8	48	1.00E-04	30 days	96.71%
4.1	(512,512)	8	20	1.00E-04	21 days	84.30%
res1.1	(160,160)	128	50	1.00E-04	4 days	76.17%
res1.2	(512,512)	64	30	1.00E-04	7 days	80.26%
res2.1	(512,512)	64	30	1.00E-04	7 days	85.79%
sep1.1	(128,128)	128	50	1.00E-04	4 days	72.07%
sep1.2	(512,512)	128	30	1.00E-04	8 days	76.17%

We used our knowledge of previous image classification experiments, such as those found in 'Deep Learning with Python' (Chollet, 2017a), coupled with the available information about the pre-trained models to determine a sensible starting point for our experiments. The pre-trained models ranged from 16 to 71 layers deep, so we decided to test different numbers of layers, starting small, to find the optimal number. We chose to start small with the aim of keeping training time as short as possible. In general, models with more layers take longer to train than those with fewer layers.

3.2.3.1 Model 1

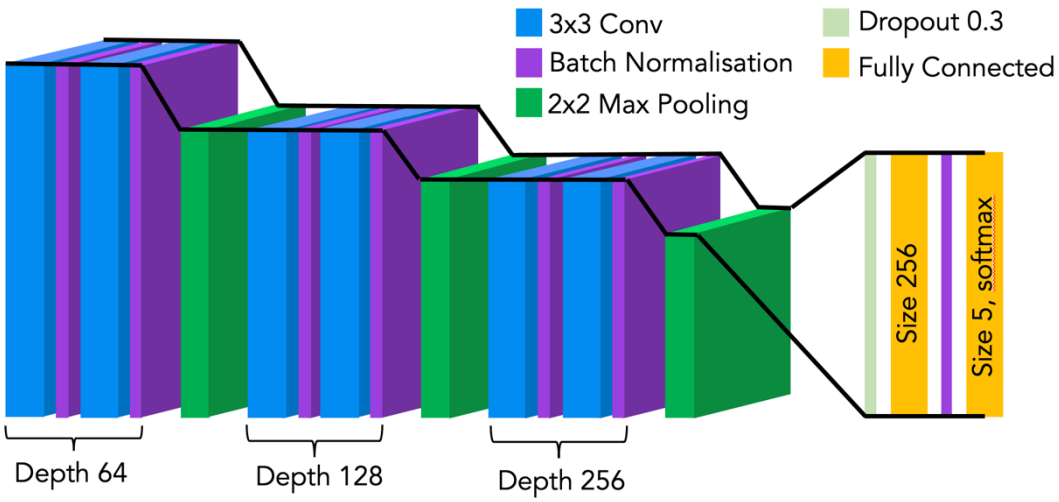


Figure 3.4: A visual representation of the model 1 architecture.

To start our experiments, we defined a CNN consisting of 6 convolutional layers, as model 1. The layers were organised in blocks of two, with batch normalisation following each and the blocks separated by a max pooling layer. We included dropout of 0.3. A visual representation of the model architecture is shown in Figure 3.4. We trained two instances of this network architecture, the first, model 1.1, with input image size (256,256) and batch size 32, and the second, model 1.2, with input image size (512,512) and batch size 128. The learning rate was set at $1x10^{-5}$. The first model ran significantly quicker than the second, where the training accuracy climbed in steps, see Figure 3.5. To a certain extent, the quicker run time was expected, however as seen in Table 3.8, the difference is quite extreme ranging from

1 day for model 1.1 to 17 days for model 1.2, We are uncertain what caused the significant change in training time or the step climbing nature of the validation accuracy, however as the second model yielded better results (Figure 3.6) we decided not to pursue model 1.1 any further.

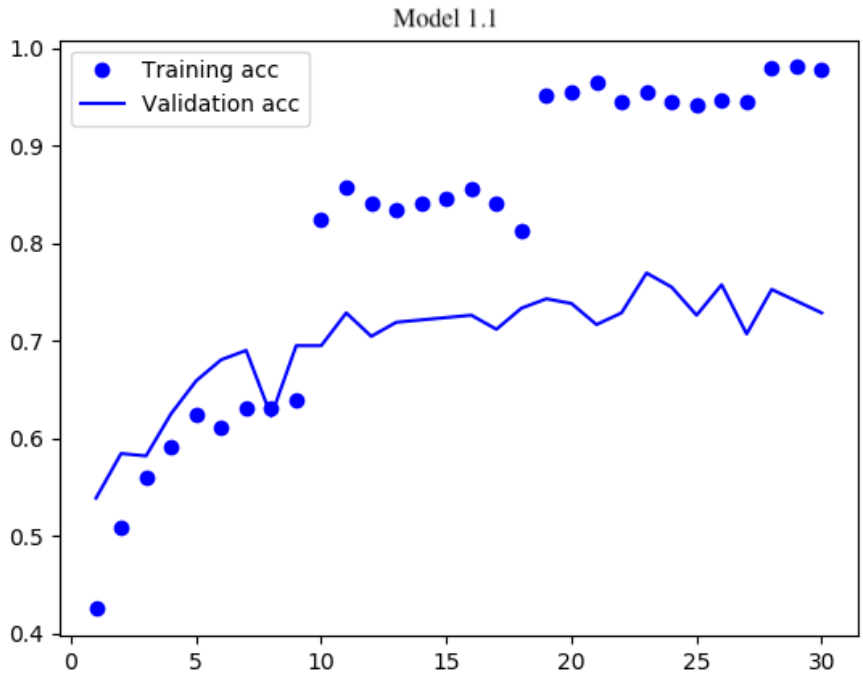


Figure 3.5: Training and validation accuracy plot for model 1.1

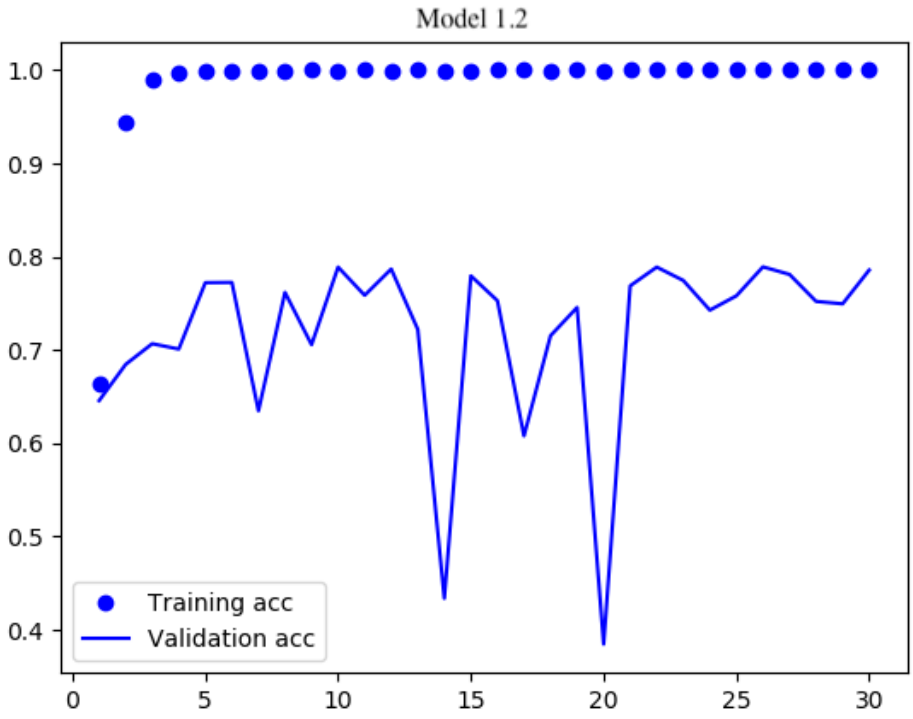


Figure 3.6: Training and validation accuracy plot for model 1.2

3.2.3.2 Model 2

As the validation accuracies did not seem to be reaching the same levels as the pre-trained models did, we decided to try a deeper architecture. The architecture, model 2, consisted of 13 convolutional layers, again each followed by batch normalisation. The layers were separated into blocks by max pooling layers, two blocks of two convolutional layers followed by three blocks of three convolutional layers. This time dropout was set at 0.5 and the learning rate was $1x10^{-4}$. A visual representation of the model 2 architecture is shown in Figure 3.7. With this architecture, 10 experiments were conducted, models 2.1 – 2.10, where the batch size, input size and number of training epochs were changed. Figure 3.8, Figure 3.9 and Figure 3.10 show the training and validation accuracy and loss plots for the three experiments using this architecture which we decided to evaluate on the test set (models 2.4, 2.6 and 2.8). In each case the validation accuracy reaches over 96%, which is

significantly higher than for the first model architecture. As such, neither model 1.1 or 1.2 were sent to test.

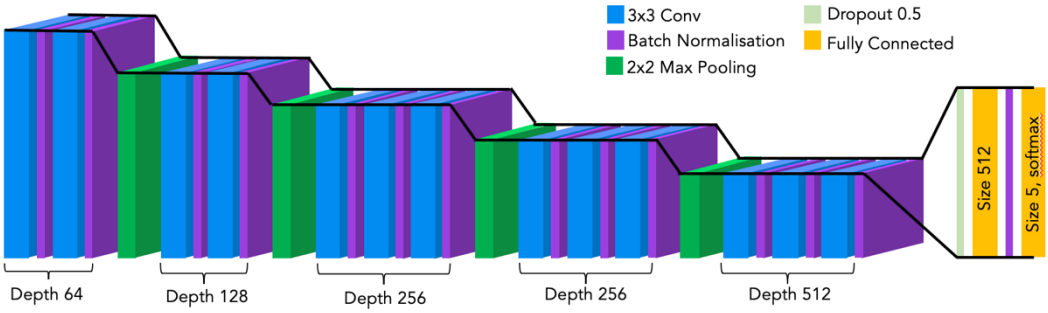
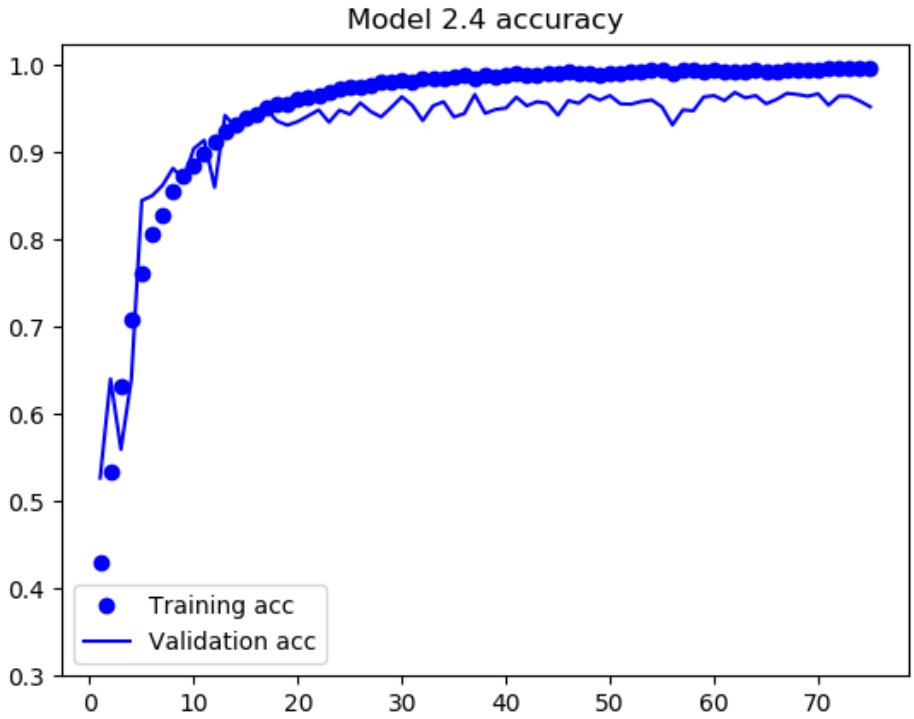


Figure 3.7: Visual representation of the model 2 architecture.

The results in Table 3.8 show three other instances for the second model architecture where the validation accuracy climbed above 96%. Model 2.7 was not sent for evaluation because the validation accuracy still appeared to be climbing, see Figure 3.11. Instead, a new version of this model, with the same hyperparameters, was sent to train for 100 epochs instead of 20 (model 2.9). This yielded the highest validation accuracy of all our experiments but took over a month to train. At this point we had already finished evaluating all three of the other models and continued with further experiments detailed in this thesis, so decided not to invest the extra time with this model.

The other model which had an accuracy over 96% which we decided not to test was 2.10. Although one of the highest validation accuracies, this model was just a version of model 2.4 but trained for 400 epochs instead of 75. This was done because there was a slight slope in the validation results for 2.4 and we wanted to see whether it would climb much further with extra training time. We concluded that the validation results were not significant enough to warrant the two months of training time required, and therefore did not pursue this model further.

a)



b)

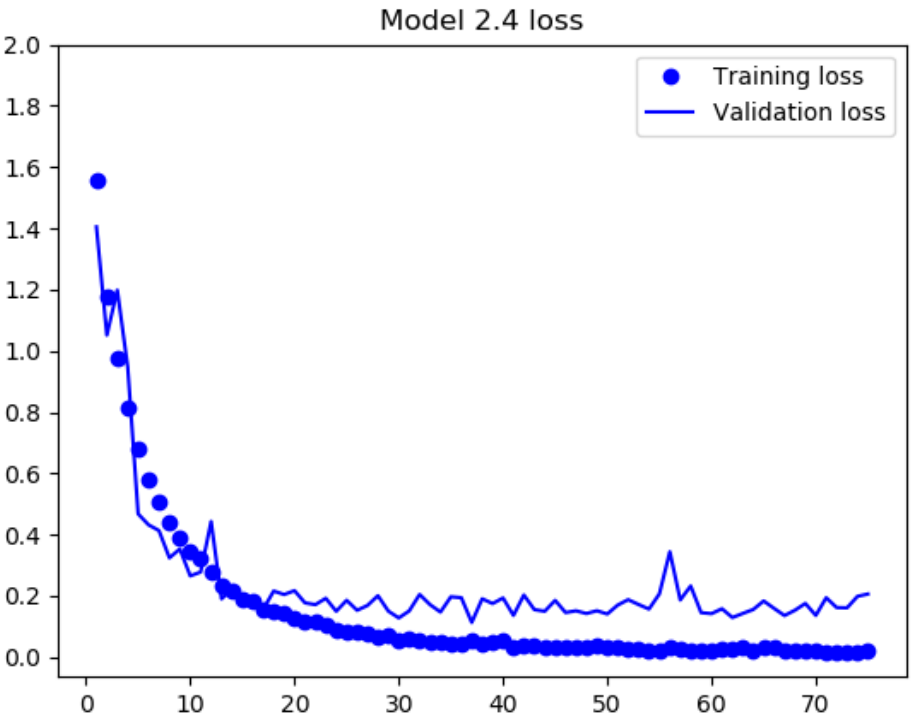
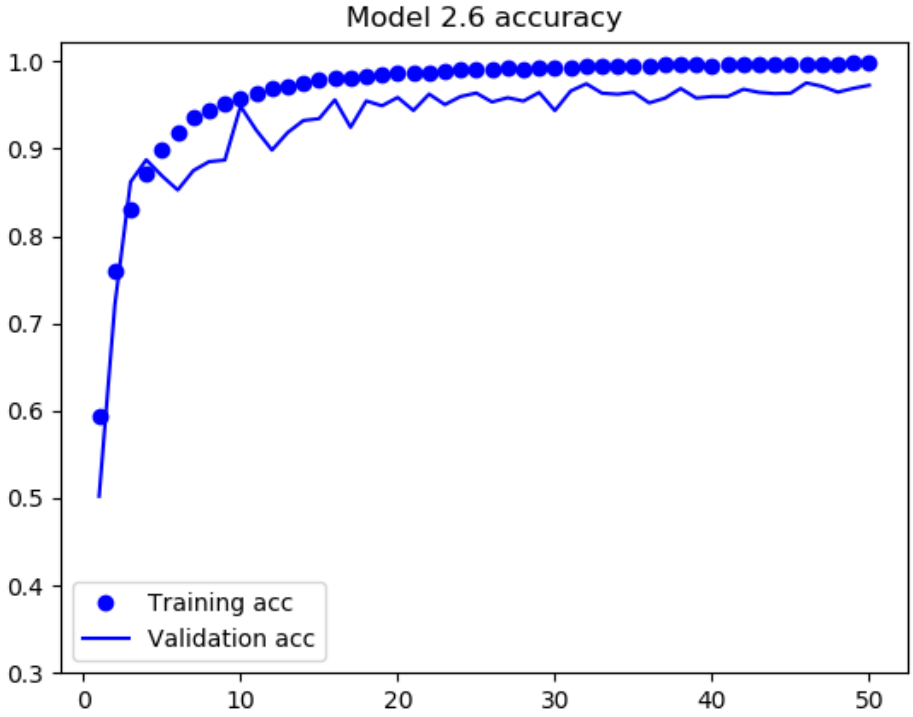


Figure 3.8: Training and validation accuracy (a) and loss (b) plots for model 2.4

a)



b)

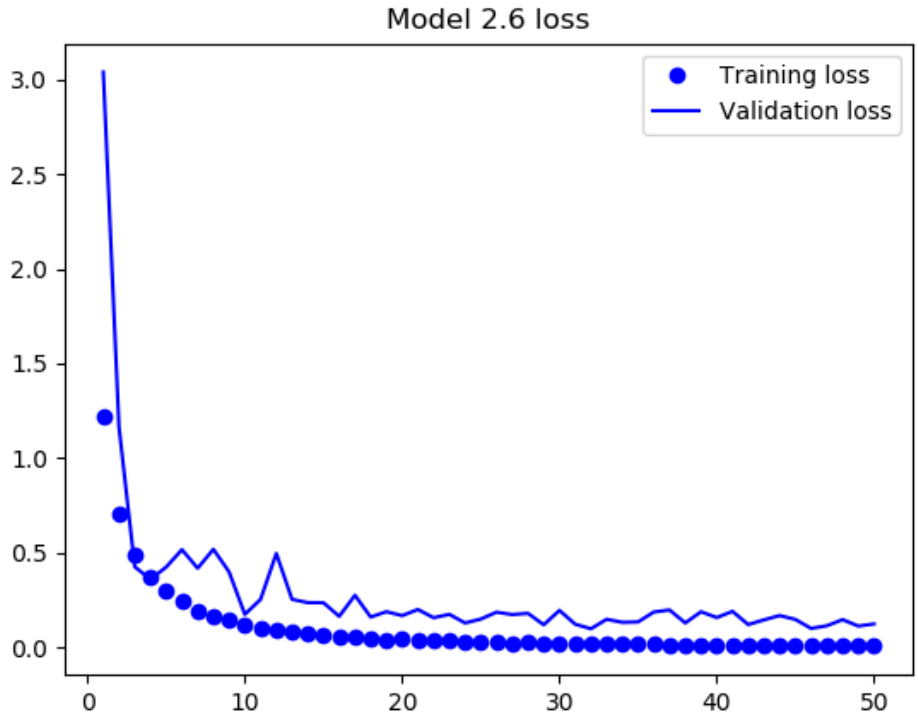
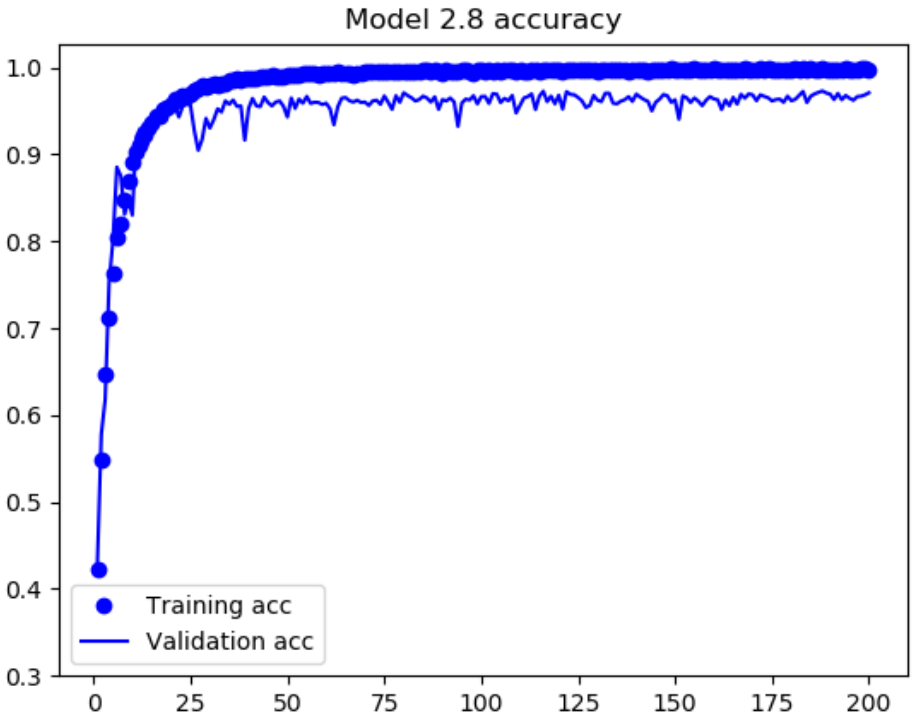


Figure 3.9: Training and validation accuracy (a) and loss (b) plots for model 2.6

a)



b)

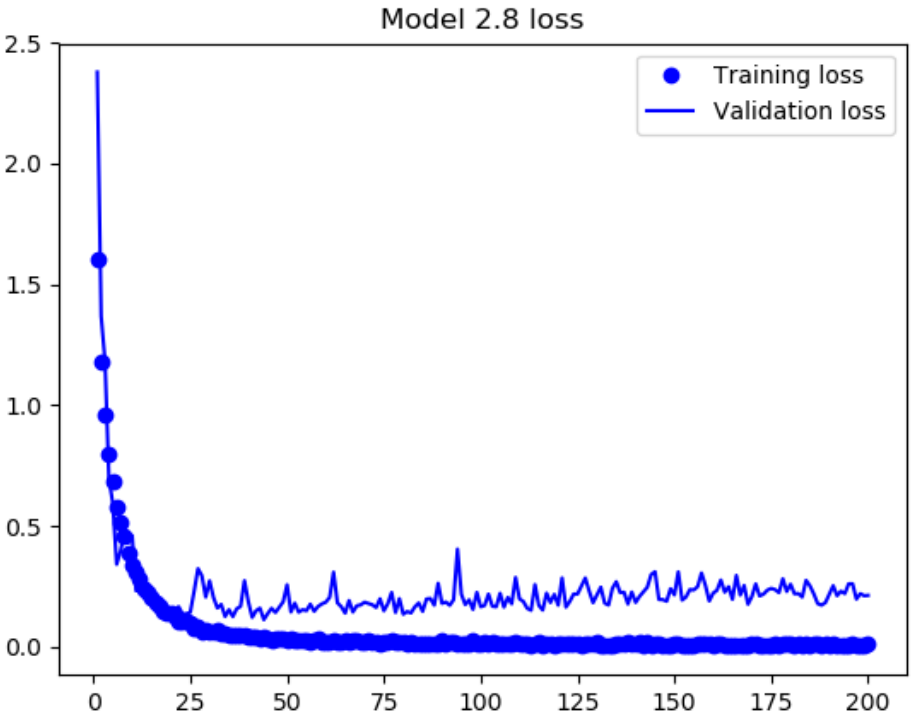


Figure 3.10: Training and validation accuracy (a) and loss (b) plots for model 2.8

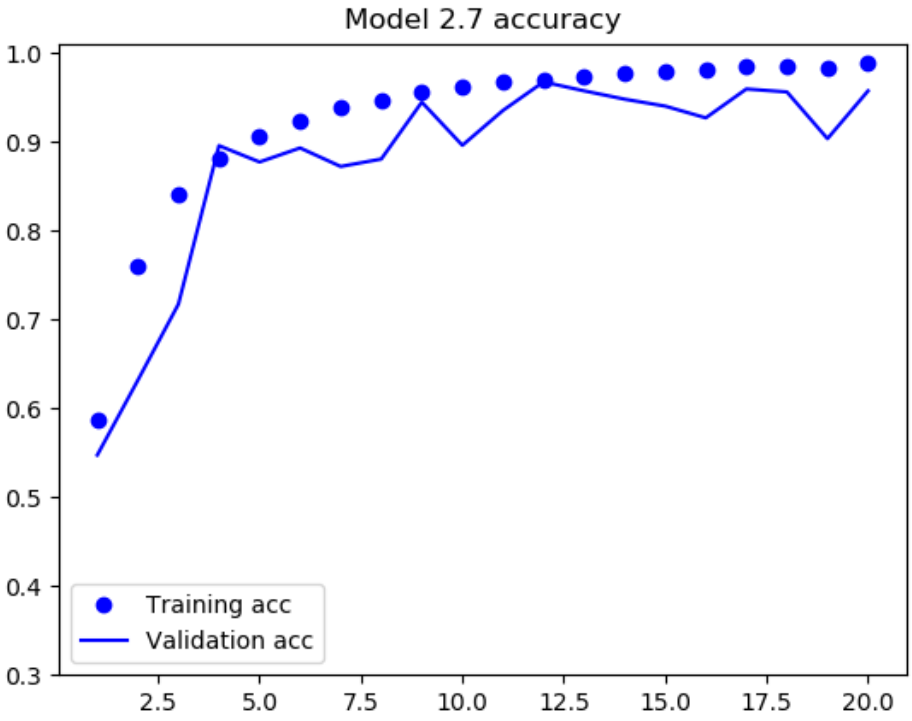


Figure 3.11: The training and validation accuracies for model 2.7. Here both accuracies still appear to be climbing at the end of training.

3.2.3.3 Model 3

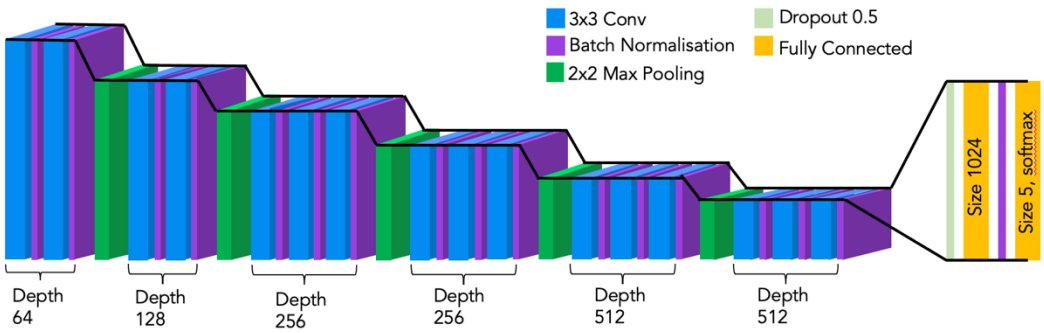


Figure 3.12: Visual representation of the model 3 architecture.

Another CNN was defined to test whether an even deeper model would be able to improve on the results of the previous architecture and if so, would it be significant enough improvement to justify the added computational cost. The third model,

model 3, architecture added a fourth block of three convolutional layers to the end of the second model, this new architecture can be seen in Figure 3.12. With this architecture, two experiments were conducted with different numbers of training epochs. Both experiments took the larger (512, 512) input image size. Initially, for model 3.1, we tried 20 epochs, however it was clear from Figure 3.13 that both the validation and training accuracy were still climbing. Consequently, more training epochs were added for model 3.2. This model trained for 48 out of the 50 intended epochs before it was cancelled due to down time on the cluster. As a result, the individual results for each epoch were not stored, nor was the trained model saved. All accuracy results were collected by manually going through the results file generated during training, see Figure 3.14.

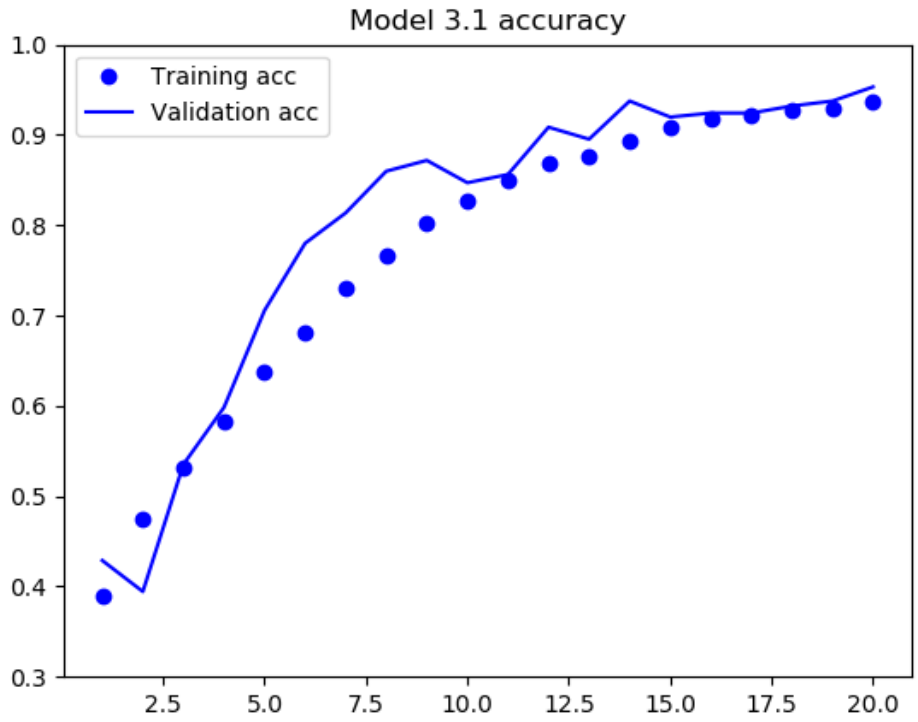


Figure 3.13: Training and validation accuracy for model 3.1. Here both accuracies are still climbing at the end of training

Again, the validation and train accuracy appear to be rising further, and it would have been possible to send a new version of the model to train for even more epochs. We decided against this due to the time taken for training. The results possibly could have gotten higher than all previous models, but we decided that the added time

investment was not worthwhile. This decision was influenced by the fact that we had already collected good results for the fully evaluated models using the shorter model 2 architecture.

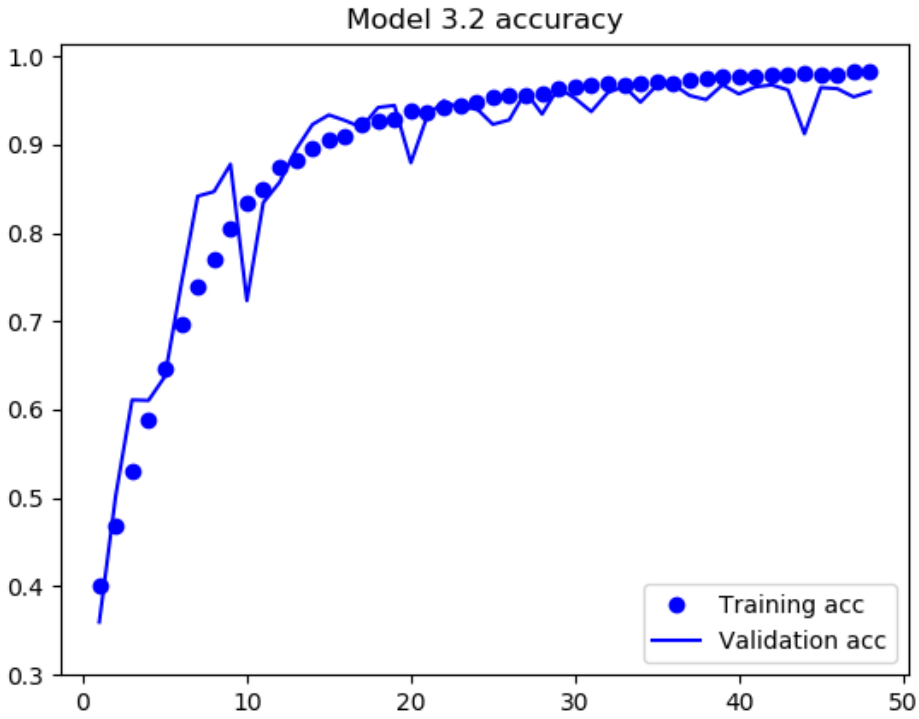


Figure 3.14: Training and validation accuracy for model 3.2

3.2.3.4 Model 4

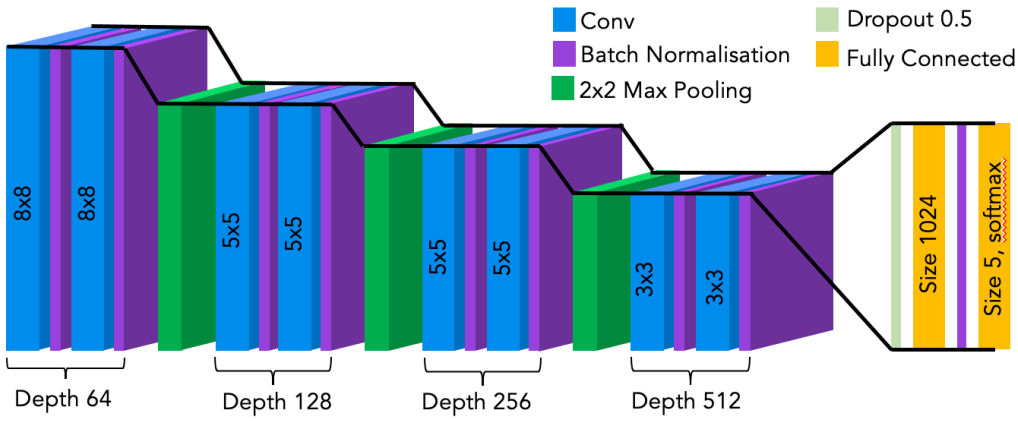


Figure 3.15: Visual representation of the model 4 architecture.

One more CNN architecture was defined, model 4, with 8 convolutional layers in four blocks of two. Max pooling, batch norm and dropout of 0.5 were included. Throughout all previous experiments, the filter size for all convolutional layers was kept at (3x3). For this model (4.1), we used different filter sizes throughout the model. In the first block the filters were (8x8). Blocks two and three had filter sizes (5x5) and the fourth block had filter size (3x3). A visual representation of the model 4 architecture can be seen in Figure 3.15. Figure 3.16 shows the validation and train accuracies for this model. It appears the model is overfitting to the train data from around epoch 10 and the validation accuracy peaks around 84% and plateaus. We conducted no further experiments with this model architecture.

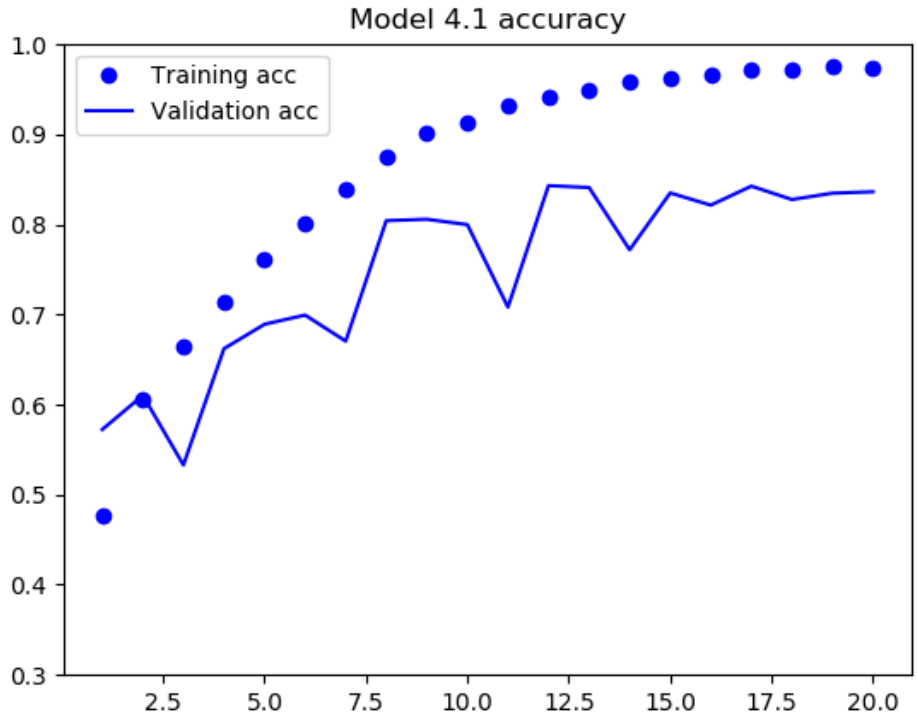


Figure 3.16: Training and validation accuracies for model 4.1

3.2.3.5 Model res1

As well as using ordinary convolutional layers in our experiments, we utilized two other layer types: depth-wise separable convolutions and residual connections. Our first experiment, res1, included three blocks of two convolutional layers with

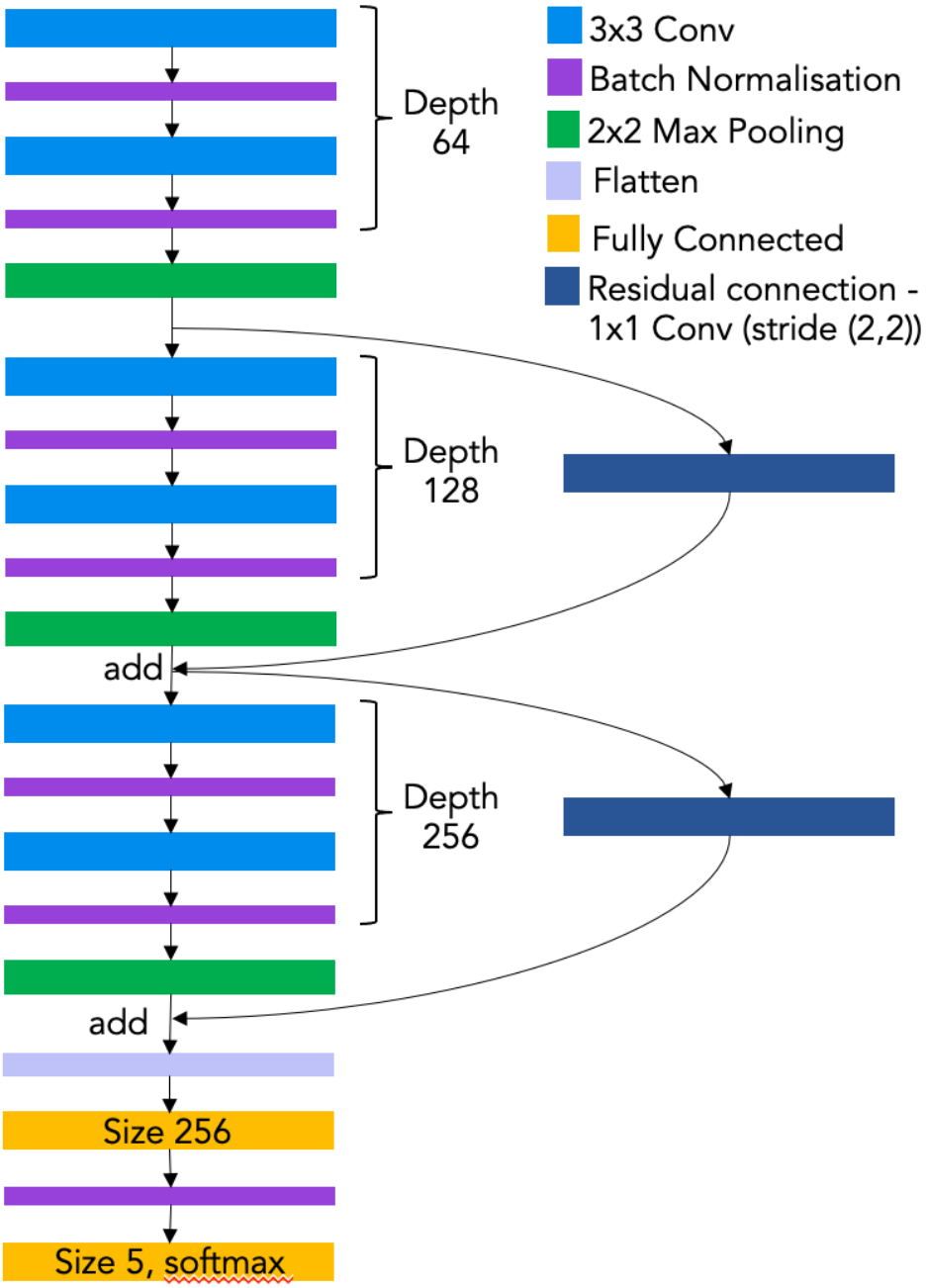


Figure 3.17: Visual representation of the res1 model architecture.

residual connections between the outputs of blocks one and two, and the outputs of blocks two and three. Max pooling and batch norm were used; however, no dropout layer was included. This was because this architecture was taken from some earlier experiments that we had conducted using the Plant Village dataset whilst learning about deep learning and we had not included dropout at that point. Figure 3.17 shows a visual representation of the res1 model architecture. We trained two models

res1.1 and res1.2 with different input and batch sizes. In both cases, the model appears to overfit almost immediately, see Figure 3.19 and Figure 3.18.

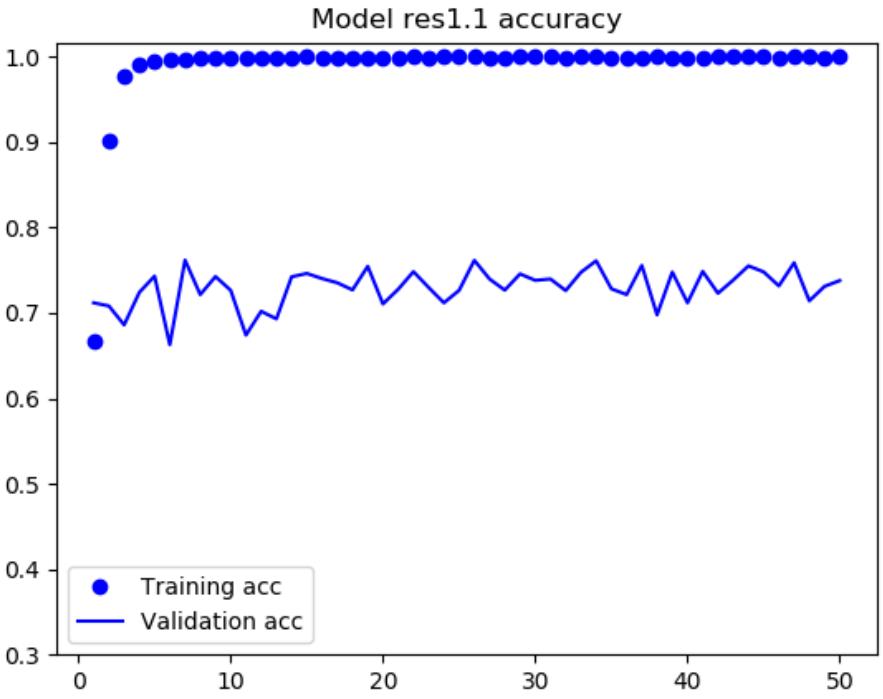


Figure 3.19: Training and validation accuracies for model res1.1

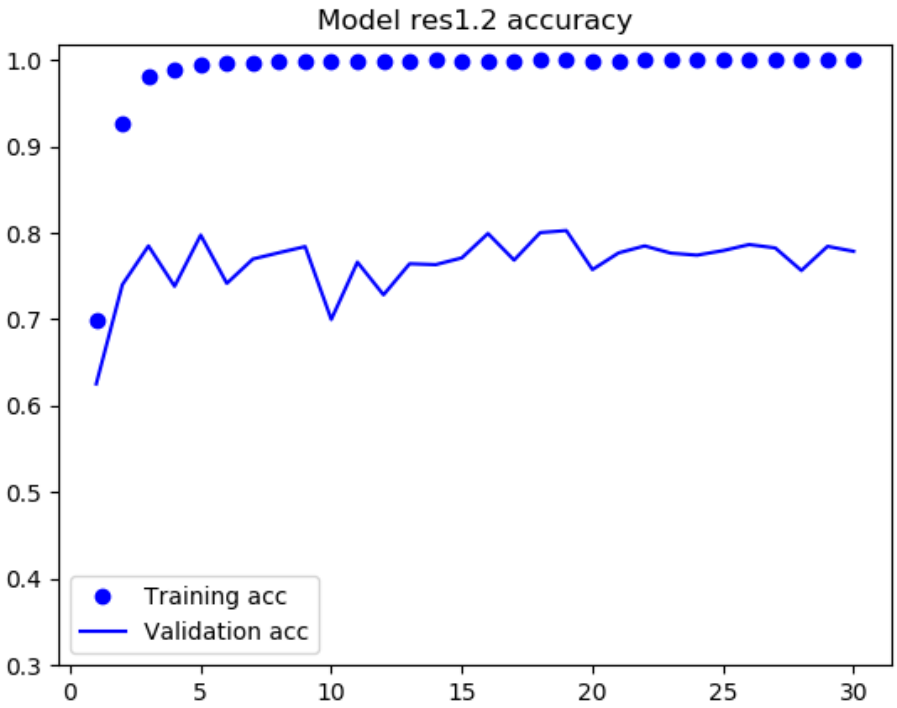


Figure 3.18: Training and validation accuracies for model res1.2

3.2.3.6 Model res2

A second attempt at a residual model, res2, added a fourth block of convolutional layers and a further residual connection between the outputs of blocks three and four. A visual representation of the res2 model architecture can be seen in Figure 3.20. We theorised that a deeper model may be better equipped to handle our data. **Error! Reference source not found.** shows that this is not the case as the model overfits in the same way as res1.1 and res1.2.

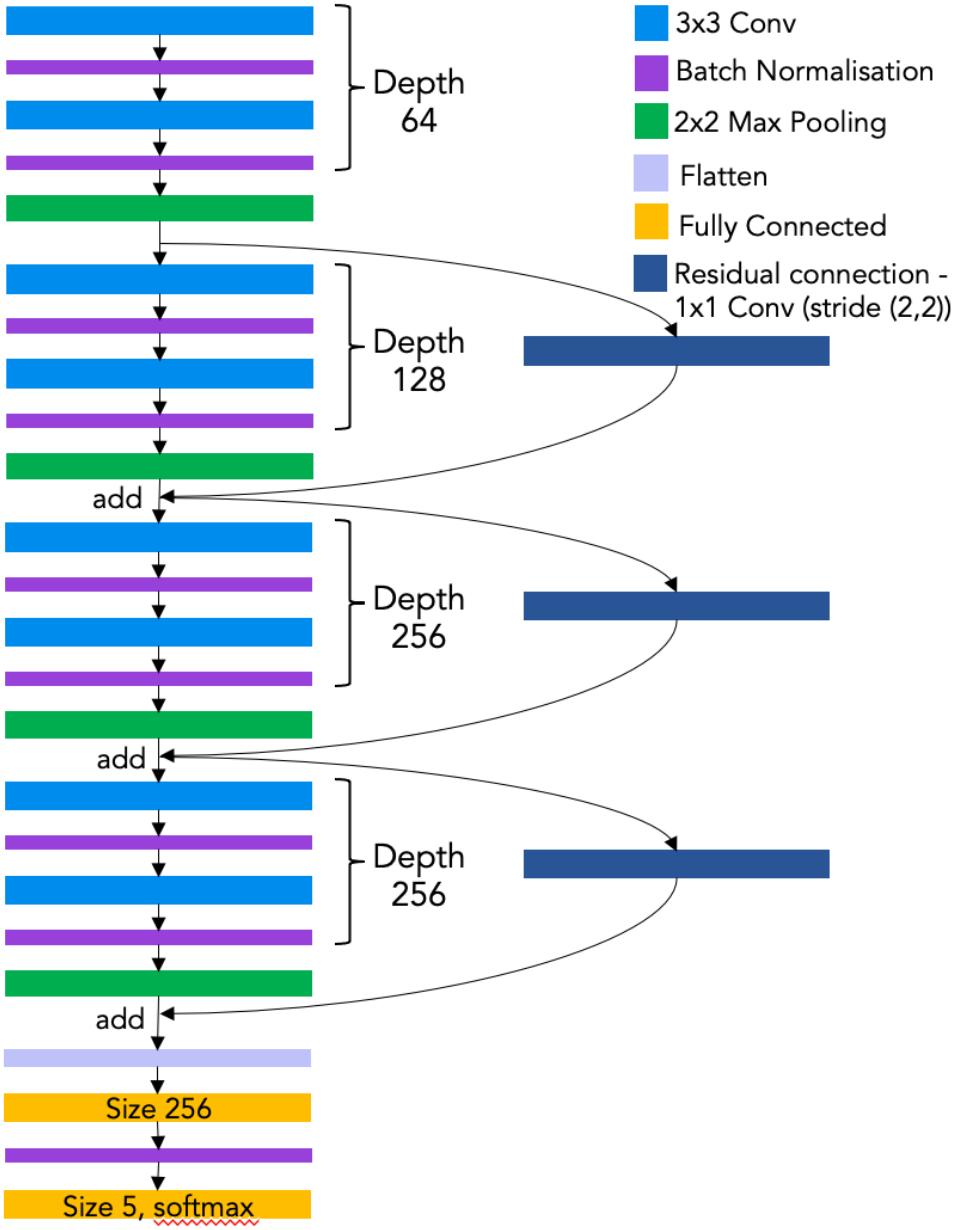


Figure 3.20: Visual representation of the res2 model architecture.

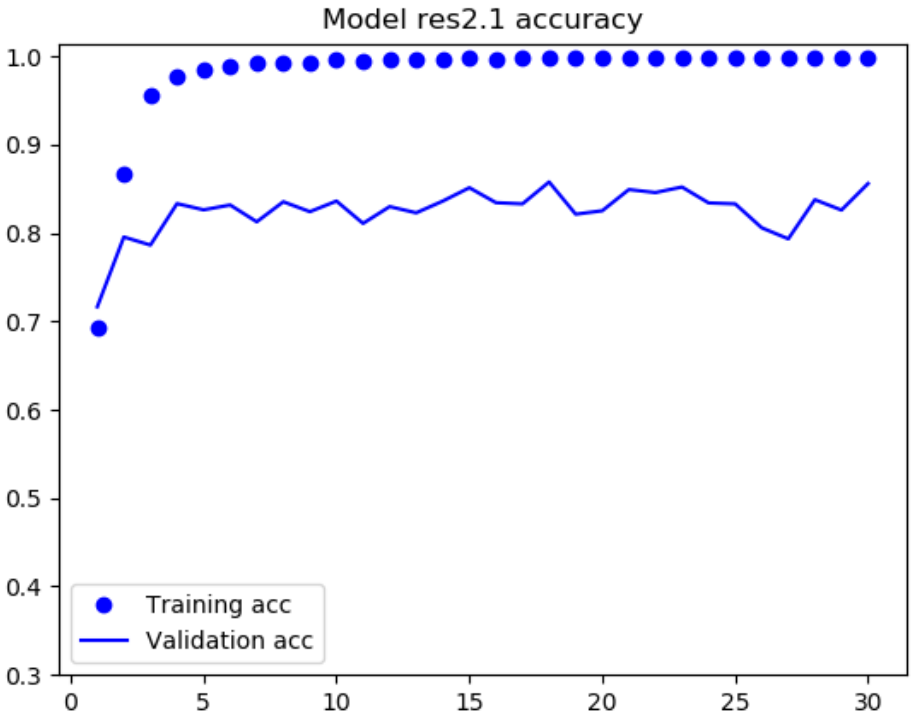


Figure 3.21: Training and validation accuracies for model res2.1

3.2.3.7 Model sep1

Our final model architecture, sep1, combined depth-wise separable convolutions and residual connections, much like Xception (Chollet, 2017b). The architecture contains a block of two convolutional layers followed by three blocks of two depthwise separable convolution layers, with a residual connection between the outputs of blocks two and three. Again, max pooling and batch norm were used without dropout. Figure 3.23 shows a visual representation of the sep1 architecture. Sep1.1 and sep1.2 were trained using different input image sizes and different numbers of epochs. Much like the residual models, overfitting occurred very early on, with the validation accuracy plateauing much lower than the training accuracy, see Figure 3.24 and Figure 3.25 .

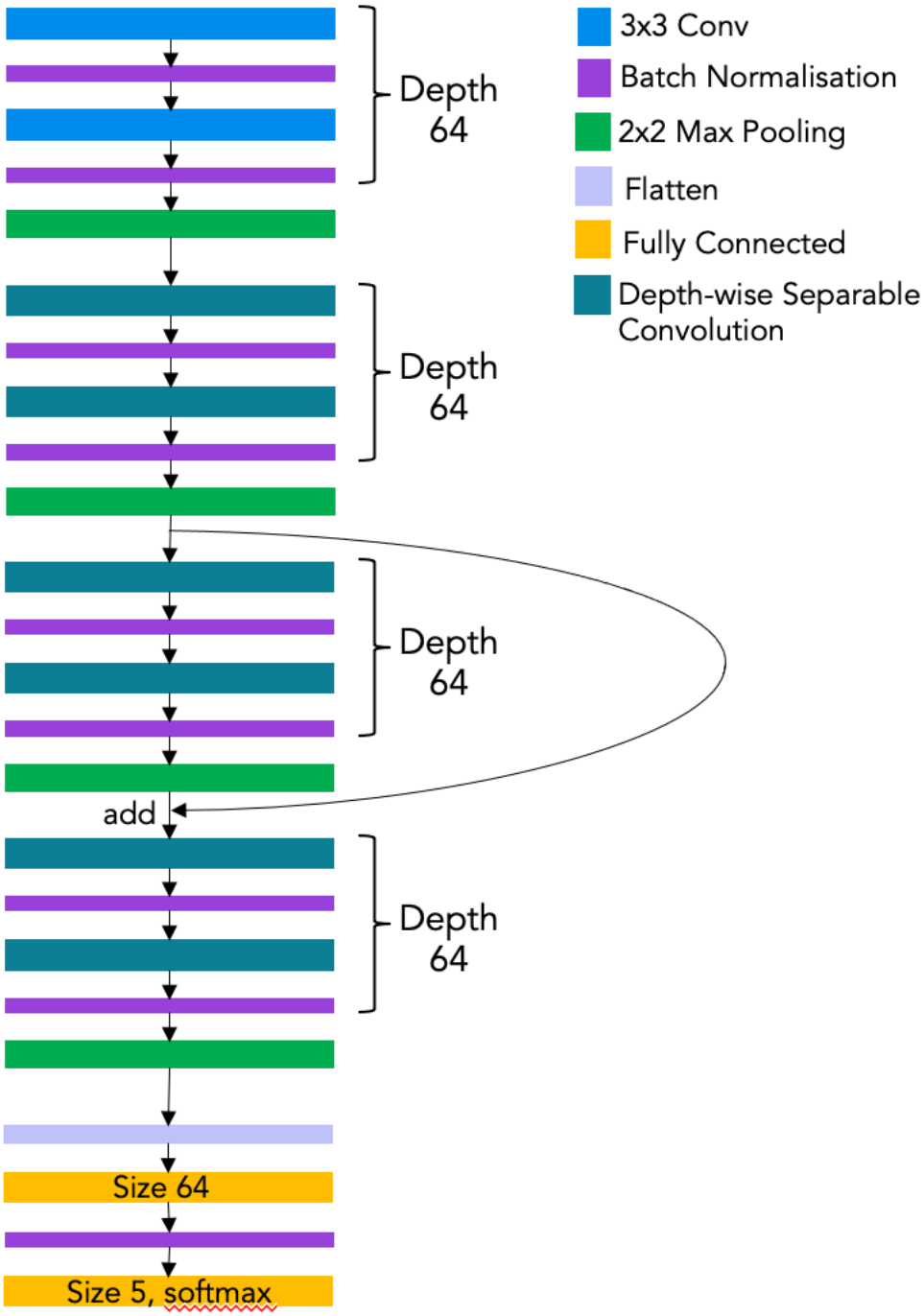


Figure 3.22: Visual representation of the sep1 model architecture.

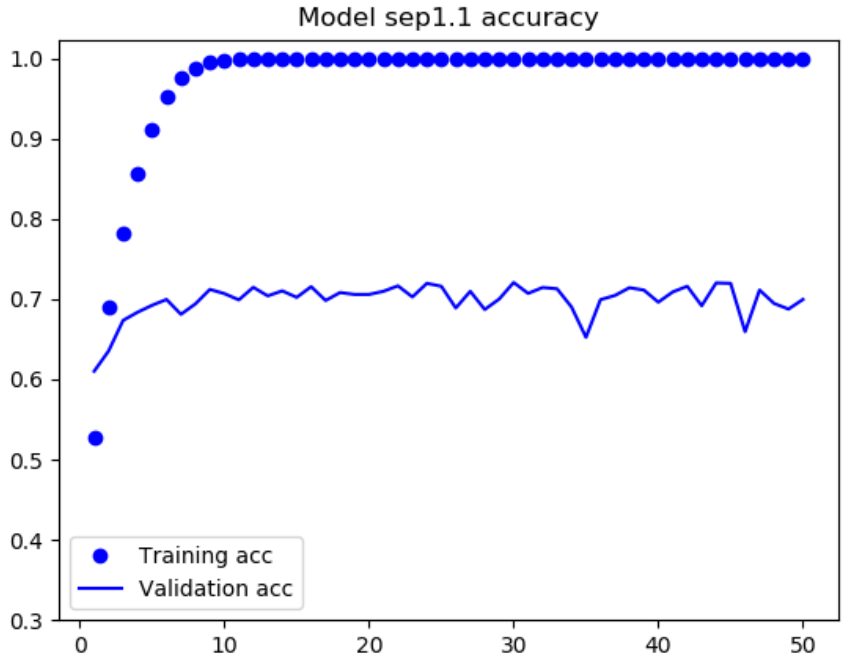


Figure 3.23: Training and validation accuracies for model sep1.1

Figure 3.24: Training and validation accuracies for model sep1.2

3.3.4 Evaluation of models

After all our experiments with different architectures and hyperparameter values, three models were chosen to be evaluated. All were variations of the model 2 architecture. Two models, 2.4 and 2.8, used input size (256, 256) and batch size 8, but were retrained on all available train data prior to testing for 75 and 200 epochs respectively. The other, 2.6, had input size (512, 512) and batch size 32, and was trained for 50 epochs. When this work was carried out, we were able to complete the evaluation process for model 2.4 using the GPU node on the cluster, so gaining results in 7 days. The other models were evaluated using the ordinary cluster nodes due to memory availability at the time. The final accuracy and loss results for the fully trained models can be seen in Table 3.9.

Table 3.9: The final classification accuracies and loss scores for the three fully evaluated models

Model	**Final classification accuracy**	**Final loss**
2.4	97.05%	0.122
2.6	97.31%	0.120
2.8	97.34%	0.213

Due to the nature of our experimentation, where the choice of hyperparameters was influenced by the results of previous experiments, the evaluation process took place at different times for these models. Consequently, we gained results for model 2.4 over a month before any other model. Whilst the others were still training and evaluating, we conducted further experiments detailed in this thesis using model 2.4 and so it is the model we will focus on for the rest of this thesis.

In Figure 3.26, the confusion matrix shows that model 2.4 performs consistently well across all categories, with 94% classification accuracy or above. Where there are misclassifications, Septoria is often the predicted category. This is to be expected due there being more Septoria data in the dataset, therefore the potential for some

bias. However, this does not seem to be affecting the model too much as the results are still consistently high.

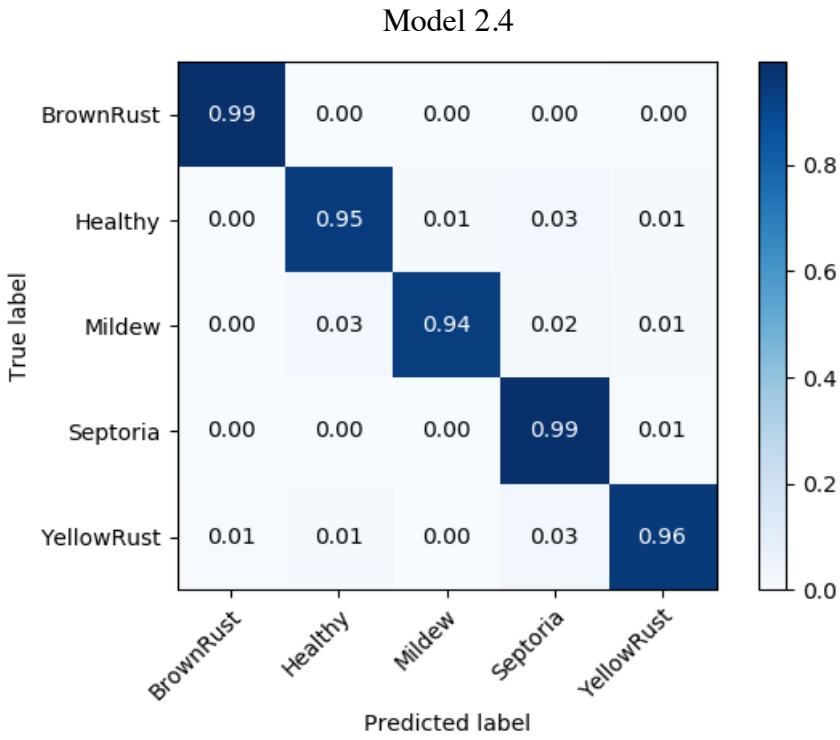


Figure 3.25: Confusion matrix of model 2.4's predictions on the test data set

3.2.5 Masking images

A number of images from our test dataset were chosen from each category and masks were added to cover the important leaf and disease information by my summer student, Douglas Brown. Examples of the masked and non-masked images used in this experiment are shown in Figure 3.27. The distribution of images across each of the categories is shown in Table 3.10.

The classification results for the original versions of the images used in this experiment and the masked images are shown in the confusion matrices in Figure 3.28. There is a clear difference between the classifications of the two sets of images. As expected, the confusion matrix for the original images' echoes that of the full

dataset, where the network classifies the images with high accuracy, making only a small number of misclassifications.

Figure 3.26: Examples of the masked images (left) and their non-masked versions (right)

Table 3.10: The distribution of masked images across the five categories in the dataset

Category	**Number of masked images**
Brown rust	148
Healthy	99
Mildew	97
Septoria	175
Yellow rust	118
Total	**637**

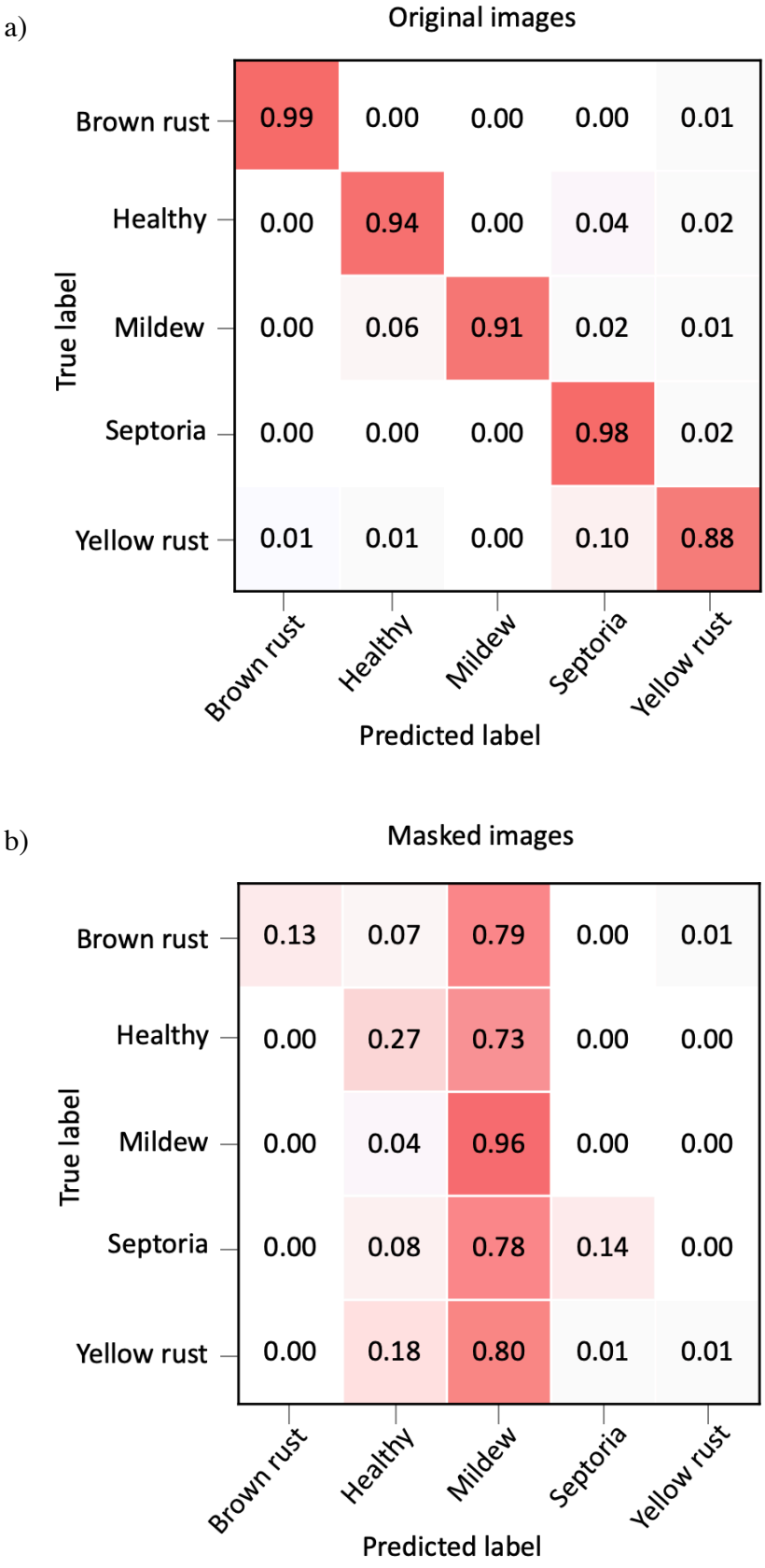


Figure 3.27: Confusion matrices for the a) the un-masked image versions and b) the masked image versions

The masked images produced different results. Here, there was a clear bias towards mildew classification over all the categories. This means that there was only a small difference between the mildew category classifications over both image sets, which implies that the model could have been using background information to classify these images. There was also a smaller tendency of incorrect classification as healthy. This was not as obvious as the bias towards mildew, but it was the only other category which was consistently picked as an incorrect classification over all categories.

3.2.6 Comparing our model against trained pathologists

To assess the performance of our deep learning model, we compared its classification performance on a portion of images from the test set against five experienced pathologists using the same subset of data. The dataset used in this experiment contained 999 images representing samples of varying levels of difficulty in classification (based on the network's performance). This dataset included 111 images that the network incorrectly classified and 888 images that were correctly classified by the network. Our model's prediction accuracy for this dataset was thus 88.88%. Table 3.11 shows the distribution of images across the categories in the smaller dataset used for this experiment.

Table 3.11: The distribution of images used to compare the model and pathologists' classifications

Category	**Number of Images**
Brown Rust	128
Healthy	122
Mildew	161
Septoria	349
Yellow Rust	239
Total	**999**

The human participants for this experiment were five expert crop pathologists, one each from the John Innes Centre, RAGT and KWS and two from Limagrain. Each participant had substantial experience in identifying and scoring these wheat diseases. Table 3.12 gives a summary of the experience and specialisations for each pathologist participant. We asked the pathologists to classify the images as individuals in order to obtain a range of results to represent the breadth of knowledge which could be expected to be found in breeding companies.

Table 3.12: The experience and specialisation of the five pathologist participants

Participant Number	**Years of Experience**	**Specialisation**
1	35	Cereal diseases, in particular mildew, yellow rust and Septoria, especially in field trials
2	40+	Cereal pathologist for major breeding company
3	20	Wheat disease observation plots, mainly for QTL mapping work
4	10	Scoring trial plots for wheat disease, in particular rusts and Septoria
5	12	Common European cereal diseases

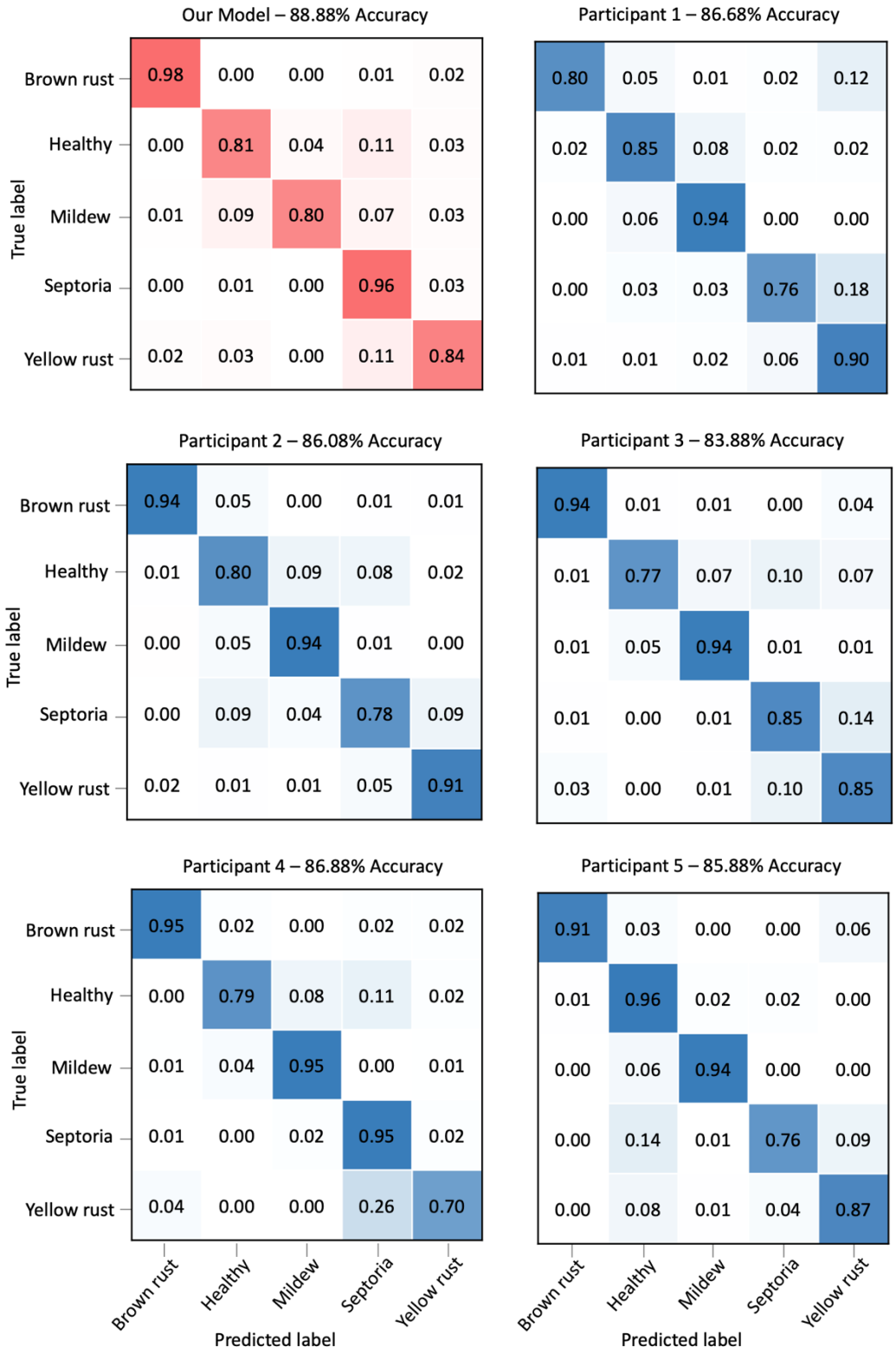


Figure 3.28: The confusion matrices and classification accuracy results for our model and the five pathologist participants

The confusion matrices in Figure 3.29 show the results of the trained network compared with the results of the five pathologist participants. Our network classified each category with an accuracy of 80% or higher, with mildew being the category with the most incorrect classifications. In contrast to this, each of the five participants classified mildew with 94-95% accuracy, making it one of the highest accuracy classes. Of the 999 images, 643 images were correctly classified by all five participants.

Another difference between the network's results and the pathologists was in the classification of the Septoria category. Here, the network performs extremely well, gaining 96% accuracy, but all but one of the pathologists were less accurate, with a range of 76-85%. For these four pathologists, the main source of incorrect classification was yellow rust, with healthy also being significantly misclassified for two of the four. The pathologist who performed similarly to the network on the Septoria category instead showed a dip in classification accuracy in the yellow rust category. Here, almost all the misclassifications were Septoria.

In a second experiment, the same five pathologists were presented the images again, but this time they classified them together as a group. For this, we removed 643 that all five participants correctly classified in the first experiment, leaving 356 to be re-classified together. In this experiment the group was allowed to disregard any image that they could not agree on a classification for. This happened due to possible misclassifications in the dataset itself, or the presence of multiple diseases that was missed during quality control.

Of the 356 images that the group were presented with, there were 46 which the group could not decide on a classification for and so were disregarded for assessment later. Of the remaining 310 images, the group correctly classified 265. Assuming that the 643 images which were correctly classified by all in the original experiment would be correctly classified by the group, that meant that all together the group correctly classified 908 images. Removing the 46 disregarded images, the total number classified was 953, thus giving a classification accuracy score for the group of 95.28%.

Of the images that were disregarded, none were in the healthy category. This was not surprising due to the clear differences between a healthy and diseased leaf. Only one image was in the mildew category, and it was a very light infection, so the discussion was between whether to classify as mildew or healthy. The remainder of the images were brown rust, yellow rust and Septoria. In all cases the discussion was whether there was a misclassification in the image, or whether there were multiple diseases present in the image. This highlighted potential issues in a few of the images contained within our dataset, which likely arose from human error during the initial labelling and quality control stages of data collection.

3.3 Discussion

In this chapter we aimed to evaluate the viability of deep learning models, specifically convolutional neural networks, for identifying wheat diseases from images taken in realistic growth conditions. In recent years, datasets acquired in the field have become more widely used (Li, Zhang and Wang, 2021). Our dataset includes images that capture the complex conditions found typically in real growth situations and represent typical examples of the kind of images that automated disease scoring in realistic situations would need to be able to deal with. Several studies pre-processed their images to remove complex background information (Barbedo, 2018), or did not include background information in their images at all (Mohanty, Hughes and Salathé, 2016; Liu *et al.*, 2018). We aimed to use our dataset as is, without removing the background information, so that there would not need to be any extra pre-processing steps when deployed in the field.

In many cases, the datasets collected and used for plant disease detection with deep learning are still significantly smaller than those used in other cases (such as the ImageNet dataset) (Liu and Wang, 2021). Our dataset aimed to overcome this bottleneck for wheat disease, whilst also being as comprehensive as possible for use in the UK. Internationally, other foliar diseases are important too, and any international user of this model would need to take that into account or provide additional training data for these diseases.

Our models gained over 97% classification accuracy on new data from the test set. These results show that, with enough data, there is no need to augment or segment the images prior to training. Deep learning models are capable of handling and classifying images with complex background information.

If the results of these experiments had been poor, we would have had to look into options for enhancing our dataset to aid learning. The first method would have been to collect further data. Another option would have been to augment the data, so creating a larger dataset by rotating, mirroring, and shifting our images to create new ones. Failing that, we could have experimented with segmenting the images to remove background information or annotating our data in some way to show the model where the disease information is in each image.

To confirm that the network was performing correctly (i.e., using the leaf and disease information to drive classification), we used images in which we masked the relevant diseased areas of the plant. A comparison of the classification accuracy between the masked and unmasked images, showed a steep drop in prediction accuracy when the important parts of the image were covered for all but the mildew category. This suggests that the network was correctly identifying diseased parts of the plant for the Septoria, yellow rust, brown rust, and healthy categories. For the mildew images the prediction accuracy using masked images remained comparable, suggesting that other, non-disease related features might have been driving the classification. This is supported by our model's higher rate of misclassification of mildew images than the other diseases in the experiment with pathologist participants.

To understand this, the conditions in which the photographs of different diseases were taken should be considered. Almost all the incorrect classifications of the masked images were healthy or mildew. The misclassifications as healthy likely arose from having little disease and much greenery present in the background. The issue with the mildew images may result from them being collected predominantly in glasshouse conditions, rather than in the field. The black pots used in the glasshouse, which are present in many of the mildew images, may have caused many

of the masked images to be classified as mildew in the absence of any other disease information, because the black masks may have been mistaken for plant pots.

If this is the case, then our assumption that the other four categories are performing as they should, could be wrong. This is because the issue with the mildew could be the overpowering force for the masked images, so even if there are other misrepresentations in the data, the black masks as mildew takes precedence. Although we believe that our dataset contains a consistent range of conditions over each category other than mildew, and so should not be learning any irrelevant features, further investigation is required to clarify that the model is basing its corrections on the correct features. We discuss potential methods for testing this in section 5.2.

In our experiment to compare the model's classification power against human participants, our network outperformed each participant by at least 2%. Furthermore, the network classified the 999 images faster (approximately one hour on a Mac desktop) than the pathologists, who took close to three hours for the same dataset. The model's computation could also be sped up by using GPUs and parallelisation much more easily than it would be to recruit more trained pathologists to perform the same job.

An important thing to note here is that this classification was performed on static images. The deep learning network achieved an impressive accuracy for such data, at least comparable to that of expert crop pathologists, however, it is likely that the performance of the pathologists would increase with real plants. In a real field situation, a pathologist would be able to take a closer look, change their viewing angle, and obtain information that would normally be available to them that isn't accessible from static images. So, although this experiment is an interesting exercise to compare classifications from images, it does not fully represent the overall performance of the pathologists in practice. That being said, a significant problem with disease scoring in large trials, such as plant breeding trials, is that for trials to be scored by a person is slow and requires a lot of time of people with valuable specialist skills. The ability to move and gain other angles in the field is likely to increase the viewing time for each specimen, so this needs to be considered.

Furthermore, there is the issue of discrepancies between human labelling to consider. For example, in this experiment there were 160 cases where only one participant gave an incorrect classification. When shown the images again as a group, in many cases there was a unanimous decision on the classification of the image, indicating a human error aspect to this experiment. However, in some cases where two or more participants incorrectly classified an image and were shown it again as a group, there was still difficulty in deciding which disease was present. This presented evidence of the difficulties in distinguishing Septoria from yellow rust, and yellow rust from brown rust when certain symptoms are present.

An automated system deployed on a mobile phone will greatly increase the throughput and reduce the cost of disease scoring, making it possible to score a large trial several times. This has the potential to increase substantially the accuracy of the process through repetitive scoring. The next step would be to use deep learning methods to quantify the amount of disease present and to give a score. Whereas a classification model is useful for identifying the presence of a disease on a plot or field, a scoring model would allow breeders to quantify the spread of the disease and be hugely beneficial for developing varieties with resistance to certain diseases. We start to look at this problem in our next chapter.

Chapter 4 Quantification of Wheat Diseases

The classification of diseases is an important step towards breeding for disease resistance. The previous chapter showed how deep learning can accelerate and automate this process. The next crucial next step is to use deep learning for quantifying the amount of disease present. Breeding for disease resistance in crops currently takes multiple growth periods and a lot of hard work and input from pathologists throughout. Often, it is only possible to score each plot once during the growth period due to the time investment required and the availability of pathologists (Bird, N., KWS, personal communications). Here, we explore ways of computationally automating disease quantification. An efficient and automated quantification algorithm would allow for continuous monitoring of crops throughout the growing season and a more robust evaluation of resistance.

A deep learning model, deployed for example on a mobile device, which could score disease would allow any member of the breeding team to apply it with little to no training. This would free up the time of pathologists for other important tasks, with the added benefit of being able to track disease progression over time thanks to multiple scores.

Here, we investigate the viability of using deep learning models to aid with scoring of wheat diseases by training various model architectures using a dataset of yellow rust disease images that we collected. Due to the complexity of the problem, we experiment with different simulated images based on the same scoring system used in our real data experiments.

4.1 Collection of a wheat disease dataset for quantification

We wanted to test the ability of deep learning models for quantifying the amount of disease present as well as being able to identify and classify. This required a new dataset of images to be able to train a model. When breeders are scoring plants, they assign a score for the amount of disease present in the entire plot. Therefore, to be useful for breeders trying to score wheat plots, we needed to collect a dataset containing images of full wheat plots.

The number of volunteers and places we could photograph were limited due to various restrictions in place as a result of the global Covid19 pandemic. As a result, we had to make some decisions about what would be best to focus on for these collection periods. We decided to focus on just one disease to begin with for quantification to begin with. Yellow rust was chosen by the pathologists from the breeding companies associated with this project, Limagrain, KWS and RAGT, because they each had access to field trials which were either sprayed against other diseases or were grown in areas where other diseases were not likely to be prevalent.

The collection of images required much preparation and planning to ensure that it contained the appropriate information for quantitative scoring. Each session of photography had to take place on the same day, or as close as possible, to when the plots were scored by a pathologist. The images needed to be collected in a way that allowed us to later label the photo of the plot with the score that was given by the pathologist. Photography took place over several weeks in the summers of 2020 and 2021. It was done by me, and representatives from each of the three companies, RAGT, KWS and Limagrain.

As with the classification of wheat diseases, we wanted to include as many growth conditions as possible in our dataset, including light, weather and plant and disease life cycle stage. It was also important to capture the images in a way that would replicate how a pathologist would see the plot and give it a score. The majority of the

time, pathologists look at the top and side of the plot, although some pathologists like to part a plot to see how much disease is present within. We decided to take a mixture of images from the top of the plot and as a side angle and not to photograph a parted plot. The reason for this is that one of the aims of training a deep learning model to quantify disease is to reduce the time needed for this process. Having to part each plot to take an image would add time to the collection process.

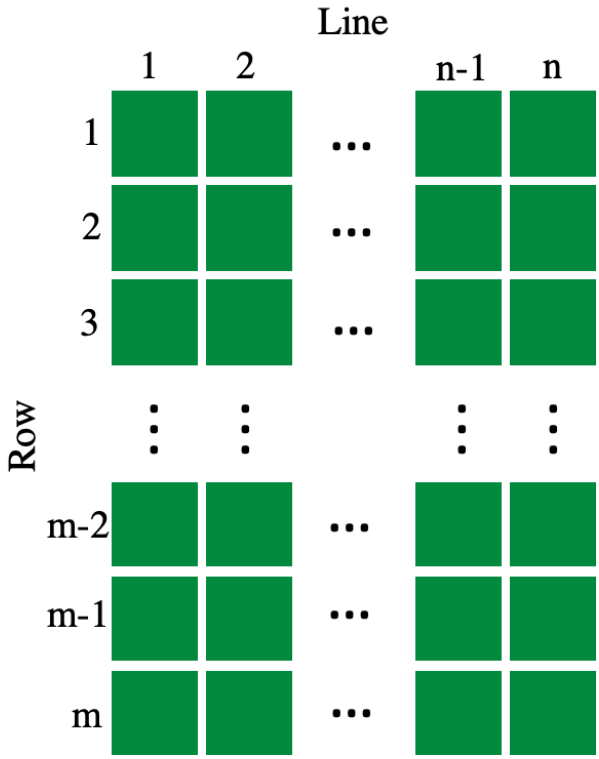


Figure 4.1: An example layout of a field trial where the plots are organised by line and row numbers

There were two methods used for collecting images over the photography period. The first method was simply taking a single image per plot, or three images where the plots were larger. Figure 4.1 depicts an example field layout for the wheat plots photographed. The layout can differ slightly depending on the company and trial type, but this model can be used as a base. The photographer would note the line number that was being photographed and take the pictures for every row in that line. A separation picture would be taken of something else to show the end of the line and make it easier to sort the images later. Once all the images were collected in this manner, they could be assigned with a score. The scores were stored in a

spreadsheet, making it easy to take the images taken for a certain line and match them to the scores for that line, see . In cases where three images were taken per plot, each line in the score spreadsheet was triplicated to allow the images to match up correctly.

a)

Image name	YR score	File name without program tag
RAGT_YellowH_CB10_above_29_05_2020_001.jpg	2	RAGT_YellowH_CB10_above_29_05_2020_001
RAGT_YellowH_CB10_above_29_05_2020_002.jpg	1	RAGT_YellowH_CB10_above_29_05_2020_002
RAGT_YellowH_CB10_above_29_05_2020_003.jpg	3	RAGT_YellowH_CB10_above_29_05_2020_003
RAGT_YellowH_CB10_above_29_05_2020_004.jpg	3	RAGT_YellowH_CB10_above_29_05_2020_004

b)

Code	Photo Date	Line	Yr Rothwell (PH) 12th June PMF	Comment	STB	PMF	Missing image STB IMAGES	PMF IMAGES
1	17th June	SKYFALL	8		x		IMG_2703	
2	17th June	LG-SKYSCRAPER	3			x		IMG_2979
3	17th June	LG-SPOTLIGHT	7		x		IMG_2704	
4	17th June	CRUSOE	0			x		IMG_2980
5	17th June	301	7		x		IMG_2705	
6	17th June	302	8			x		IMG_2981

c)

photo name		YR score
bed 71 3 per plot side view_16_06_2020_73	W1760_GALTIC	2
bed 71 3 per plot side view_16_06_2020_74	W1760_GALTIC	2
bed 71 3 per plot side view_16_06_2020_75	W1760_GALTIC	2
bed 71 3 per plot side view_16_06_2020_76	W1450_PENNANT	2
bed 71 3 per plot side view_16_06_2020_77	W1450_PENNANT	2

Figure 4.2: Segments of the spreadsheets used to align scores with image filenames for a) RAGT, b) Limagrain and c) KWS

The second method of collection used a free application called “Inventory Photos Plus”, which can be found in the google play store for android phones as “INVPHOTOPK: Inventory Photos Plus K”. From here we will refer to the application as IPP. The reason that IPP was not used by all photographer volunteers is because it was not available for use with iOS smartphones, meaning those with apple devices had to use the first method detailed above.

IPP allows the user to give photos taken a custom name with index number and saves them into a custom folder, see Figure 4.3. For our task, we were able to label images with the date taken, photography location and company, and align the photo index numbers with the plot numbers. All photos were then easily aligned with the

correct scores using the plot numbers in the score spreadsheet and the index numbers of the images.

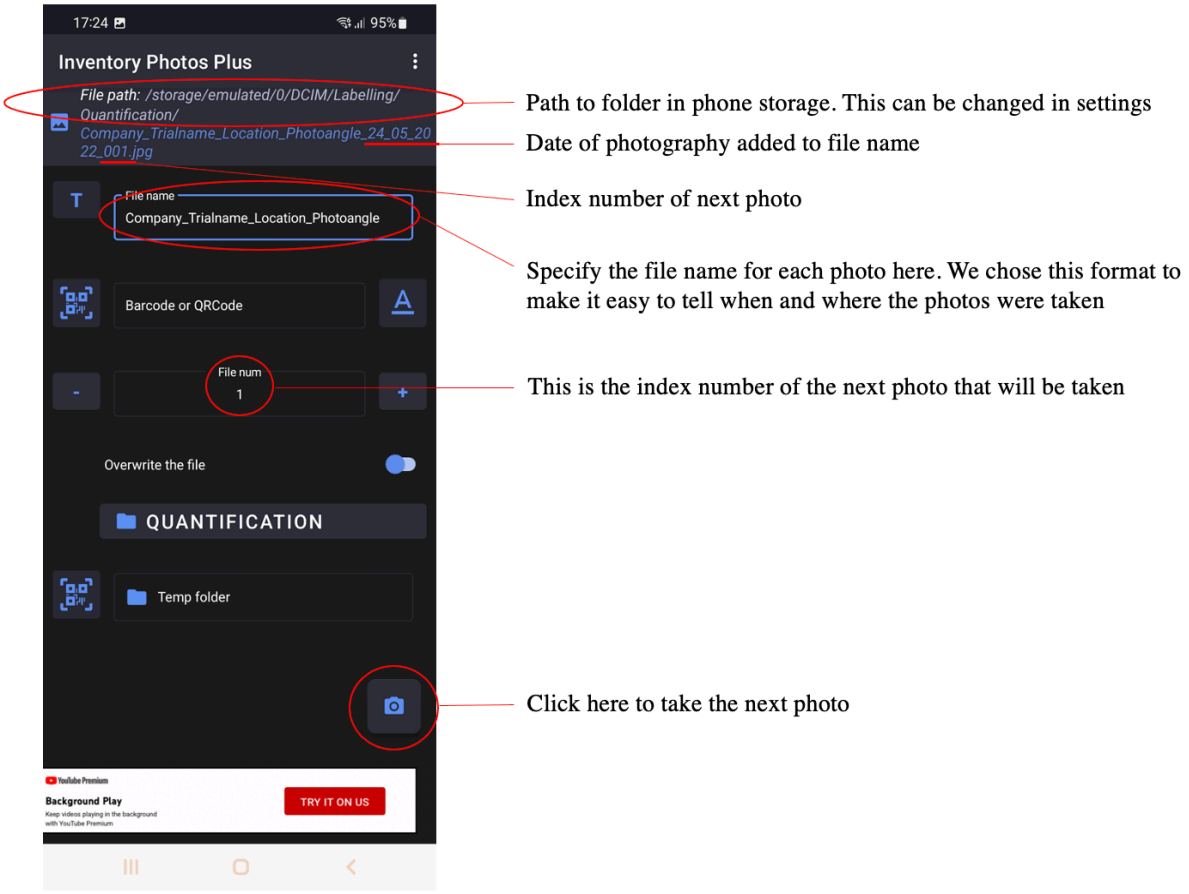


Figure 4.3: The user interface for IPP, used for collecting, labelling, and storing images

The locations for photography were dedicated yellow rust trials, which had been sprayed against other diseases. In some cases, other diseases were not fully controlled by fungicides, and so the plots were scored for these diseases as well as yellow rust. We chose to remove any images where a disease other than yellow rust had been given a score of 5 – 9. This left only images with no other disease or only small amounts of other diseases present other than yellow rust.

Over 6000 images were collected across England at breeding sites belonging to KWS, RAGT and Limagrain. Discussions with all three companies involved determined that the best scoring method to use for our datasets so that the resultant

model would be most useful all round was the NIAB 1-9 scale (see section 1.2.2). All plots that were photographed at the KWS sites were scored using this method, however plots at RAGT and Limagrain were scored using slightly different systems unique to the company.

For the images taken at Limagrain, the plots were scored by being given a percentage of infection. During image sorting and quality control, we had to convert these scores into the 1-9 categories. Table 4.1 shows the ranges of percentages that were sorted into each of the 9 score classes.

Table 4.1: The percentage ranges for sorting Limagrain images into 1-9 categories

Score	**Infection percentage range**
1	0
2	0.01 – 0.4
3	0.5 – 2
4	2.1 – 7
5	7.1 – 17
6	17.1 – 35
–	35.1 - 60
8	60.1 – 80
–	80.1 - 100

The plots that were photographed at RAGT were scored similarly to the 1-9 scale however, they included .5 categories also. Upon discussion with the pathologist from RAGT, we decided that the .5 scores would be assimilated into the score above. For example, plots with a score of 2.5 would be included in the score 3 category in our dataset.

Having matched all images with their scores, the images were sorted into folders corresponding to the score number. We then manually quality checked each folder for any images that were not acceptable for inclusion in the final dataset. Images that were removed were those that were considered too blurry to show enough information (for example, where the leaf and background information could not be distinguished between), those that were clearly not showing the same score as its classification or any that showed significant amounts of any other disease.

We chose to divide the dataset in three different ways based on the amount of data collected. The original dataset, we have called YR1, represented the optimal scenario of 9 categories. Ideally, a model would be able to learn how to classify the images into one of nine categories for each of the 9 scores. However, with the complexity of the images collected and the number of them, there may not have been enough data for the model to learn to distinguish between that many categories. For this reason, we combined the categories in two different ways as well as keeping one version of the dataset with all nine categories separate.

Having consulted with pathologists from the three companies, the first way we combined categories was to have the first six scores as separate categories, as in the first dataset. Then, due to a lower number of images in the higher categories and because a breeding line with a score of 7 or above would usually be rejected, we grouped scores 7, 8 and 9 into a single category. This dataset we have called YR2.

In the event that this would still not be enough to train a network to a high accuracy with the available data, we combined the scores one more way into a third dataset. Score 1 was kept on its own as a 'no disease' category. Scores 2 and 3 were combined to make a 'low disease' category. Scores 4 and 5 made a 'moderate disease' category and scores 6 – 9 combined to make an 'unacceptable' category. This dataset we have called YR3.

All three datasets were divided into train, validation, and test sets ready for training deep learning models. 60% of the images for each category were put into the train set, while 20% were added to each the validation and test sets.

After quality control was complete, we were left with a dataset of 5526 images. Table 4.2, Table 4.3 and Table 4.4 shows the number of images per category for the three datasets and their distribution into the train, validation, and test sets for YR1, YR2 and YR3 respectively.

Table 4.2: The distribution of images in the YR1 dataset

Score category	Total no. images	Train set	Validation set	Test set
1	1683	1010	336	337
2	844	507	168	169
3	686	412	137	137
4	556	334	111	111
5	753	452	150	151
6	530	318	106	106
7	324	195	64	65
8	115	69	23	23
9	35	21	7	7

Table 4.3: The distribution of images in the YR2 dataset

Score category	No. Images	Train set	Validation set	Test set
1	1683	1010	336	337
2	844	507	168	169
3	686	412	137	137
4	556	334	111	111
5	753	452	150	151
6	530	318	106	106
7 +	474	285	94	95

Table 4.4: The distribution of images in the YR3 dataset

Score category	No. Images	Train set	Validation set	Test set
No disease	1683	1010	336	337
Low disease	1530	918	306	306
Moderate disease	1309	786	261	262
Unacceptable	1004	603	200	201

4.2 Experimentation with deep learning models for quantification

Having collected and sorted our images into new datasets for training deep learning models to quantify the amount of disease present, we moved on to experimenting with different model architectures.

As with our classification work in chapter 3, we needed to find the baseline for our models. These baselines were different for each of the three datasets we used in these experiments. Table 4.5 a), b) and c) show the percentages of the entire dataset contained within each category for YR1, YR2 and YR3 respectively.

Table 4.5: The percentage of the entire datasets contained within each category for a) YR1, b) YR2 and c) YR3

a)

Score category	Percentage of dataset %
1	30.46
2	15.27
3	12.41
4	10.06
5	13.63
6	9.59
7	5.86
8	2.08
9	0.64

b)

Score category	Percentage of dataset %
1	30.46
2	15.27
3	12.41
4	10.06
5	13.63
6	9.59
7+	8.58

c)

Score category	Percentage of dataset %
No disease	30.46
Low disease	27.69
Moderate disease	23.69
Unacceptable	18.16

As the most common class in each of the three datasets is score 1, or no disease for YR3, the zero-rule baseline for all three datasets is 30.46%. In the case of the weighted random guess, each of the three datasets has a different baseline. For YR1 this is 17.33%, for YR2 it is 17.68% and for YR3 it is 25.86%.

The models we used for these experiments were CNN's. Different layer types and architectures were experimented with including depth-wise separable convolutional layers (Chollet, 2017b), residual connections (He *et al*., 2016) and attention mechanisms (Wang *et al*., 2017b). See chapter 2 for descriptions. The neural networks were developed using keras version 2.2.0 (Chollet and others, 2015) in Python 3.5.1. Training was carried out using a RMSProp optimiser.

Each model architecture was sent to train using each of the three datasets. The best performing model was retrained using all available training data (train and validation sets combined) for all three yellow rust datasets prior to being evaluated using the test dataset to gain a final accuracy score. All images from the test set were sent through the trained models to get a classification prediction. These predictions along with the true labels for each image were used to create a confusion matrix showing where the models made misclassifications. The confusion matrices were generated using functions from the scikit-learn (sklearn) (Pedregosa *et al*., 2011) package in python.

Having determined that our classification model 2.4, used in the experiments with pathologists and masked images, was able to handle complex input images, we chose to begin our quantification experiments with this network architecture. See chapter 3 for details about model 2.4. We could then adjust the model, test different architectures, and eventually tune the hyperparameters to get the best results for the new problem.

A new instance of model 2.4 was sent to train for each of the three datasets, YR1, YR2 and YR3. We chose to go straight in at 75 epochs as we were dealing with more complex data than the classification data, therefore it was unlikely that the model would be able to train to a high accuracy with fewer epochs. The models took approximately two days to run, significantly quicker than our classification model due to the smaller dataset.

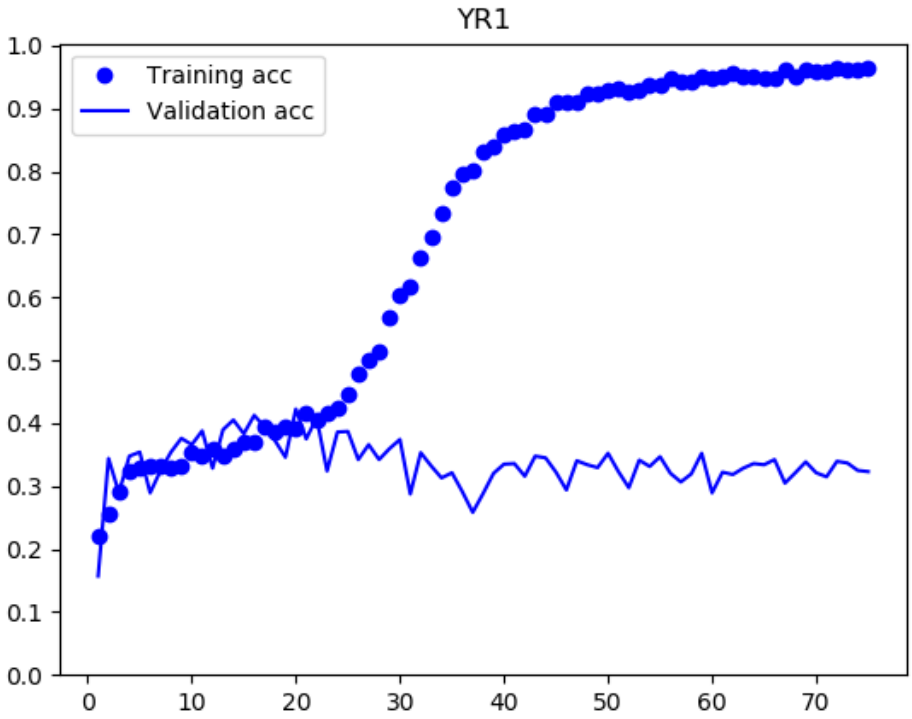


Figure 4.4: Training and validation accuracies for model 2.4 trained with YR1

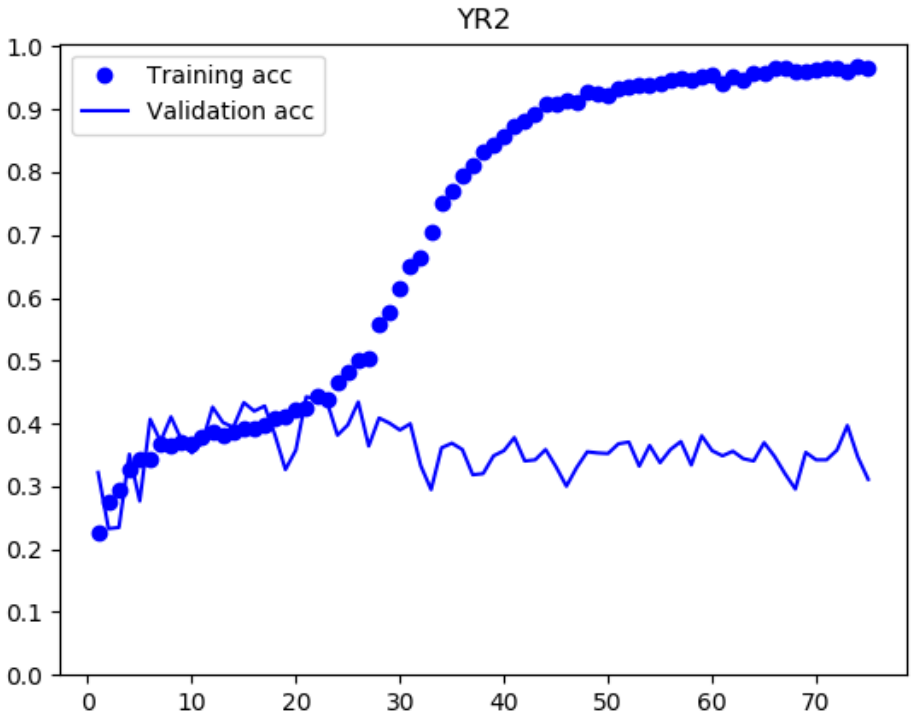


Figure 4.5: Training and validation accuracies for model 2.4 trained with YR2

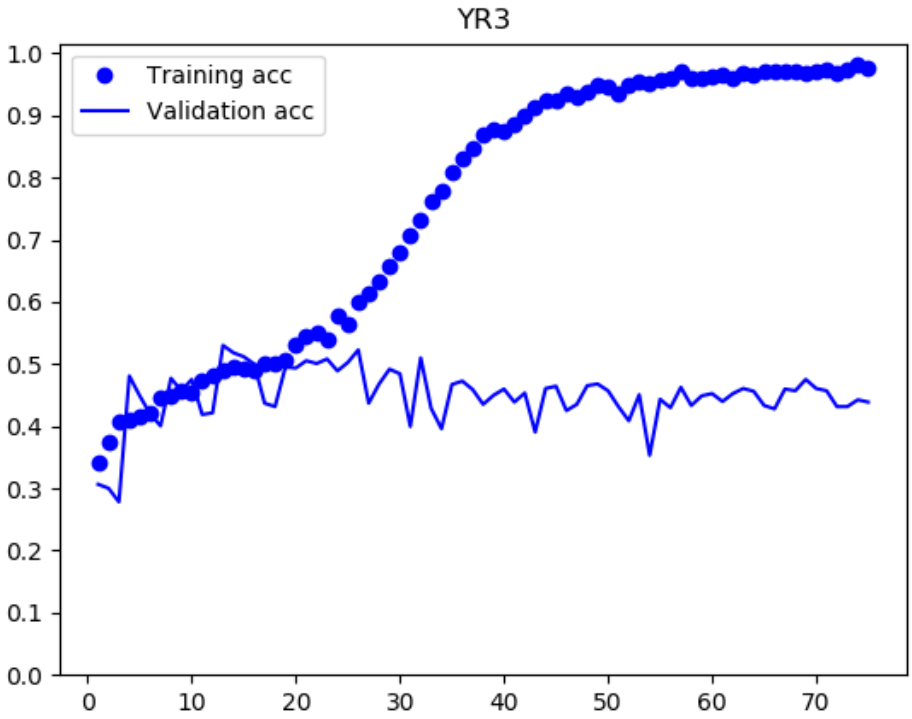


Figure 4.6: Training and validation accuracies for model 2.4 trained with YR3

Figure 4.4, Figure 4.5 and Figure 4.6 shows the training and validation plots for all three datasets. In each case, it is clear that the model starts to overfit between 20 and 25 epochs, when the validation accuracy stops plateaus. The validation accuracy peaks at approximately 40% for both YR1 and YR2 and at 50% for YR3. While this is better than our weighted random guess and zero rule baselines, they are not accuracies that would make a model useful for work in the field.

We were inspired to research attention mechanisms and modules by the work by Mi *et al*., (2020), where they used attention mechanisms for classifying wheat stripe rust on individual leaves. This work was very similar to our current problem, only using single leaves instead of full plots. We hypothesized that the methods would be transferrable, and a model of the same type would perform better on our data than model 2.4.

We repurposed a code by Deontae Pharr (https://github.com/deontaepharr/Residual-Attention-Network) which creates a residual attention network for image classification using keras. This began with a convolutional layer, then max pooling. The remainder of the convolutional base of the model consisted of residual blocks (containing residual connections) and attention modules. The model ended with three fully connected layers followed by dropout, then the final fully connected classification output layer. See Appendix 1 for full attention module code used. For each dataset, we trained an instance of this residual attention model with input image size (256,256). Initially the models were sent to train for 20 epochs, however the training accuracy seemed to be climbing still and we hoped that the validation accuracy would follow suit with further training time. Therefore, we sent the models to train for 200 epochs.

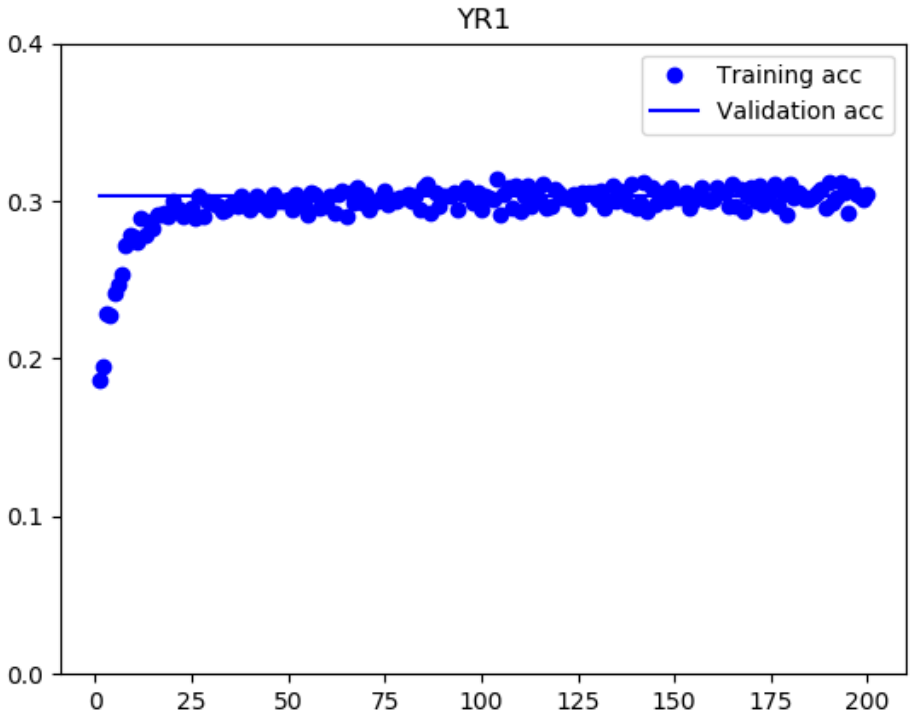


Figure 4.7: Training and validation accuracy for our residual attention model trained with YR1

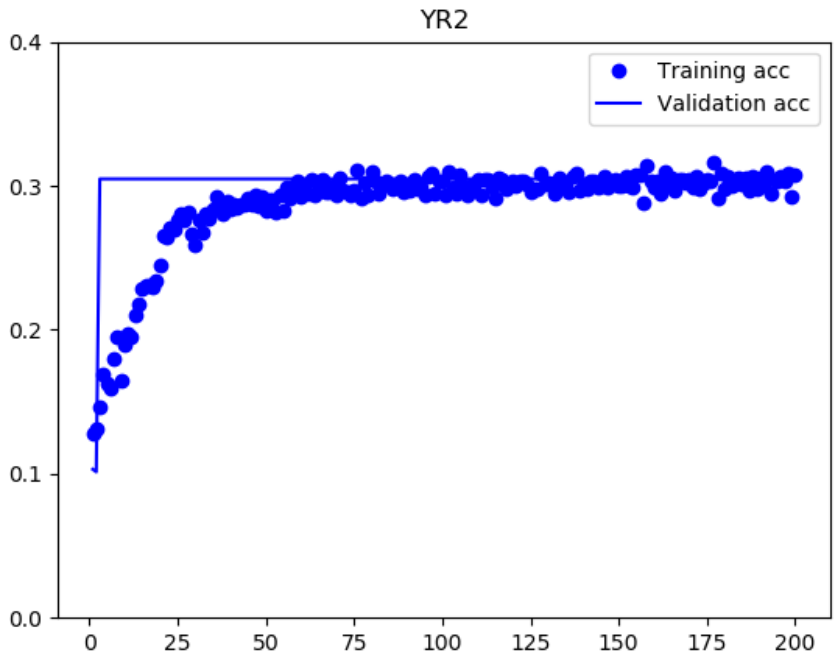


Figure 4.8: Training and validation accuracy for our residual attention model trained with YR2

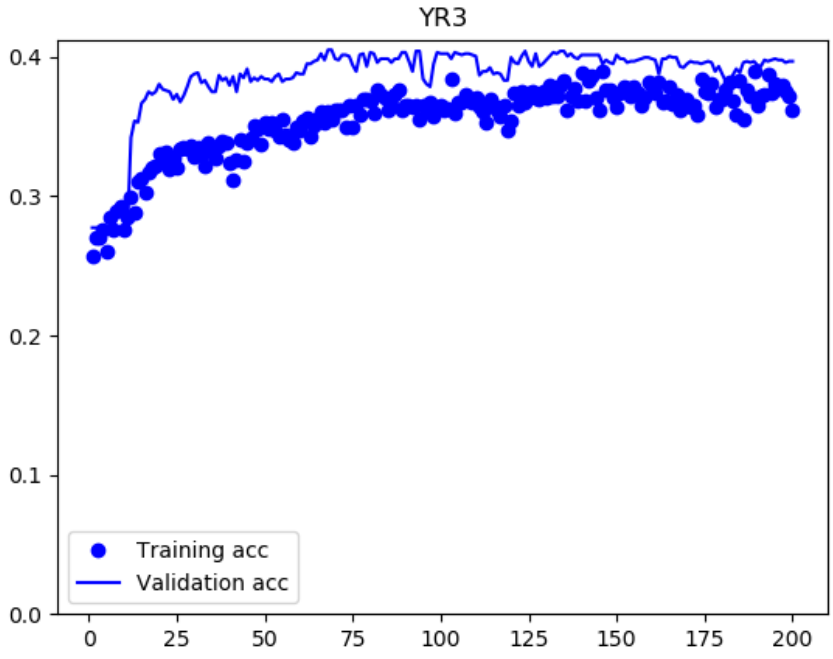


Figure 4.9: Training and validation accuracy for our residual attention model trained with YR3

The train and validation accuracy plots the residual attention model trained with YR1, YR2 and YR3 are shown in Figure 4.7, Figure 4.8 and Figure 4.9. It can be seen that in each case, the model gains both a lower train and validation accuracy. The training accuracy peaks at approximately 30% for YR1 and YR2, and approximately 38% for YR3. Meanwhile, in each case the validation accuracy yields strange results. For YR1 and YR2, the validation accuracy hits 30% immediately and does not move from that point. For YR3, the validation accuracy actually climbs higher than the training accuracy.

This could be for a few reasons. The first could be due to the use of dropout. In this model we use three dropout layers, where 50% of the features are set to zero each time during training. When the model is validated however, all of the features are used, therefore leading to a higher accuracy on the validation data than the training data. Another potential reason for the higher validation accuracy is due to the size of the dataset. The model may be learning patterns in the data, and due to the larger size of the train set when compared with the validation set, there is more variance in the train set. This leads to a higher error rate for the train set than the validation set.

We decided not to perform any further experiments with this model. Although the results were not particularly useful in terms of building a model for scoring yellow rust in the field, it did point out a potential pitfall (the size and lack of variance) in our dataset, which is important for continuing this work.

We took the res1 and sep1 model architectures from our classification experiments (section 3.2.3) to try with our three datasets. Each model was trained using each of the three datasets using an input size of (256,256). In each case, the model overfits almost immediately. In an attempt to combat this, we added dropout of 0.5 to the end of both models. We also increased the input image size to (512,512) with the aim of allowing the models to capture more features from the data.

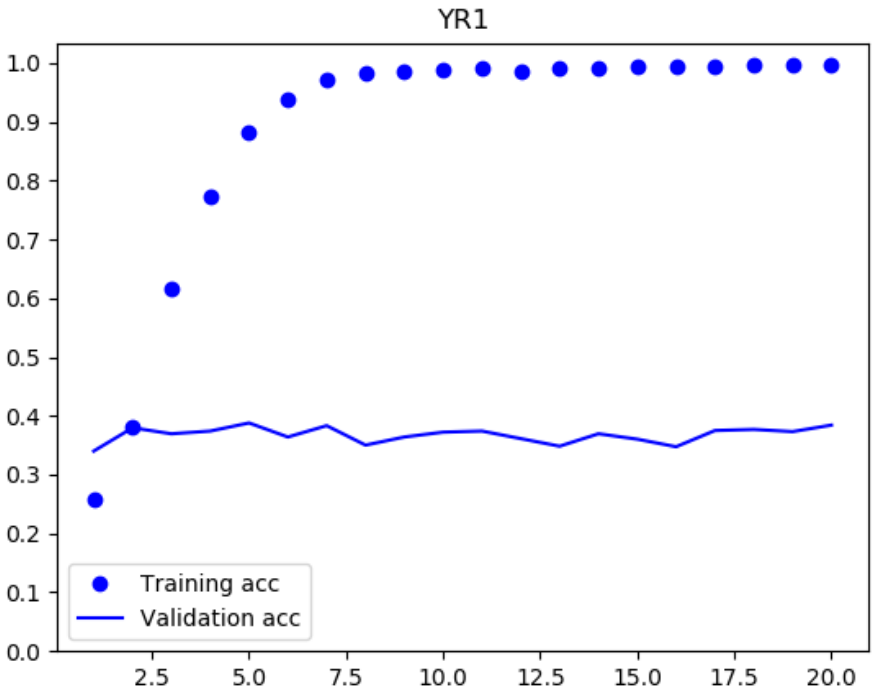


Figure 4.10: Training and validation accuracies for res1 model with added dropout for YR1

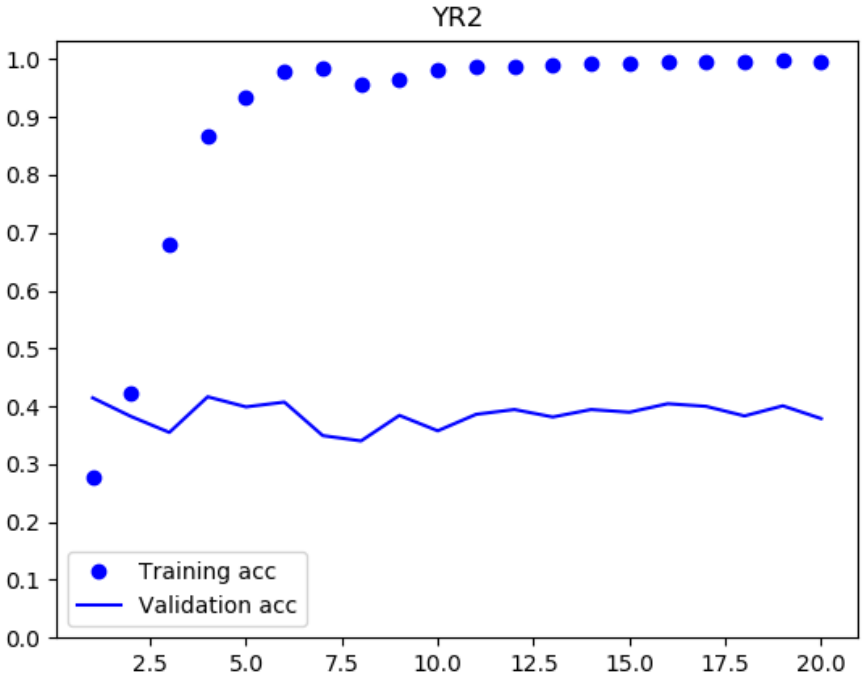


Figure 4.11: Training and validation accuracies for res1 model with added dropout for YR2

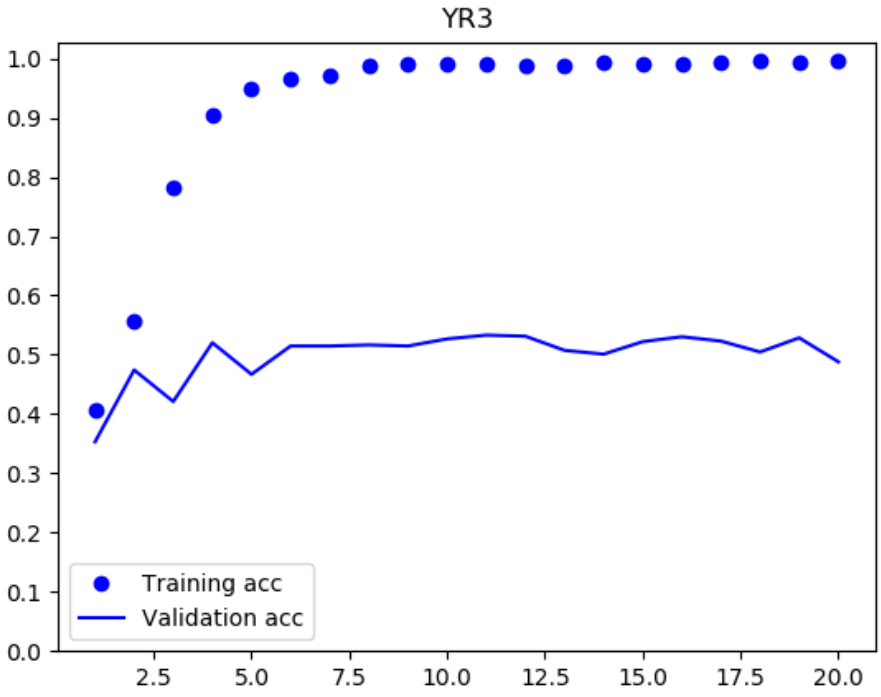


Figure 4.12: Training and validation accuracies for res1 model with added dropout for YR3

The results of these experiments yielded results that were much the same those without our additions to combat overfitting. The second residual model, with added dropout, however produced the highest validation accuracy over all the experiments. The training and validation accuracies for this model can be seen in Figure 4.10, Figure 4.11 and Figure 4.12, for YR1, YR2 and YR3 respectively.

We chose this model to send for testing for all three datasets. The purpose of testing at this point was to gain information about the classifications the models were making. From the training results, it was clear that there was going to be many misclassifications across the board. Using the predictions on the test sets from each fully trained model, we produced confusion matrices. Ideally, we would want to see the misclassifications for each score coming from the score to either side, e.g., images of score 5 would be misclassified as score 4 or score 6 when not correctly classified as score 5. If this were the case, it would show that we require more data for training, but that the data is of the correct sort and that the model should be

capable of learning to classify with the inclusion of further data. Figure 4.14, Figure 4.13 and Figure 4.15 show the confusion matrices for our residual model trained and tested using YR1, YR2 and YR3 respectively.

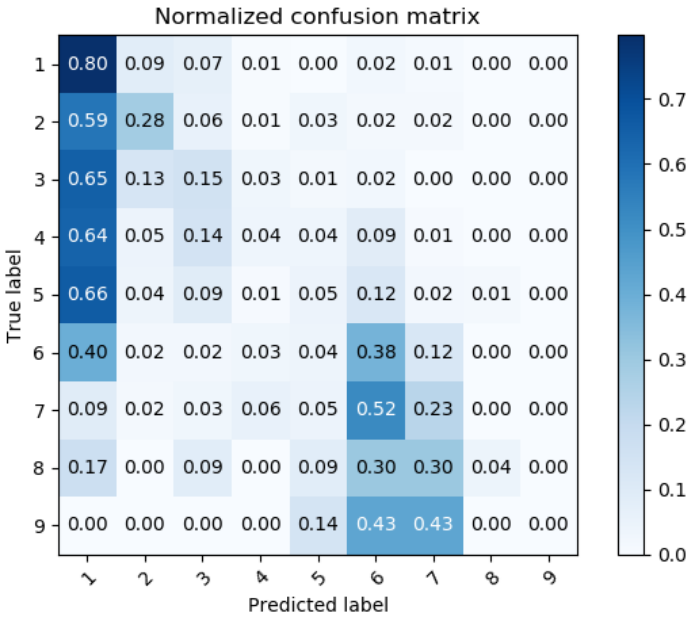


Figure 4.13: Confusion matrix of classifications for the res1 model architecture with added dropout trained on YR1

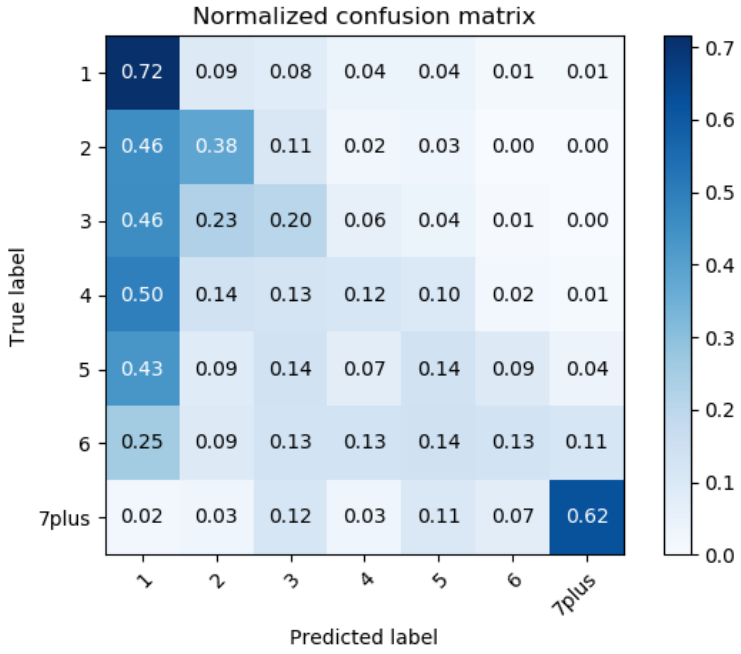


Figure 4.14: Confusion matrix of classifications for the res1 model architecture with added dropout trained on YR2

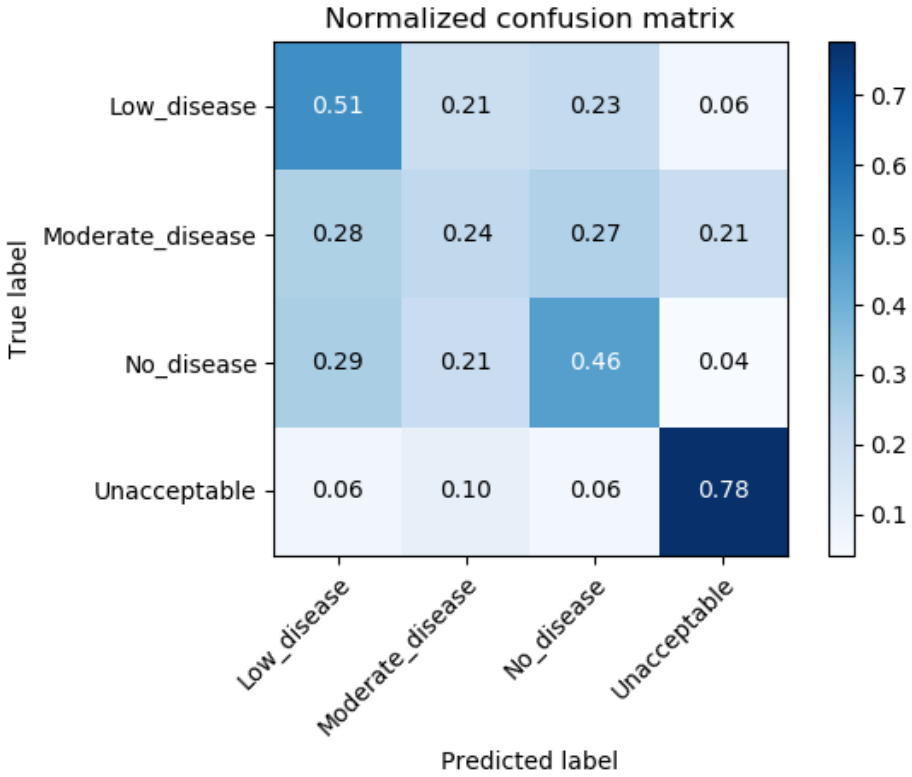


Figure 4.15: Confusion matrix of classifications for the res1 model architecture with added dropout trained on YR3

Unfortunately, all three confusion matrices show that the misclassifications are not limited to the neighbouring categories as we hoped. Instead, for YR1 and YR2, there is a strong bias to the score 1 category, which is unsurprising due to the imbalance in the dataset. Where the misclassifications are not as score 1, they are scattered across the rest of the categories suggesting that the model is struggling to learn features which are representative of the different classes.

The performance of the YR3 model is marginally better, where there are more correct classifications, especially in the unacceptable category. We believe this is because this dataset was more balanced in terms of number of images per category than the other two datasets.

It is clear that the data we have for these experiments is not sufficient for training a model to score the amount of disease present. The scattered misclassifications

throughout suggest that the data is not fully representative of the labelled classes, a problem which is also not helped by the imbalance in image quantities throughout the data for YR1 and YR2. To continue with this work, further image collection efforts will be required to balance the datasets and ensure accurately scored images. At this point we decided to not do any hyperparameter tuning with these models. We determined that any tuning would be unlikely to affect the accuracy results significantly enough to make the models useful in the field. In a situation where we were confident that the images contained accurate score information, we could have experimented with removing some of the 'no disease' data to help balance the dataset. As we did not have this confidence, we did not pursue this idea.

4.3 Creation of simulated datasets

Following the results of the work with real field images for yellow rust quantification, it became clear that a much larger and much more time-consuming image collection effort would be needed to have a chance of producing high enough accuracies for use in the field. Therefore, we devised a set of experiments with simulated data to test whether deep learning models would be capable of quantifying the amount of disease present using the 1-9 score categories from an image under more controlled conditions.

For our first experiments we created simulated images of black (0) and white (1) pixels, where black represented an 'uninfected' pixel and white was an 'infected' pixel. We used NumPy in python to create arrays of zeros of size (500, 500) where each pixel had a percentage chance of being 'infected' depending on the disease score the image was representing. We chose this size as it was similar to the input size for our classification model (512,512) and was easily divisible into percentages.

We followed the same categories for the scores as with the real field data, so created a dataset with 9 categories labelled as score_1 to score_9. Table 4.6 shows the percentage of infection each score represented and the range of infection percentages that this included to make the dataset more representative of real conditions.

For each category, a value within the percentage range was chosen using the numpy.random.uniform function. This was the probability of infection for a single pixel, we will call this POI. A 500 x 500 array of 0's and 1's was then created using numpy.random.choice, where each pixel had a 1-POI chance of being a 0 and a POI chance of being a 1.

Table 4.6: The percentage ranges used to represent each score category

Score	**% of infection**	**% range**
1	0	0
2	0.1	0.01 – 0.4
3	1	0.5 – 2
4	5	2.1 – 7
5	10	7.1 – 17
6	25	17.1 – 35
7	50	35.1 – 60
8	75	60.1 – 80
9	100	80.1 – 100

The resultant arrays were converted to datatype uint8 and assigned a unique universal identifier (UUID). They were then saved to a folder corresponding to the score number as a .png using python image library (PIL).

For the initial dataset we made 10,000 images per category, 7000 in the train set, 2000 in the validation set and 1000 in the test set, we will call this dataset S10k. This was used as the ideal dataset, which would have enough images per category to show the full potential of deep learning models for this problem.

Following this, we made four other smaller datasets with 100, 250, 500 and 1000 images per category sorted in the same proportions, these will be known as S100, S250, S500 and S1k respectively. These were designed to try and find guide for the number of images required for producing high accuracies. Each dataset contains images of the same type, but new images were generated for each dataset purely because it took less computational time to generate brand new images than it did to

copy a portion of the original dataset. Image generation was done using a MacBook Pro connected to the internal JIC VPN, to allow them to be stored in the group scratch location. Figure 4.16 shows an example image from each of the 9 score categories for the S datasets.

Table 4.7 shows the number of images in each category for the train, validation, and test sets for all five S datasets created.

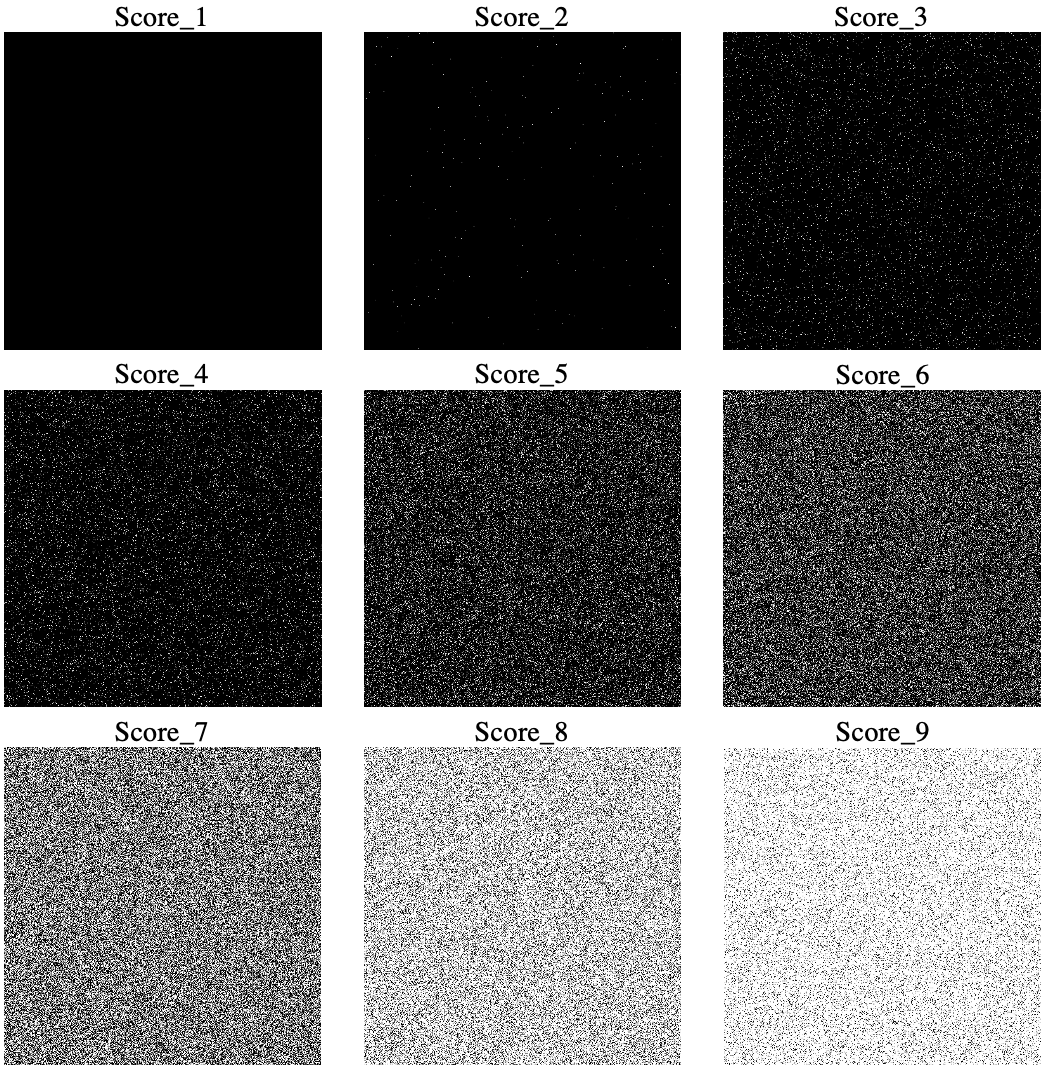


Figure 4.16: An example image for each category in the S datasets

Table 4.7: The distribution of images across the whole dataset and the train, validation, and test sets for all the S datasets

Dataset	Total images per category	No. of train images per category	No. of validation images per category	No. of test images per category
S10k	10,000	7,000	2,000	1,000
S1k	1,000	700	200	100
S500	500	350	100	50
S250	250	175	50	25
S100	100	70	20	10

The next step towards creating a more realistic simulated dataset was to arrange the infected pixels in a way that was more representative of disease lesions on a leaf. To do this we made a new dataset of simulated images, again using zeros as 'healthy tissue' and ones as 'infected tissue', however this time the ones were added in line, or stripe, formations. This was done to imitate yellow rust lesions, to test whether having patches of infection produces a different outcome to having a uniformly distributed infection.

To create this data, we again started with a 500 x 500 NumPy array of zeros. We also defined a 2 x 50 array of ones, which was our individual stripe lesion. The POI was randomly selected in the same way as before for each category and used to calculate the number of pixels that would need to be converted into ones to reach that percentage, we will call this the infection level. To place the stripe lesions onto our zeros array we used a loop to randomly select a zero coordinate. There we replaced the surrounding pixels with the ones in our stripe array. After each loop, the number of ones in the array were counted and if it were fewer than the infection level, then the process of adding a stripe lesion was repeated. If the number of ones was equal to the infection level, then the loop was broken, and the array was ready to be saved as an image. Finally, if the number of ones was greater than the infection level then we calculated how much greater and changed that number of ones back to zeros

(randomly selecting ones across the entire array). The arrays were saved in the same way as for the S datasets (S10k, S1k etc.). We created two datasets, first with 1000 images per category and then with 10,000 images per category, named stripe1k and stripe10k respectively. Figure 4.17 shows an example image for each score category in the stripe datasets.

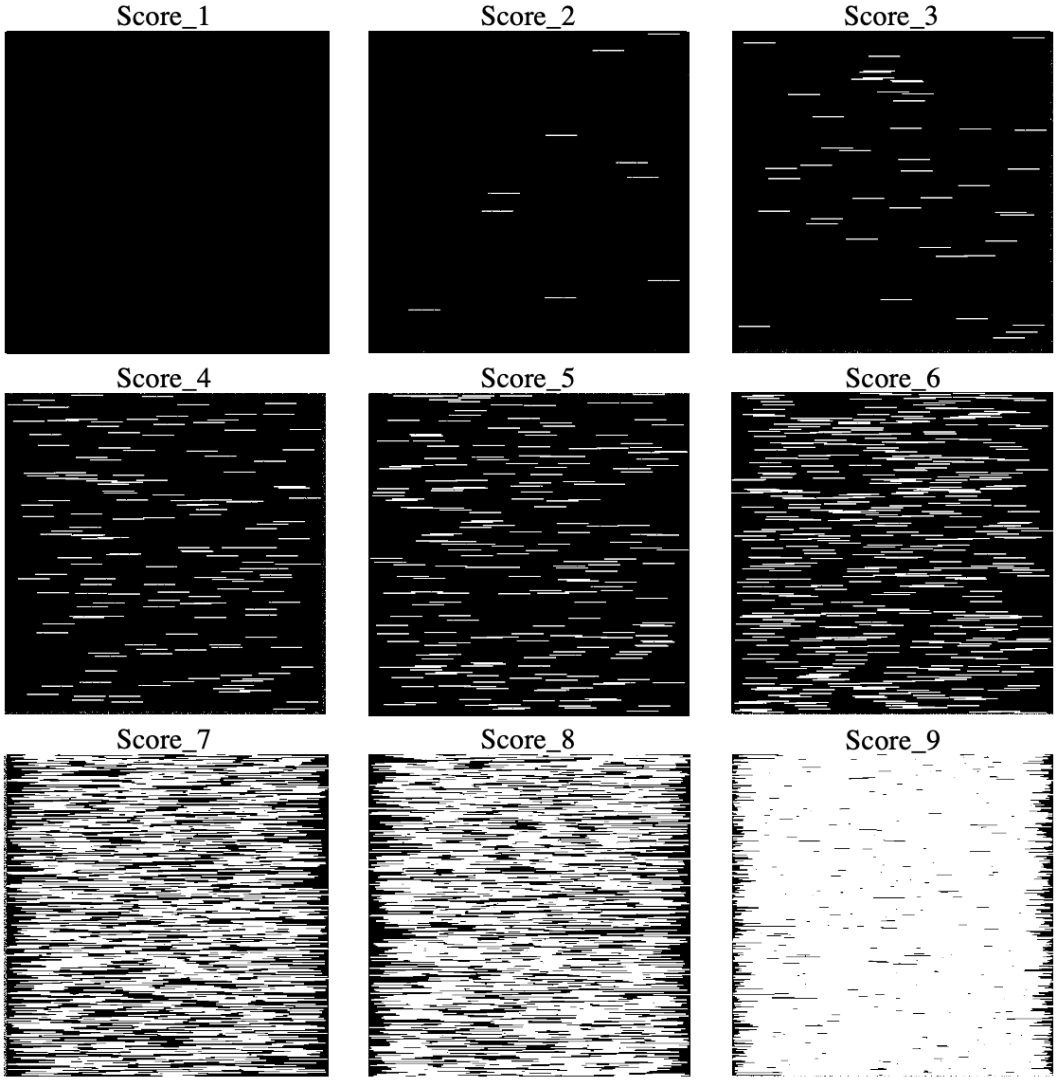


Figure 4.17: An example image for each score category for the stripe datasets

We were interested to see how using more realistic colours in our images would affect the results of training. We hypothesised that it would make little difference to the results, however it would be useful to discern whether the model is learning any internal colour representations. We created a third set of data in the same way as the

stripe dataset, however when converting the array to an image prior to saving, we used a colour palette to assign green to all zeros as ‘leaf tissue’ and orange to all ones/ stripes as ‘infected tissue’. Again, we created two datasets with 1000 and 10,000 images per category, named colstripe1k and colstripe10k. Figure 4.18 shows an example image from each score category for the colstripe datasets.

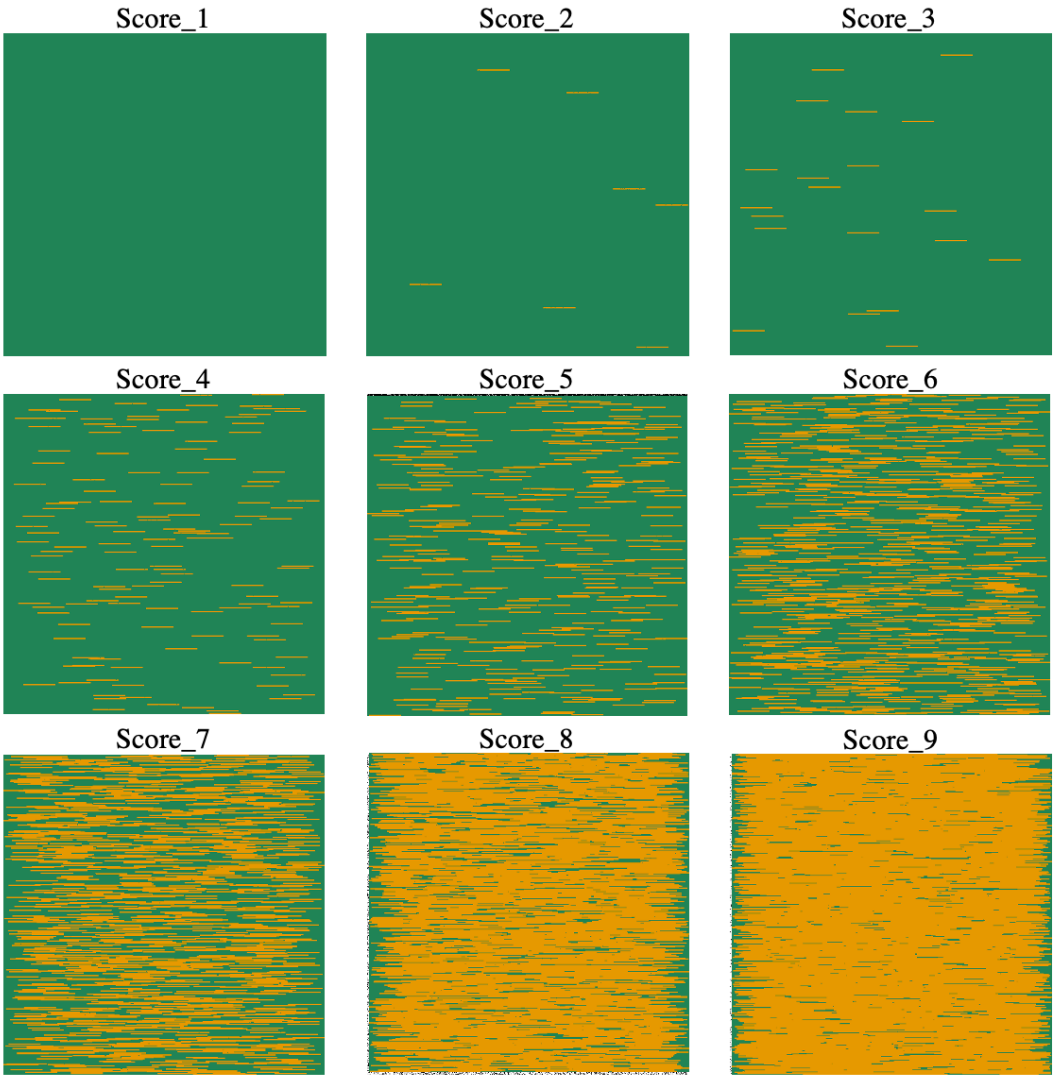


Figure 4.18: An example image for each score category in the colstripe datasets

In real field images, there is a lot of background information, such as sky, soil, or stones, as well as leaves and disease. We wanted to introduce this into our simulated data and so take them one step closer to realistic images. A fourth set of images was made, again beginning with a 500 x 500 array of zeros. This time, instead of

calculating the infection level for the whole array, it was calculated for 90% of the array. We considered this 90% of the array to be leaf tissue which could be infected (225,000 pixels) and the other 10% would be background information. The stripes were placed again using the same method until the infection level was reached. Once all stripes were placed, the background information was added. In our colour palette, we assigned each of the five background numbers a colour that would be typical of the kind of background information that would be found in the field (brown, blue, grey). 25,000 pixels (equal to 10% of the array) containing a zero (healthy pixel) were randomly chosen and replaced with either 2, 3, 4, 5 or 6, with a 20% chance of choosing each value. This left an image with 10% of the pixels representing background information uniformly distributed across the array. To pick the colours used in our data and get their RGB values we used the website coolors.co. See Figure 4.19 for the palette we chose from this website. The two datasets created in this way were names bg1k and bg10k. An example image for each score category in the bg datasets can be seen in Figure 4.20.

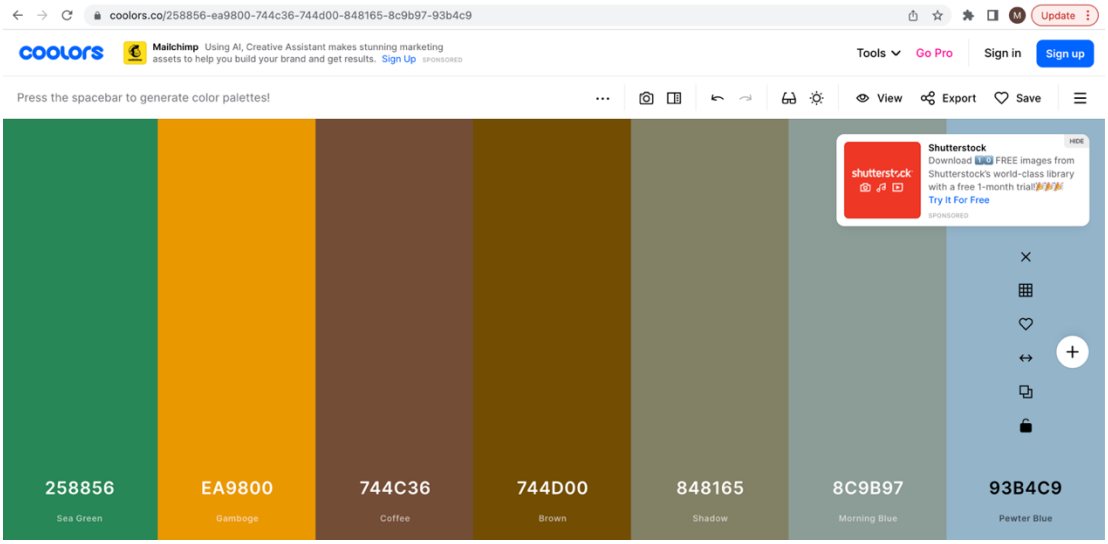


Figure 4.19: The colour palette used to make our datasets, from coolors.co

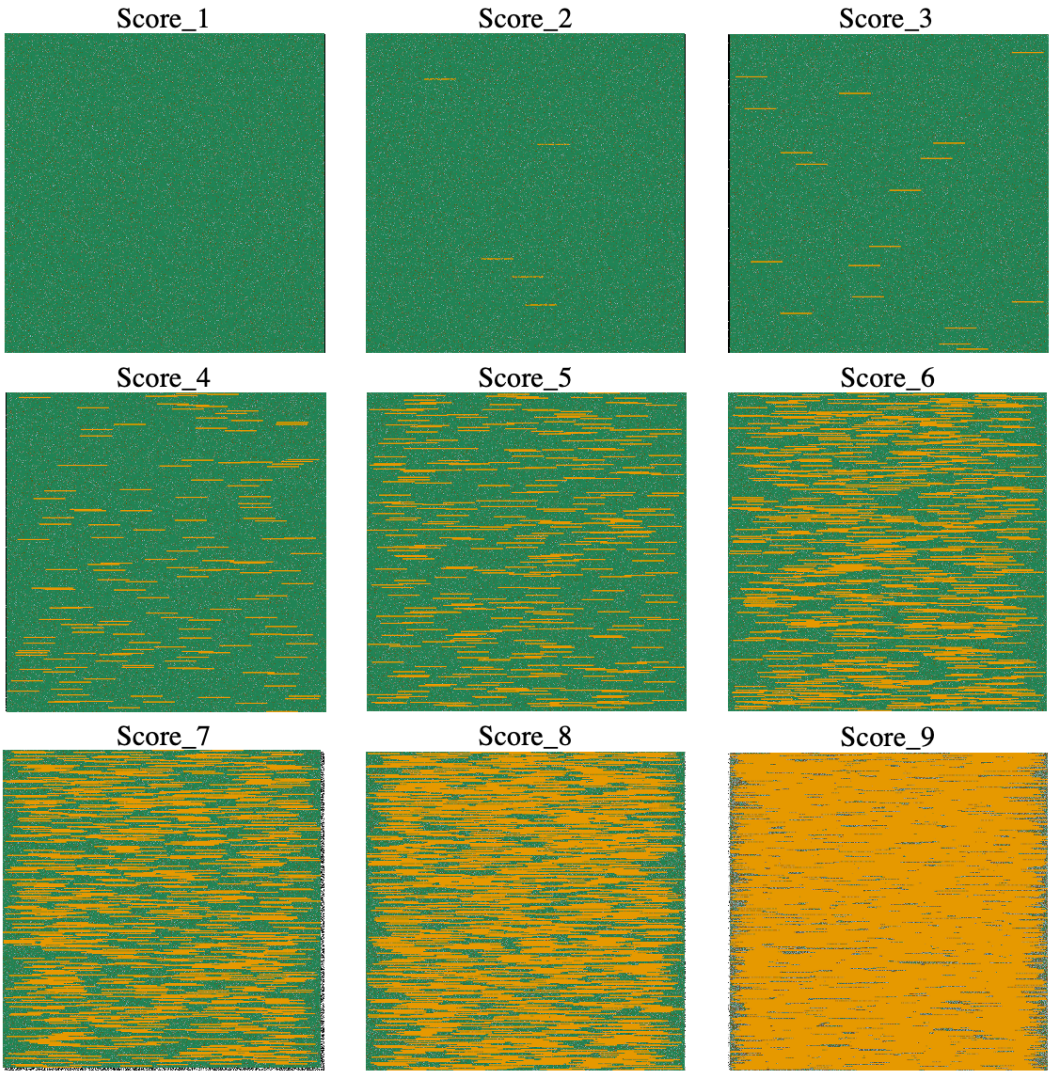


Figure 4.20: An example image for each score category for the bg datasets

The simulated images for the stripe, colstripe and bg datasets required more computing power and time to produce than for the S dataset images. For this reason, the GPU node on the JIC HPC facilities were used to speed up the process.

4.4 Training deep learning models using simulated data

For the purpose of testing whether a deep learning model has the potential for disease quantification from images, we simply took the final model architecture from our classification network and used it for all of our experiments. We had shown this

architecture was capable of classifying complex images, which we hypothesised would translate to this problem also. As we were only testing the viability of using deep learning models for scoring levels of infection and not aiming to find the ideal network architecture, we did not do any hyperparameter tuning at this stage.

Initially, we trained a version of our model for 5 epochs using the large S10k dataset. The aim of this was to determine whether a deep learning model would be able to classify an image into severity of infection (where white pixels are infected and black are healthy). The results of the real field data experiments showed that training broke down within the first few epochs, so it was logical to test whether the same would happen here before training for longer. Following this the four smaller S datasets were used to find a guide for the minimum number of images required to train a model of this kind. A new instance of our model was trained with each of these datasets.

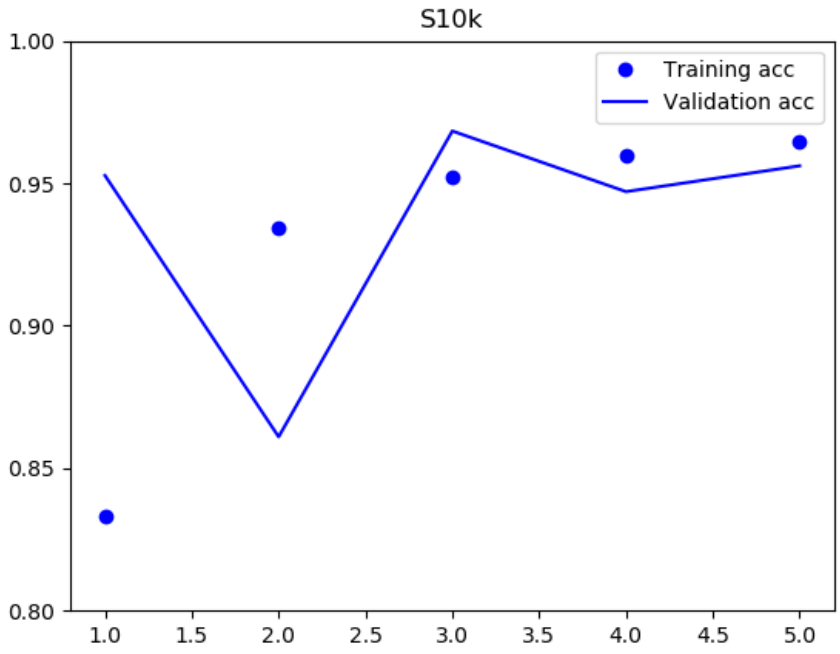


Figure 4.21: Training and validation accuracies for model trained with S10k for 5 epochs

Having decided on the optimum number of training epochs for each of the datasets, a new instance of the model was trained on all available data (train and validation together) for the determined number of epochs. Then, the model was shown the test images, which it had never seen before, and a final accuracy score was given. As with the real field data, we used the test data to collect classification predictions and generate confusion matrices for each of the fully trained models.

Figure 4.21 shows the training and validation accuracies for each of the five epochs when trained using S10k. We can see that the validation accuracy stays close to the train accuracy, with expected fluctuation in the earliest epochs, and shows no obvious signs of overfitting. It reaches approximately 95% accuracy, however there is possibility that it could climb a percentage point or two higher with more training epochs.

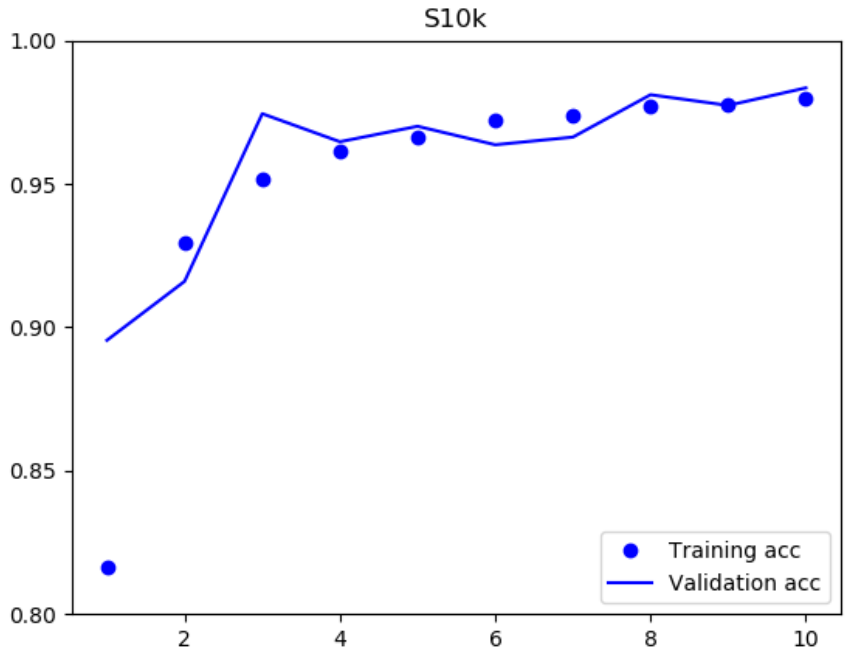


Figure 4.22: Training and validation accuracies for model trained with S10k for 10 epochs

For this reason, we re-ran our code to train the model for 10 epochs. It can be seen from Figure 4.22 that the validation accuracy stays very close to the train accuracy throughout training. There is still potential for higher accuracies with further

training, however this was not an exercise to maximise classification accuracy. Therefore, this model was sent to test to get a final accuracy and obtain classification prediction information for the confusion matrix, see Figure 4.23. The confusion matrix shows that the model classifies all categories with very high accuracies (95% or above), and that all misclassifications occur within the score categories adjacent to the true label.

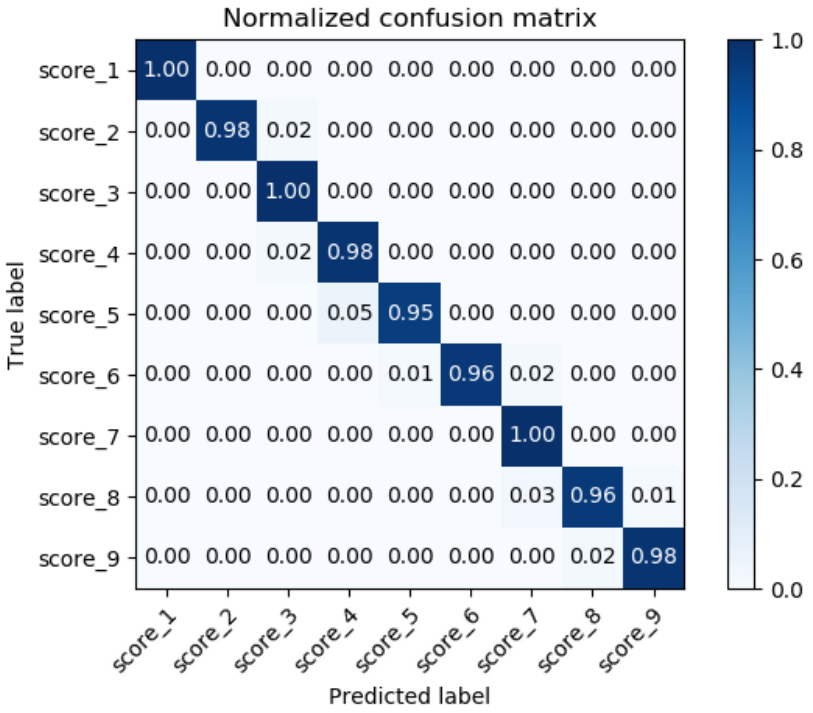


Figure 4.23: Confusion matrix of the classifications by model trained with S10k

We created the four smaller datasets with the aim of finding a lower limit on the amount of data required for a problem like this. Using each dataset, a new model was trained for five epochs initially. Again, this was done to ensure that training would run as expected and that there would be no overfitting due to smaller amounts of training data.

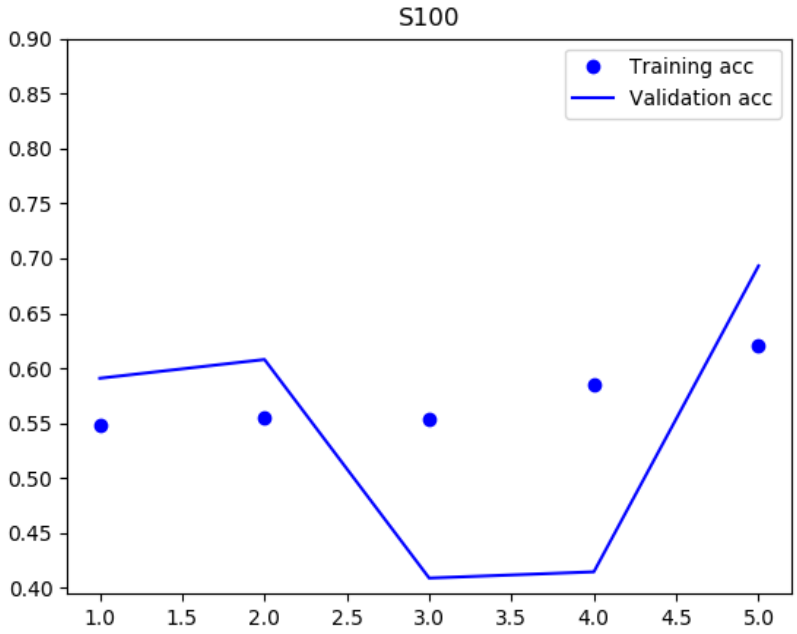


Figure 4.24: Training and validation accuracies for model trained using S100 for 5 epochs

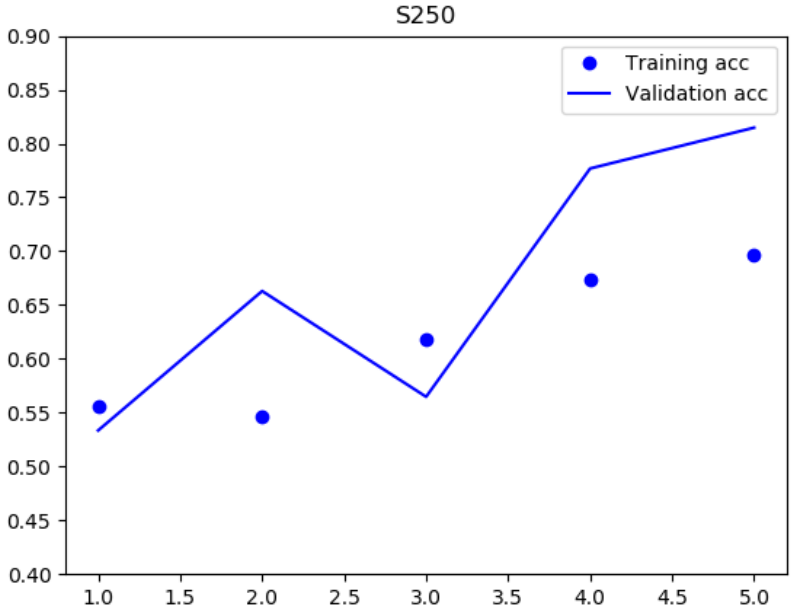


Figure 4.25: Training and validation accuracies for model trained using S250 for 5 epochs

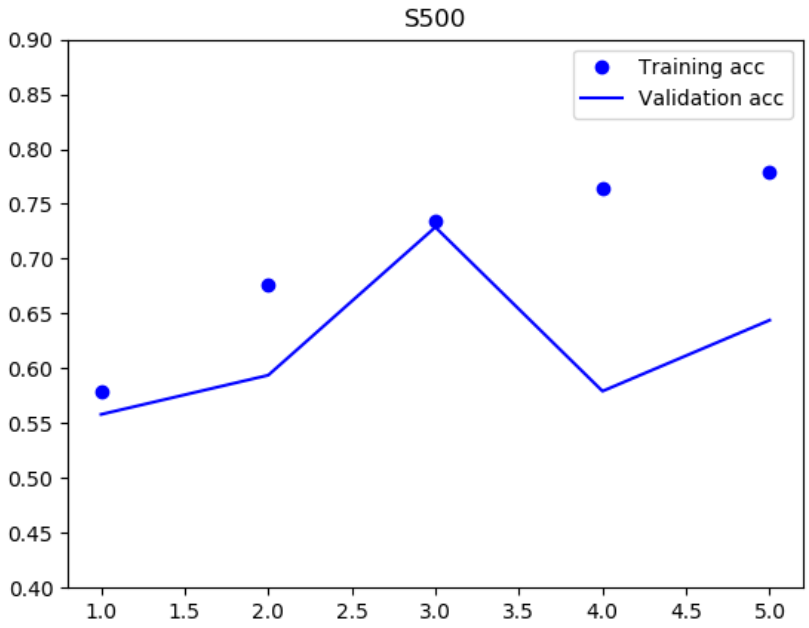


Figure 4.26: Training and validation accuracies for model trained using S500 for 5 epochs

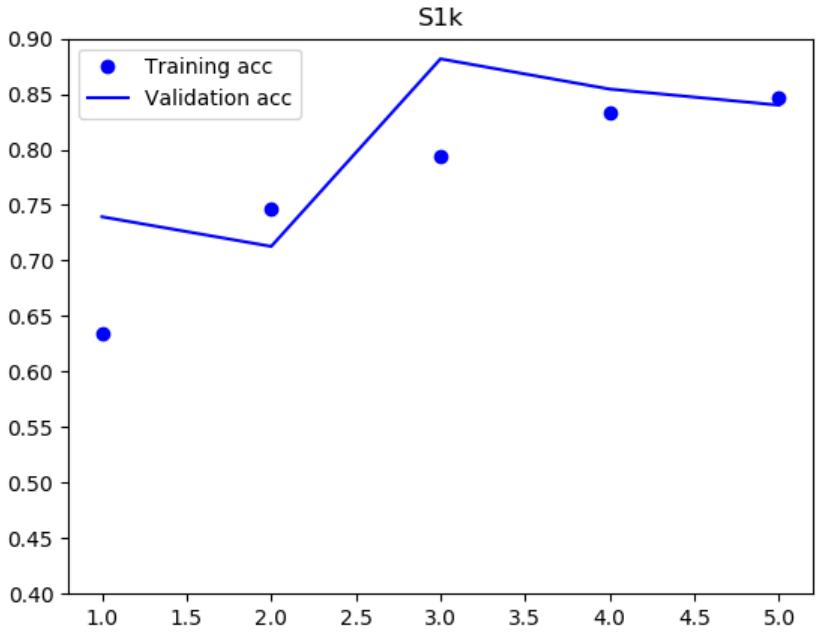


Figure 4.27: Training and validation accuracies for model trained using S500 for 5 epochs

Figure 4.24, Figure 4.25, Figure 4.26 and Figure 4.27 shows the training and validation plots for the models trained on S100, S250, S500 and S1k for five epochs. In each case the training accuracy still appears to be climbing. The validation accuracies are harder to interpret due to the small number of epochs. For S250 and S1k, the validation accuracy appears to be mostly climbing in the same way as the training accuracy, however for S100 and S500 the validation accuracies are a little more erratic. This could be due to overfitting, however more likely simply the early stages of training where the use of the validation set of images is ensuring that the models do not learn specifics from the train set. Training for further epochs would confirm this.

We concluded that it would be beneficial to try training for all four datasets for further epochs. We trained a new model for each dataset for 20 epochs. Following this the S100, S250 and S500 models all looked like they had the potential for their accuracies to climb further with more training, so they were sent to train for 50 epochs. The validation accuracy for the S1k model appeared to peak within the 20 epochs, therefore it was not sent to train for further epochs.

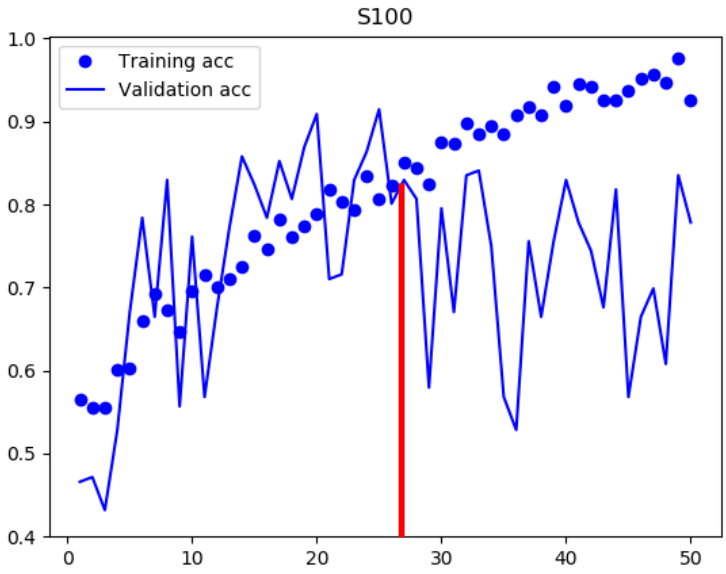


Figure 4.28: Training and validation accuracies for model trained using S100. The red line indicates where the validation peaks and the number of epochs that the model will be trained for using all available data

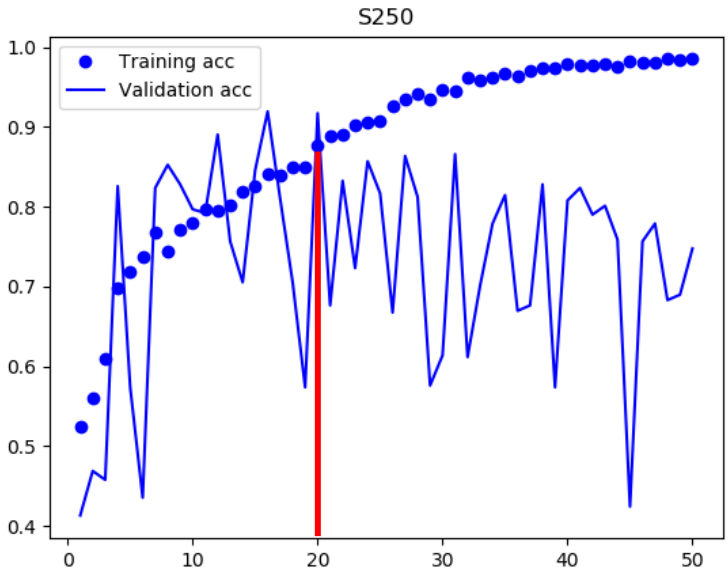


Figure 4.29: Training and validation accuracies for model trained using S250. The red line indicates where the validation peaks and the number of epochs that the model will be trained for using all available data

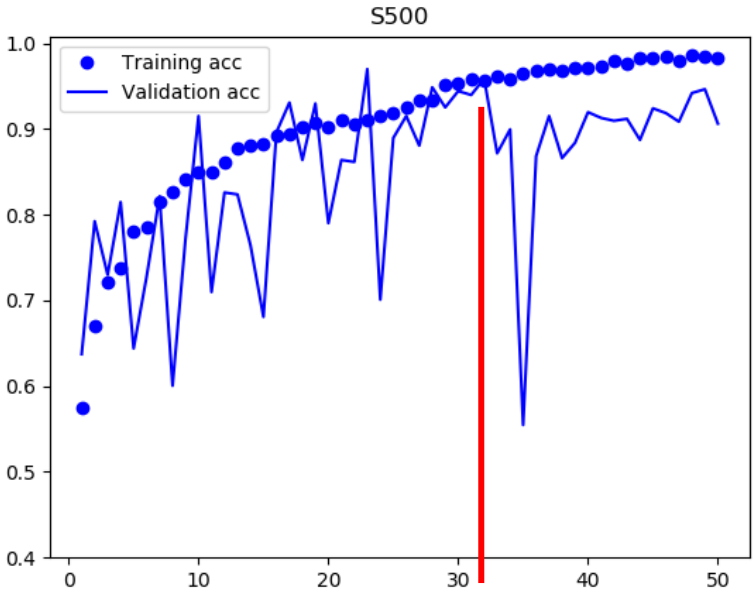


Figure 4.30: Training and validation accuracies for model trained using S500. The red line indicates where the validation peaks and the number of epochs that the model will be trained for using all available data

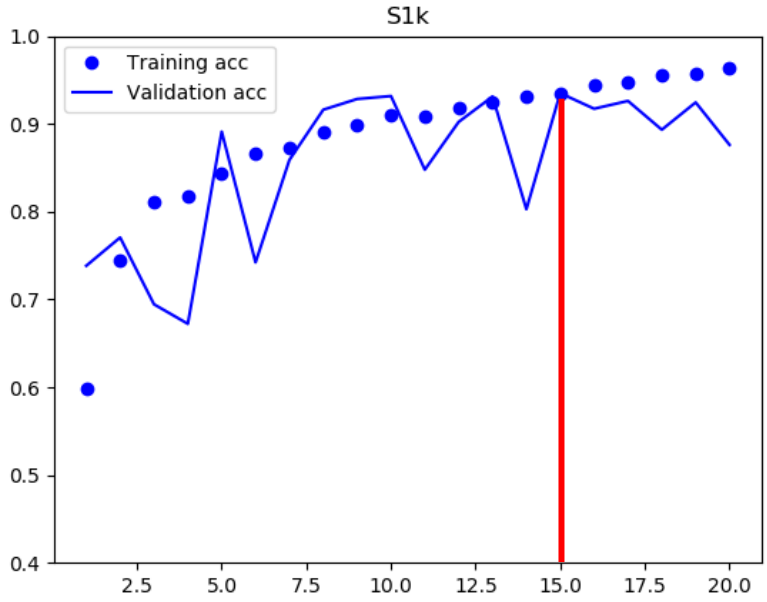


Figure 4.31: Training and validation accuracies for model trained using S1k. The red line indicates where the validation peaks and the number of epochs that the model will be trained for using all available data

After training models for 20 epochs with S1k and 50 epochs with S100, S250 and S500, we were able to see clear points where the validation accuracy reaches its highest point before plateauing and the model starts to overfit. Figure 4.28, Figure 4.29, Figure 4.30 and Figure 4.31 shows the training and validation accuracy plots for these four models with the epoch where the validation peak is reach marked with a red line. This point marks the number of epochs that each model will be trained for using all available data (training plus validation) prior to being tested with new images from the test set. See **Error! Reference source not found.** for the number of epochs used for final training of the four models along with their final accuracy results on the test data.

Table 4.8: The number of epochs used for final training and the final accuracies on the test sets for each of the S datasets

Dataset	**No. epochs for final training**	**Final accuracy**
S100	27	78.4%
S250	20	66.9%
S500	32	94.2%

S1k	15	94.1%

Figure 4.32 shows the confusion matrices for datasets S100, S500 and S1k. There is a clear improvement in the results as the size of the dataset increases, with the S1k dataset producing the best classification results. The misclassifications for each dataset predominantly fall within the classes on either side of the true label, which is what was expected. One class that seems to consistently cause some issues throughout is the score 7 class. We thoroughly checked the data and concluded that the images all contained the correct score level.

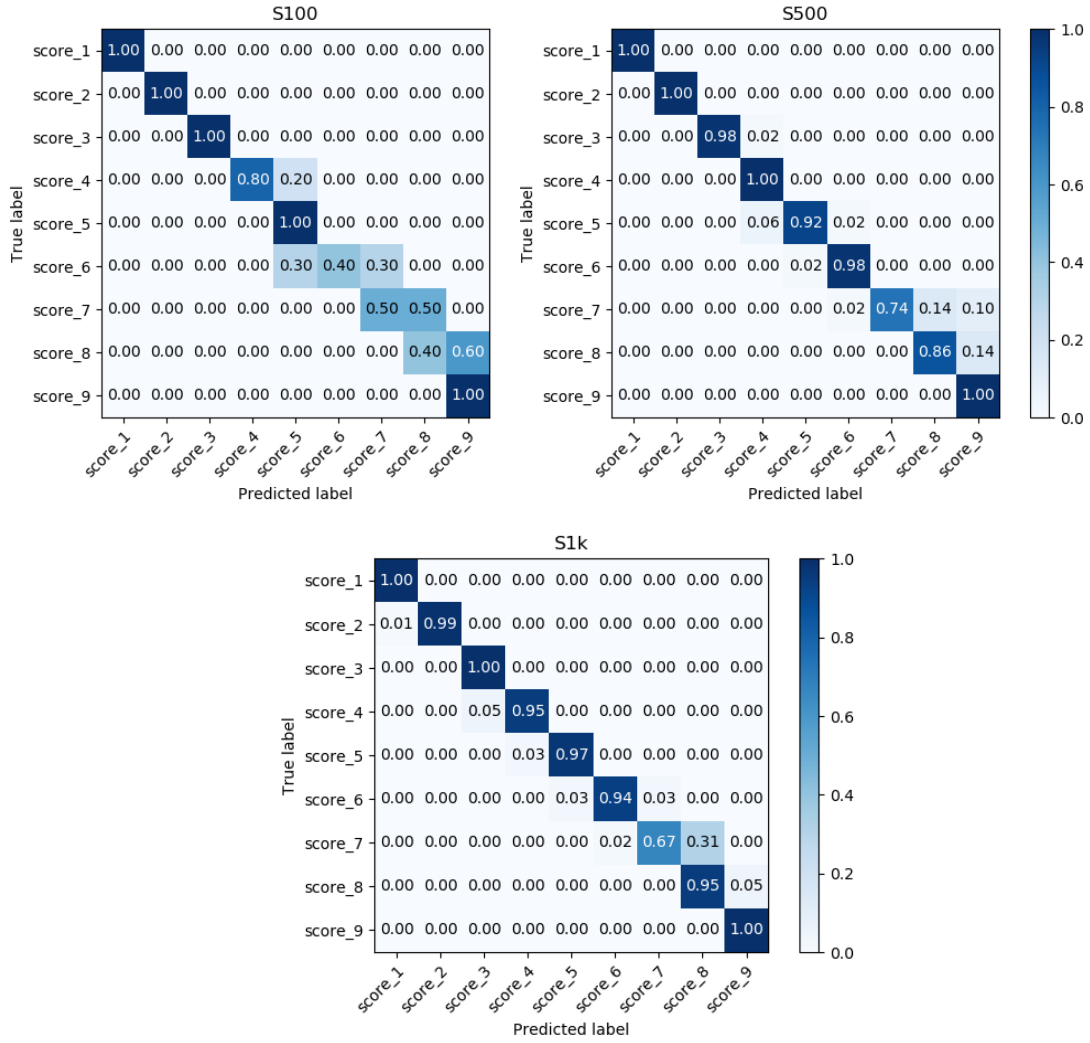


Figure 4.32: Confusion matrices for models trained using S100 (top left), S500 (top right) and S1k (bottom)

We theorise that the issues which occur with the higher score categories are due to the wider ranges for the percentage of infection, thus creating more variation in the images within those categories. Looking at small thumbnails of a selection of images from a lower score category and a higher score category, there is a clear difference in the average shade range. In the lower category, e.g., score 2 (see Figure 4.33) the images are all a very similar shade of grey. For the higher score category, score 7 (see Figure 4.34), the shade of grey varies more due to a wider range in the number of infected pixels that the images could contain. This could be causing the model to have more trouble learning the range of features for the higher categories than the lower ones with less variation.

The results that were gained from the S250 dataset proved to be more problematic. There were many more misclassifications across the board, with one category (score 6) gaining 0 correct classifications, see Figure 4.35. Furthermore, the misclassifications were not restricted to the classes on either side of the true label but were instead more scattered. Scores 6, 7 and 8 had many misclassifications as score 4 and score 9 was misclassified as score 6.

Upon collecting these results, the dataset and code to create the dataset were thoroughly checked for any issues which could have caused the discrepancies. No issues were found, so we decided to repeat the full train and testing process, with a newly generated set of S250 data (S250_2), to see if the results were replicated. Using the training graph from the original experiment as a guide, we trained a new instance of our model with the new data for 30 epochs (in case the validation accuracy took longer to peak in experiment). The validation accuracy peaked at epoch 20, the same as previously. Final training was done for 20 epochs and the model evaluated on the test set from S250_2. Figure 4.36 shows the confusion matrix for the second experiment, with the new S250_2 dataset. The issue with score 6 is still present, however the misclassifications throughout are confined to the adjacent categories to the true label.

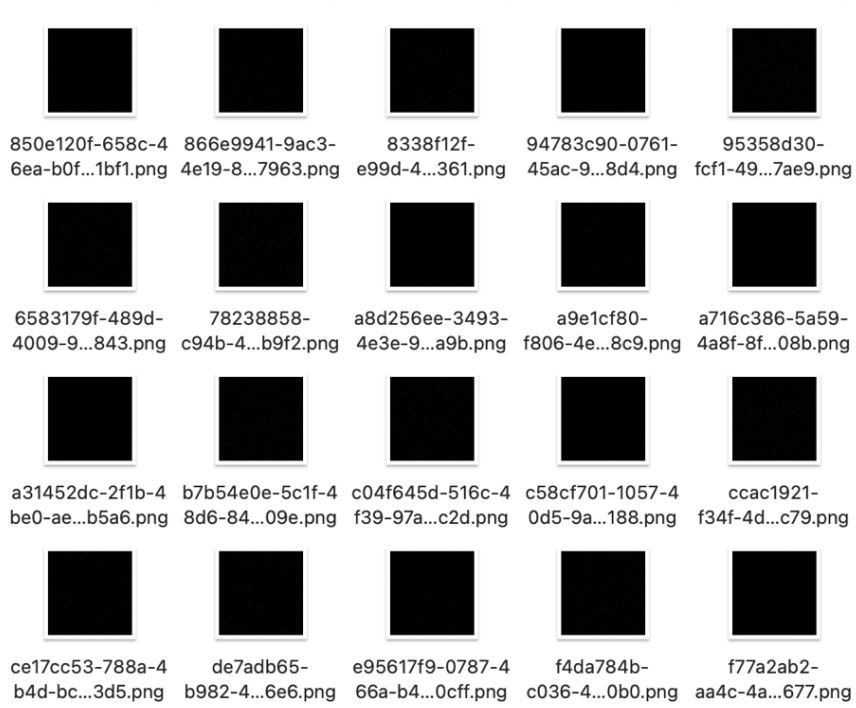


Figure 4.33: A selection of thumbnails of the images contained in the score 2 category of the S datasets

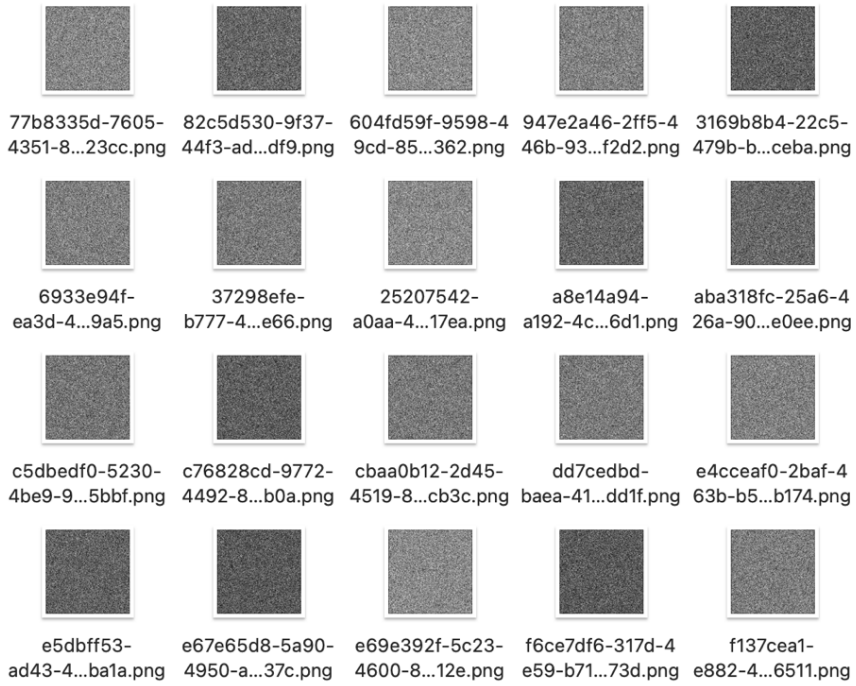


Figure 4.34: A selection of thumbnails of the images contained in the score 7 category of the S datasets

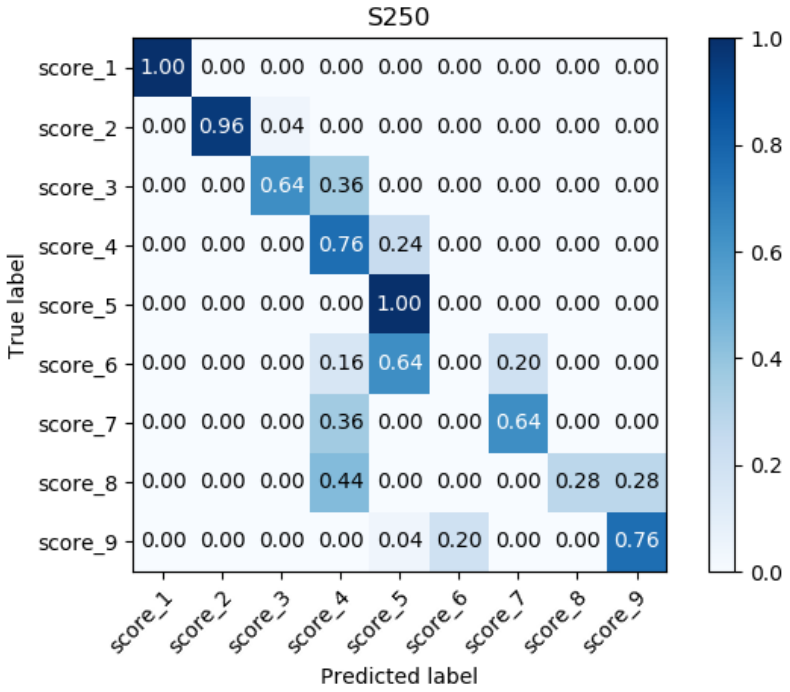


Figure 4.35: Confusion matrix for model trained using S250 dataset

Upon collecting these results, the dataset and code to create the dataset were thoroughly checked for any issues which could have caused the discrepancies. No issues were found, so we decided to repeat the full train and testing process, with a newly generated set of S250 data (S250_2), to see if the results were replicated. Using the training graph from the original experiment as a guide, we trained a new instance of our model with the new data for 30 epochs (in case the validation accuracy took longer to peak in experiment). The validation accuracy peaked at epoch 20, the same as previously. Final training was done for 20 epochs and the model evaluated on the test set from S250_2. Figure 4.36 shows the confusion matrix for the second experiment, with the new S250_2 dataset. The issue with score 6 is still present, however the misclassifications throughout are confined to the adjacent categories to the true label.

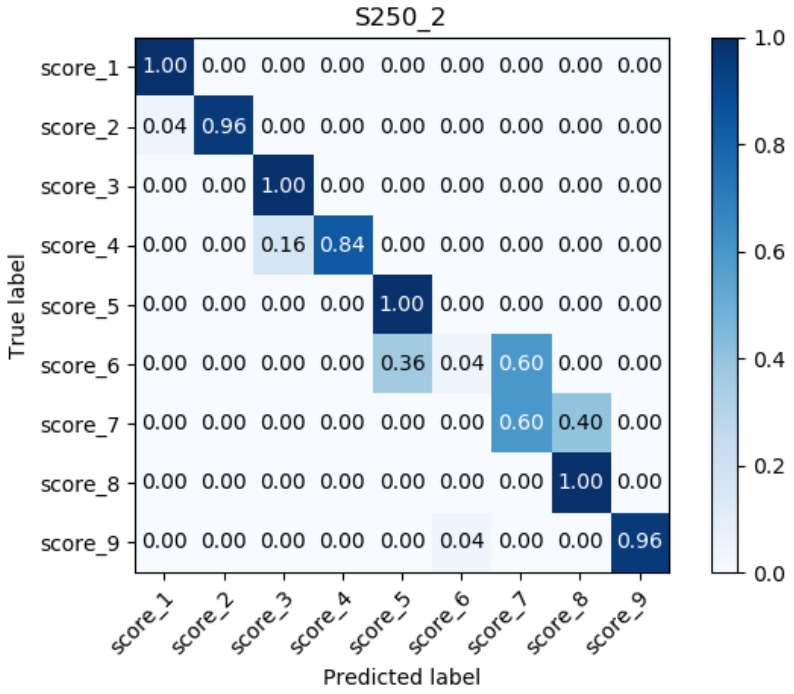


Figure 4.36: Confusion matrix for model trained using S250_2 dataset

We hypothesise that the reason for the issues in the two S250 experiments is due to the amount of available training data. There is not enough data in the validation set to be able to guide the model into making the correct representations from the images. The larger datasets are able to better find the patterns for each category, thus leading to fewer misclassifications.

The reason we do not see the same sort of results in the even smaller S100 dataset could simply be because the test set used is so small. With only 10 images per category for testing, it is harder to gauge the true performance of the network as it has fewer opportunities for misclassifying. To test this, we used the model trained using the S100 data to generate predictions using the larger test sets from the S250, S500 and S1k datasets. Figure 4.37 shows the confusion matrices for each of the three experiments. Using more data highlights an issue in the score 7 category similar to the issue we saw in the first S250 experiment.

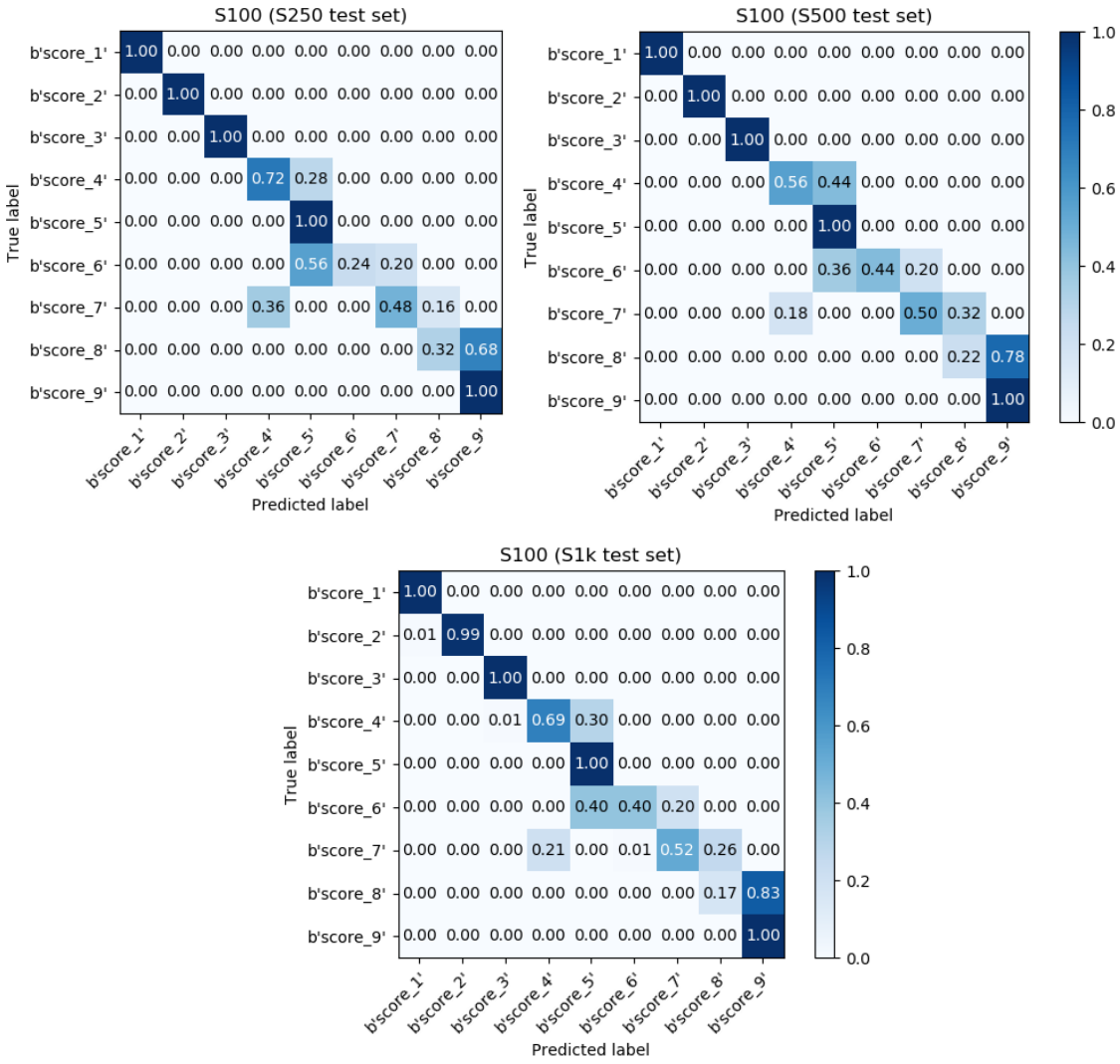


Figure 4.37: Confusion matrices for model trained using the S100 dataset and evaluated on the test sets from S250 (top left), S500 (top right) and S1k (bottom)

Our experiments show that the S1k dataset produces the most consistent classifications throughout. For this reason, we determine that 1000 images per category is a good guide for creating a dataset. Any fewer results in some of the classes with higher levels of infection getting misclassified more than the lower levels of infection.

Following the experiments using the S datasets, we worked through the other simulated datasets (stripe, colstripe and bg) to test the ability of our network as we move towards more realistic data. In each case, the larger datasets containing 10,000 images per category were used to determine the results with an ideal amount of data.

The datasets with 1000 images per category were used to test how the model would perform with our smaller size datasets, which is a more realistic amount for collecting in the field.

To test our model with a distribution of infected pixels that was more realistic, we created the stripe and colstripe datasets. Using these, we were able to test whether a deep learning model would be able to classify the images by level of infection when the infected pixels were not distributed uniformly across the image. As we had been working with yellow rust, which appears with stripe lesions on the leaf, we decided to place the infected pixels in stripe formations. We did this both in black and white, like the S datasets, and in green and orange to mimic the green leaf tissue and orange yellow rust lesions. This was done to test whether the colour of the images had any effect on the results.

We also added noise to our data in a third set of data. This bg data included uniformly distributed 'background' information. In realistic settings, there would be soil, sky and stones etc. included in any image taken. The bg datasets were used to determine how the results would be affected with the addition of this background noise.

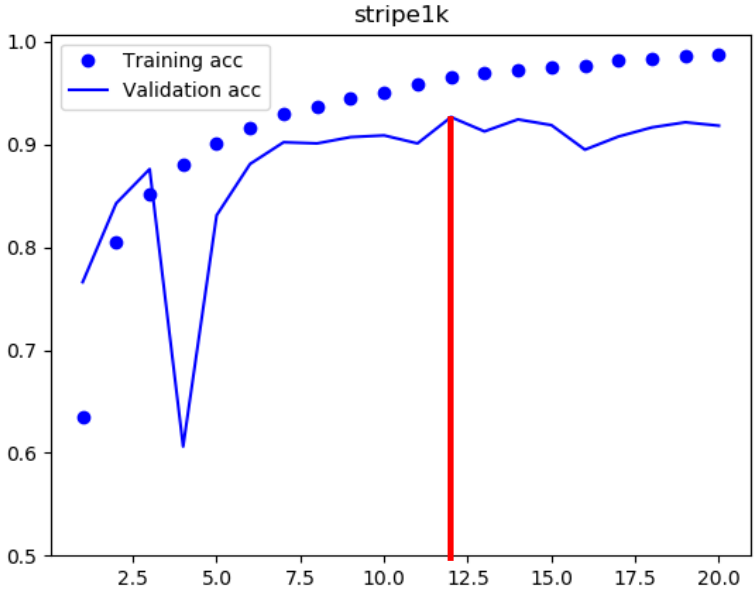


Figure 4.38: Training and validation accuracies for model trained with stripe1k dataset. The red line shows the point where the validation accuracy peaks, and the number of epochs used for final training

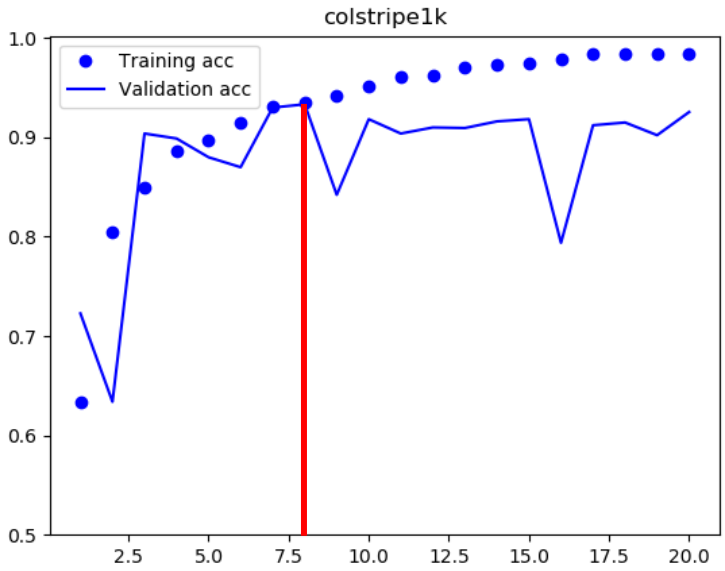


Figure 4.39: Training and validation accuracies for model trained with colstripe1k dataset. The red line shows the point where the validation accuracy peaks, and the number of epochs used for final training

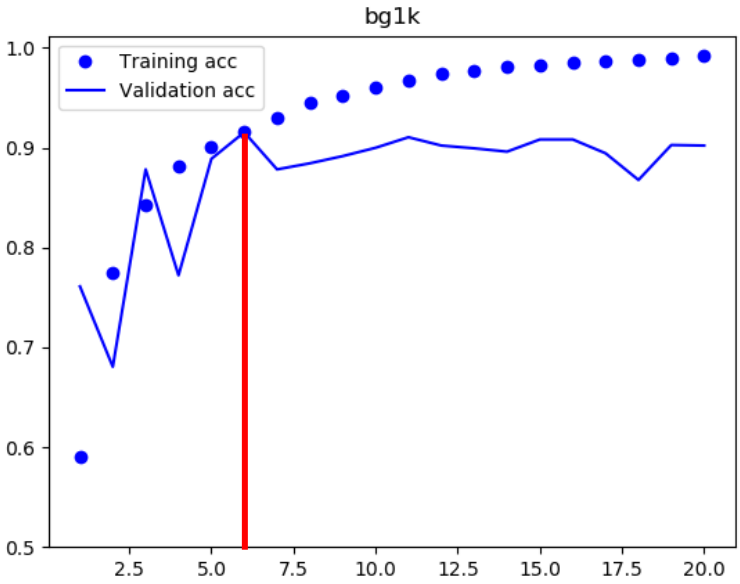


Figure 4.40: Training and validation accuracies for model trained with bg1k dataset. The red line shows the point where the validation accuracy peaks, and the number of epochs used for final training

The first instances of our model were trained using datasets containing 1000 images per category. We sent the models to train for 20 epochs. Figure 4.38, Figure 4.39 and Figure 4.40 show the training and validation accuracies for stripe1k and colstripe1k and bg1k respectively. In all three cases the validation accuracy appears to stop rising and peaks between 90% and 95%. Therefore, we did not train the models for any further epochs. The red lines on the graphs show where the validation accuracy peaks and determines the number of epochs that the models were trained for using all available data prior to testing.

Table 4.9: The number of epochs for final training and the final classification accuracies for all the datasets with 1000 images per category

Dataset	**Final training epochs**	**Final classification accuracy**
S1k	15	94.1%
Stripe1k	12	91.85%
Colstripe1k	8	94.08%
Bg1k	6	83.1%

Table 4.9 shows the number of epochs each of the 1k datasets was trained for prior to testing and the final classification accuracies. The S1k results are included for completeness. The confusion matrices in Figure 4.41 show that for all three datasets the models are performing as expected, with no misclassifications other than in the adjacent score categories. It is interesting to note that the colstripe1k model performs slightly better overall. This is probably due to individual patterns learned by the separate networks which cause the differences in accuracy. Unsurprisingly, the model trained using the most complex dataset, bg1k, has the lowest classification accuracy.

We also trained an instance of our model with 10,000 images per category for each type of data. In the same manner as for S10k, the models were trained for 10 epochs. The validation and train accuracies did still appear to be rising as can be seen in Figure 4.42, much like for S10k, however due to time constraints we decided to test each model at 10 epochs and not train further.

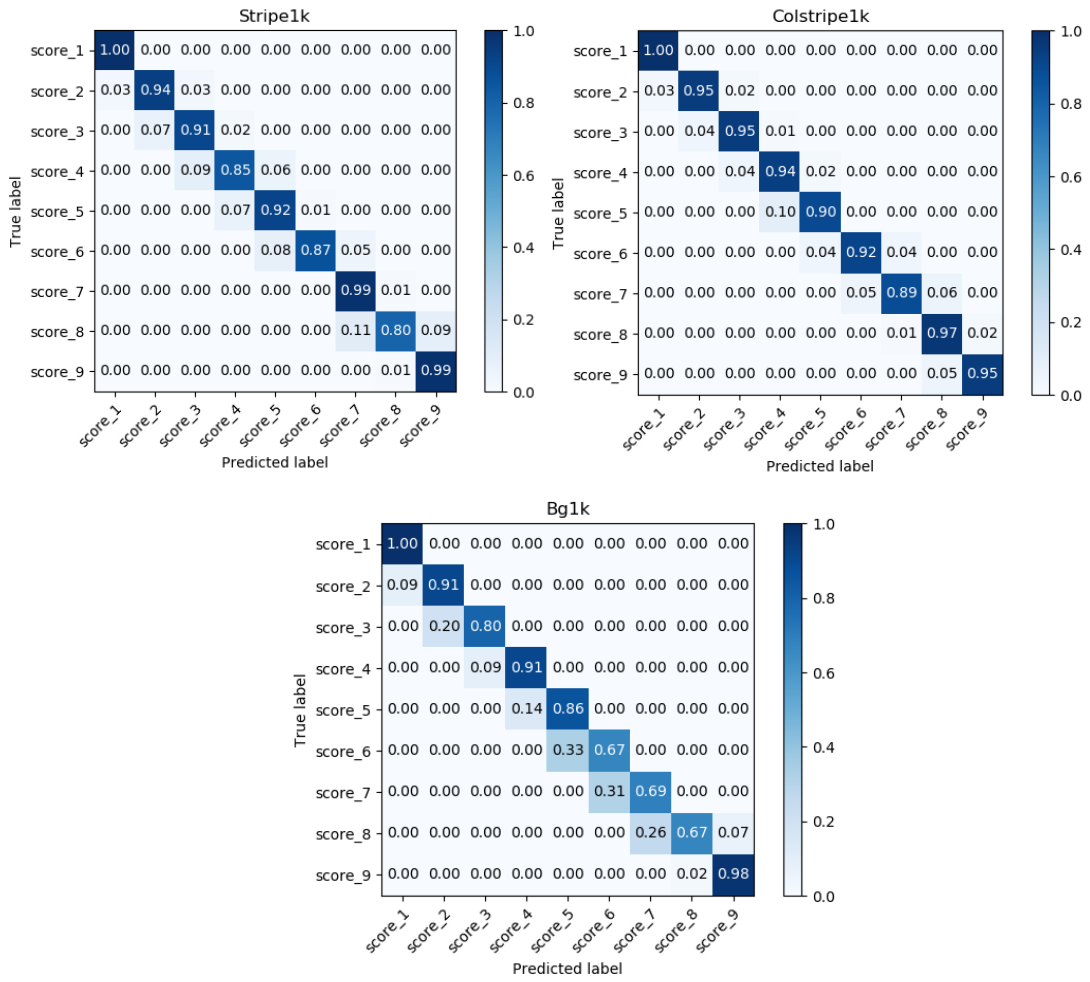


Figure 4.41: Confusion matrices for the models trained using stripe1k (top left), colstripe1k (top right) and bg1k (bottom)

The results for the 10k datasets can be seen in the confusion matrices in Figure 4.43. Unsurprisingly, each set of data produces high accuracy results throughout, which is a clear sign that using more data (where possible) will produce the better results. That being said, we also show that it is possible to get high accuracies with smaller datasets. In the case of relatively simple, and idealistic simulated data, 1000 images per category proves to be enough to hit accuracies of 90% and above. However, this number can be expected to be much higher as the data gets more complex and moving into real field data.

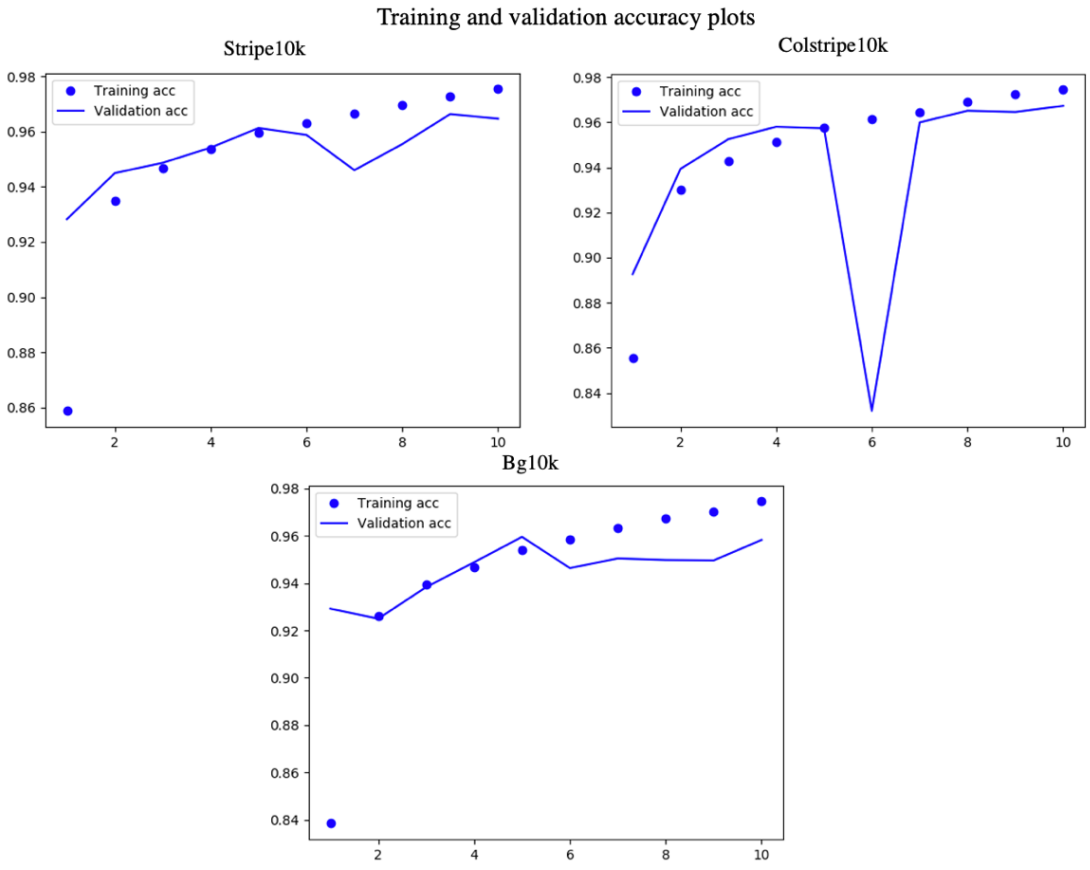


Figure 4.42: Training and validation accuracies for the Strike10k (top left), Colstripe10k (top right) and BG10k (bottom) datasets.

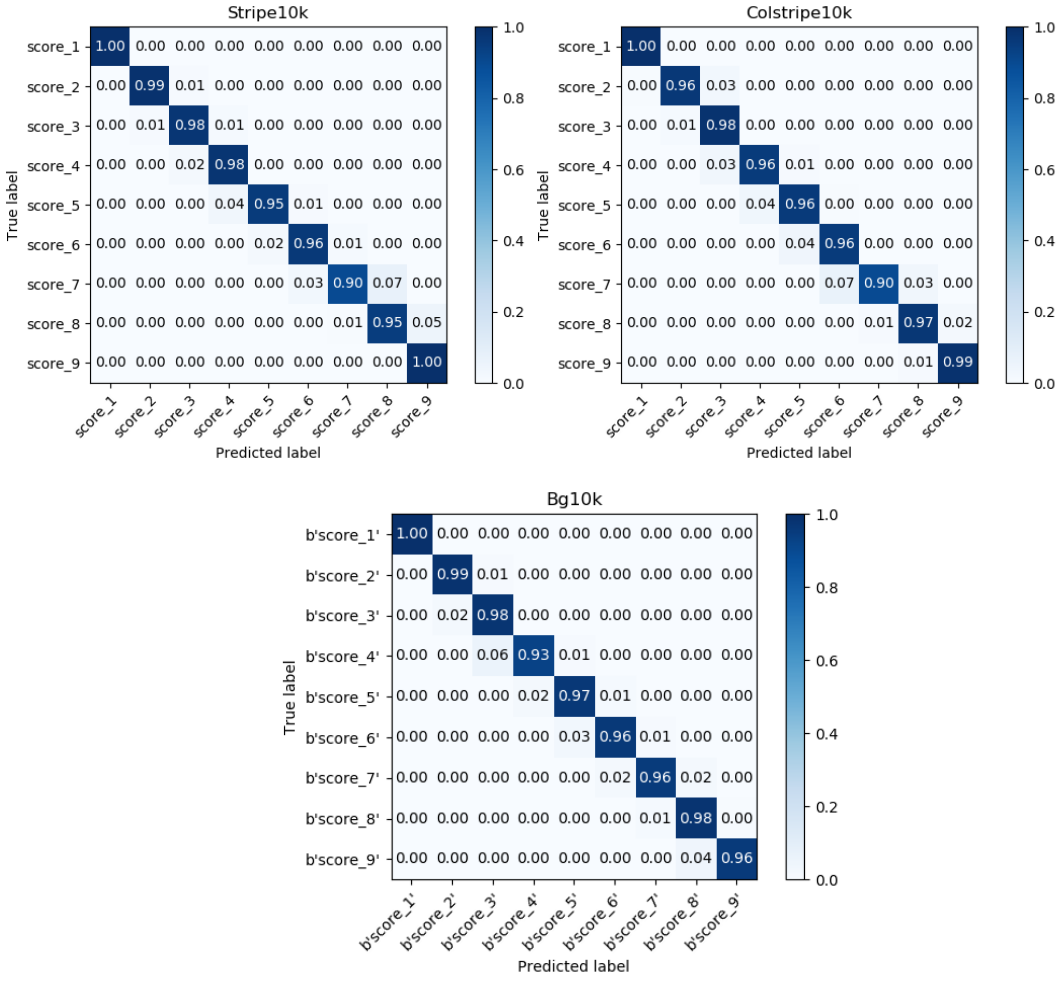


Figure 4.43: Confusion matrices for the models trained using stripe10k (top left), colstripe10k (top right) and bg10k (bottom)

4.5 Discussion

4.5.1 Deep learning for quantification with real field data

Due to the few studies for crop disease quantification which had been published at the time the planning for this work was taking place (Lin *et al*., 2019; Wang *et al*., 2019), we used our knowledge gained from chapter 3 and the desired outcomes of the project to guide our data collection effort. Instead of taking single leaf images, we photographed full plots, as would be expected from an automated scoring method. With our work we could determine the pitfalls and get an idea for the requirements for working with this kind of data.

Before even beginning our experiments with real field data for wheat disease quantification, it was clear that this was going to be a much more complex problem than disease classification. Where we took images that focused on a single leaf or plant in our classification dataset, here we had to image entire plots. This meant including messier data without a clear focal point in the image.

Before training of any model was even started, we could see some potential issues with the collected dataset. The first was the number of images that the dataset contained overall. In our classification dataset from chapter 3, we had 19,000 images spanning five categories. For the quantification of yellow rust, with 9 score categories, we had a dataset less than a third of the size. It would have been illogical to assume that the model would perform as well on a more complex problem, with much less training data.

The second issue was the imbalance of data across the 9 categories. For score 1 we had 1683 images, whereas at the other end of the scale in score 9 we only had 35, which in our view was far too few for the model to be able to learn accurate representations. There was a general trend of having fewer samples per category as the disease severity increased.

Upon collecting all the results from our experiments with this dataset, it was clear that no amount of fine tuning for these networks would provide the high accuracies that would be useful in the field. We had another look at the dataset with James Brown, who has many years of experience in scoring wheat diseases and found there were many images where the score assigned to the image did not match the symptoms visible in the photo. We do not believe this is due to misclassification of the plots themselves, but due to the angle of the photo or part of the plot photographed, the image does not match the score of the plot.

In future, a dataset collected in realistic field conditions would need to be significantly larger than the one used in this work and have a better balance across all categories. Furthermore, the method of collection and assigning of scores would need to be changed. Our work makes it clear that simply scoring a plot and photographing close to the time of scoring is not sufficient for getting images which accurately represent the plot score, and that a new method would need to be determined which aligns better with the methods used by pathologists. We discuss potential ways to tackle this problem in section 5.3.

4.5.2 Deep learning using simulated data for disease quantification

The hypothesis behind our experiments with the simulated datasets was that deep learning models would be able to quantify the amount of disease present in an image provided it was supplied with enough data of sufficient quality. Our first series of experiments provided an ‘ideal’ scenario where the images were very uniform without any background information or patterns within the infected areas. Clearly, these datasets are a very simplified version of different infection levels and do not fully represent what is found in the field. Their purpose was to be used as an initial proof of concept that it was possible to use deep learning models for a quantification problem.

Although the results using these ideal simulated images are very encouraging, it's important to note the potential pitfalls when trying to relate this back to the problem in the field. These datasets provide the models with very uniform images which, to the naked eye, do not show major differences between the images within a category. As a result, there is the possibility that our models are not in fact learning to count the number of infected pixels or calculate the percentage of infection, but instead are classifying the images based on the average grayscale. As the percentage of infection or number of white pixels increases, the overall colour of the image moves from black, through gradually lightening shades of grey, to almost white at the highest infection level.

Our further experiments went some way to proving that the model is actually classifying the images based on level of infection, rather than the average colour of the image. With the infected pixels distributed in clusters more representative of disease lesions, there was less opportunity for the model to differentiate the classes based on colour. This was especially true in the lower score classes where the level of infection was depicted by only a few lesions, so the infected areas were more concentrated. The addition of background noise also affected the overall colour interpretation of the images.

The results of our simulated data experiments provide a starting block for determining the minimum number of images required to achieve a certain level of accuracy for a given number of categories. With some further experiments and optimization of the codes, this methodology could be used to determine the best experimental design for network training. Simulated data of increasing complexity can be generated much quicker than collecting a full dataset of field images. Finding the amount of data required to reach an ideal accuracy would mean that the data collection in real conditions could be more guided, and the risk of collecting too little data reduced. This is especially important where data is collected over a growth period, like with crop breeding, as there is a long period to wait before more data can be gathered.

5 Discussion

Having presented our results in the previous chapters, we will here discuss their importance and contribution to the field. We will also mention the limitations of the work conducted, as well as suggested improvements and opportunities for future research. We will then conclude this thesis with a summary of the key findings.

5.1 Wheat disease classification

An important first step towards an automated, deep learning, scoring system for wheat disease is simply being able to identify and classify the diseases in realistic conditions. If this were not possible, then it is highly unlikely that any deep learning model would be able to quantify the amount of disease present.

Many previous studies have trained deep learning models to classify crop diseases. These studies typically use carefully curated data, taken in controlled conditions (Mohanty, Hughes and Salathé, 2016; Barbedo, 2018). Whilst some of these datasets are quite large, they do not contain the necessary variation in conditions which would be expected in field situations. It would be unrealistic to presume that a model trained on this kind of data would be able to perform when confronted with more complex field data. Conversely, studies which use realistic field images for training their models (Lu *et al*., 2017; Oppenheim *et al*., 2018), generally have small amounts of data which would not be able to cover the range of conditions expected and so would generalise to the available training data.

At the time this work took place, we were not aware of any openly available datasets of wheat disease images with the potential to overcome these limitations. We have since been made aware of the Wheat Fungi Disease Dataset (Genaev *et al*., 2021), however upon inspection we found there to be many potential misclassifications in several categories. Furthermore, the images in the dataset were much more varied,

from entire plots to single plants, which we did not consider was especially relevant to field scoring of diseases.

Our data collection effort produced a dataset with over 2000 images per category, taken in realistic growth conditions. The purpose was to capture the range of weather, light and growth stages of disease and plant which would be encountered in the field, so any trained model should hypothetically perform when taken out to test on new field data. We consider this dataset to be a valuable asset for the scientific community. Not only is it useful for training deep learning models to classify important UK wheat diseases, but it also has the potential to be used for training purposes. Breeders, pathologists and even farmers would be able to use the range of images to learn the symptoms of each of the disease, to allow them to be able to identify these diseases themselves. Until such time as an automated system is deployable, this would help people make decisions about their crops relating to disease management.

We trained a model using our collected dataset and found it was able to classify the images with an accuracy that rivals trained pathologists. It is interesting to note that the classification accuracy we achieve (97.05%) is very close to those from studies with much more controlled data. Amara *et al*., (2017) gained 98.61% classifying banana leaf diseases and Brahimi *et al*., (2017) achieved 99.18% classification accuracy on tomato leaf diseases, both taken from the Plant Village dataset. This suggests that it is not necessary to take images in controlled conditions, and that field data is sufficient for training models to classify diseases provided there are enough training examples. Our fully trained model has the potential to be used by farmers, agronomist, and breeders for identifying disease on their crops if it were deployed in a usable fashion (such as on a mobile application). This is something we'd have liked to develop; however limited time meant it wasn't possible at this time.

It is important to note that this model is trained to classify individual wheat diseases and does not deal with multiple simultaneous infections. In reality, there are often cases where two or more diseases occur at once on a single plot. There is now the opportunity to explore methods which can identify and classify multiple infections in

a single image. This would be particularly helpful for farmers without access to trained pathologists needing to determine what is infecting their crop in order to deploy any countermeasures.

Another limitation that is worth mentioning is that our data contains only visible infections, from the earliest visible signs through to late infection. As a result, our model would not be able to identify asymptomatic disease on the plants. The models we used learn features about the images in order to make a prediction about their contents. An asymptomatic leaf would appear the same as a healthy leaf in the visible spectrum, therefore we assumed that our model would classify all asymptomatic images as healthy. Some studies are making use of multiple spectrums by utilising hyperspectral imaging devices (Jin *et al*., 2018; Nagasubramanian *et al*., 2018; Wang *et al*., 2019). There may be features in spectrums other than the visible spectrum which could aid a model in classifying the disease before symptoms are visible to the naked eye. The downside of this method is that hyperspectral cameras are often very expensive, and so a model that requires one to use it would not be accessible to a large portion of the community.

Comparing our model to the performance of five pathologists showed that it is capable of classifying the diseases from images at least as well as a trained expert. This is an important factor for a model that would be used in the field to eliminate the need for a trained pathologist. We also found that the model classified the images much faster than the participants. This speed can also be increased by using GPU's and parallelisation much easier than it would be to hire more pathologists to do the same job.

5.2 Evaluation of model performance

It is not only important for any trained model to perform with high accuracy, but it also needs to be working as intended. For a model to be useful, it needs to be using the correct information to drive its classifications. In our case, the disease and leaf information are the important information which should be used to determine the

correct class. If a model were using an arbitrary background feature instead, it would not perform well given new data and it could be highlighting a bias in the dataset. We aimed to determine whether our model was using the correct information for fuelling its classifications.

Our first approach was to try using Grad-CAM (Selvaraju *et al*., 2020) to produce area important heatmaps for our images. This method produces a heatmap using the information contained in the last convolutional layer of a model, which can then be stretched over the input image. Areas which are important in driving the classification of the model would appear red and less important parts (such as background information) would appear blue. Unfortunately, due to the number and types of layer we use in our final model, the output feature map for the final convolutional layer was only 4x4 pixels, therefore the heatmap produced was also 4x4 pixels. When stretched over a much larger input image, these heatmaps did not provide any useful information about the important locations of the images.

We took a selection of images from our test set and placed a black mask over the important disease and leaf information. We hypothesized that covering the important information would result in a drop in performance, providing the model has learned to use the correct information for driving its classifications. When confronted with masked images, our model had a higher misclassification rate. Classifications were heavily biased towards mildew, with some bias towards healthy. It is likely that the bias towards healthy is due to the green background information being mistaken as healthy leaves. The mildew misclassifications are suggestive of an underlying issue with the training data, stemming from the different collection conditions in the mildew category. Ideally, we would have liked to collect more mildew images taken in the same conditions as the other categories images and re-trained the model. However, we were not able to do this due to there being little mildew in the field thanks to high levels of resistance in many UK wheat varieties.

This experiment provides important insights about the necessity of collecting data in the same capture conditions for all categories of a dataset. Covering a wide range of conditions is not enough on its own, this range needs to be as similar as possible

over all classes. This will eliminate biases in the data which the network could learn instead of focussing on the important information.

While the masked images caused the confusion that we expected for a model using the correct information to drive classifications, this experiment does not definitively prove that this is the case. Further investigation is required to show the features which are most important for making class predictions. One particular method we discovered which could be incredibly valuable for this issue is the activation atlas (Carter *et al*., 2019). The activation atlas is a method for visualising features in a network, which can show what features the model is using for making its classifications. It can show the feature space for each layer in the network, and the features in the final layers can be very informative. Figure 5.1 shows the activation atlas for the final layer in an example model trained using the ImageNet (Deng *et al*., 2009). The main grid shows the feature space for the chosen layer. Hovering over the individual features gives information about their attributions to a given class. In the far-left column one can select the individual classes to highlight the important features used for making classifications. The second column is where the different layers can be selected, to show how the features develop throughout the model. We believe that this could be a valuable tool for use with any classification model and would be fruitful for future research.

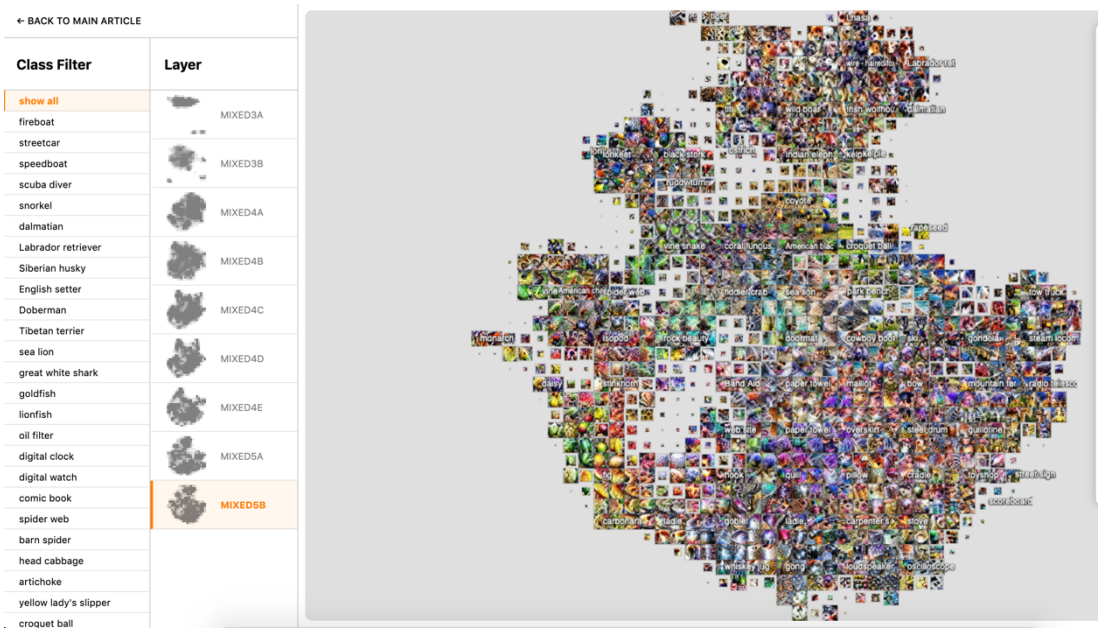


Figure 5.1: Example activation atlas screen taken from https://distill.pub/2019/activation-atlas/

5.3 Quantification of disease

An automated disease scoring method requires a model which is able to quantify the amount of disease present. It was always our intention to begin by training a model to quantify a single disease at a time, however we hoped to be able to do this for multiple separate diseases. Due to the global Covid-19 pandemic, we were more limited with the data we were able to collect for this part of the project. We chose to focus on collecting images of yellow rust infections, due to the availability of yellow rust trials which would be scored over the growth season.

We were encouraged by the work of Mi *et al*., (2020), where they were able to score images of yellow rust leaves with 97.99%. Their work used images of single leaves, which were cropped to remove the majority of background information. We hoped to build on this work using our collection of full wheat plot images, with the hope that we could eliminate the need for additional pre-processing such as cropping.

We collected a dataset of over 5000 images, sorted into 1 – 9 categories that represent the amount of disease present, which was used to multiple deep learning networks in three different configurations. Unfortunately, the results of these experiments revealed that it was unlikely we would be able to train a model to high accuracies using this data alone.

This work highlighted multiple issues with our quantification dataset. The first, and most obvious, was the clear imbalance of data across the categories. The second issue to note is the size of the dataset as a whole. Finally, we show that simply taking an image of a scored plot does not necessarily mean that the image is going to show the same score. Prior to fixing the amount and imbalance of the data, it is first important to fix the content. For use in the field, an image taken would need to be representative of the score of the plot to ensure that the plot as a whole is scored correctly. This presents a challenge thanks to the possibility of patchy infections. In reality a pathologist can look at a plot and be able to give the average score for that plot, however in this work the model has to decide based on one image which may not show all the available information.

Overcoming these problems is going to require a more substantial data collection effort. One method we suggest may work is to use multiple images per plot as input for a network, then the model could use all the data to calculate an average plot score in a similar way to a pathologist. This method could motivate a move from smartphone images collection (and later deployment of a model) to the use of drones. Drones have already been used in studies for remote sensing in agriculture (Yang *et al*., 2017; Inoue, 2020). It would be easy for a drone to move over a plot and collect for example 10 images before moving to the next plot, thus easily collecting data for many plots without much human interaction. It would also mean that a future model could also be deployed on a drone, so eliminating the need for someone to take the model out manually. Clearly, there is much opportunity for building on our work and using our findings as a guide for progressing with the problem of automated disease scoring.

We created multiple simulated datasets of increasing complexity for determining the viability of scoring disease on this scale given more controlled conditions. Our results suggest that a deep learning model would be able to handle quantifying disease in this way, provided that there was enough informative data for training. These experiments can be used as a first assessment of the number of images required to reach a certain level of accuracy. The methods we define could be used to determine a minimum number of images required per category for a desired accuracy with a given number of categories. It could be a powerful means of evaluating the best experimental design for network training.

5.4 Conclusion

The aim of this thesis was to evaluate the viability of using deep learning models for the identification and quantification of wheat diseases using images taken in complex, realistic growth conditions. This work would provide a starting point for eventually producing an automated method to aid breeders in scoring wheat disease on their plants.

We collected a large dataset of wheat disease images, including four disease categories and one healthy category. This dataset is used to train various deep learning models with the aim of finding the best performing architecture in terms of classification accuracy. The accuracy results of over 97% show that deep learning models are capable of handling images with complex background information, such as would be found in realistic growth conditions. When compared with the performance of five human participants, our model performs at least as well as expert pathologists. Masking images highlights the need for collecting images of the same type across all categories to ensure accurate results.

Another dataset of yellow rust plot images was collected, sorted into disease score categories to train a model in quantification of disease. This dataset was significantly smaller, yet contained more categories, than the classification dataset and our experiments suggest the need for a much larger training set of images, especially with regard to such a complex problem. Creating simulated images provided a foundation for determining the ideal amount of data necessary for achieving a desired classification accuracy.

This thesis demonstrates the potential for using deep learning in the field to classify and quantify diseases on wheat. The results of the classification model are highly encouraging and open the door for expanding the problem to other diseases, crops, and potentially multiple, simultaneous infections. The data collection efforts highlighted various issues with regards to datasets for quantification. Further work is required to overcome these issues and move closer towards an automated scoring system.

Data Availability

The dataset of 999 images used in the experiment with pathologists is freely available from https://zenodo.org/record/7573133. Due to the size of the full datasets and agreements with the associated companies, these can be requested from james.brown@jic.ac.uk,

References

Abadi, M. *et al*. (2015) 'TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems', p. 19.

AHDB (2020) 'The encyclopaedia of cereal diseases'. AHDB Cereals & Oilseeds. Available at: https://ahdb.org.uk/knowledge-library/encyclopaedia-of-cereal-diseases.

Albattah, W. *et al*. (2022) 'A novel deep learning method for detection and classification of plant diseases', *Complex & Intelligent Systems*, 8(1), pp. 507–524. Available at: https://doi.org/10.1007/s40747-021-00536-1.

Ali, S. *et al*. (2011) 'A rapid genotyping method for an obligate fungal pathogen, Puccinia striiformis f.sp. tritici, based on DNA extraction from infected leaf and Multiplex PCR genotyping', *BMC research notes*, 4, p. 240. Available at: https://doi.org/10.1186/1756-0500-4-240.

Amara, J., Bouaziz, B. and Algergawy, A. (2017) 'A Deep Learning-based Approach for Banana Leaf Diseases Classification', in, pp. 79–88.

Baenziger, P. (2016) 'Wheat Breeding and Genetics', in *Reference Module in Food Science*. Available at: https://doi.org/10.1016/B978-0-08-100596-5.03001-8.

Barbedo, J. (2018) 'Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification', *Computers and Electronics in Agriculture*, 153, pp. 46–53. Available at: https://doi.org/10.1016/j.compag.2018.08.013.

Bolton, M.D., Kolmer, J.A. and Garvin, D.F. (2008) 'Wheat leaf rust caused by Puccinia triticina', *Molecular Plant Pathology*, 9(5), pp. 563–575. Available at: https://doi.org/10.1111/j.1364-3703.2008.00487.x.

Boulent, J. *et al*. (2019) 'Convolutional Neural Networks for the Automatic Identification of Plant Diseases', *Frontiers in Plant Science*, 10, p. 941. Available at: https://doi.org/10.3389/fpls.2019.00941.

Brahimi, M., Boukhalfa, K. and Moussaoui, A. (2017) 'Deep Learning for Tomato Diseases: Classification and Symptoms Visualization', *Applied Artificial Intelligence*, 31(4), pp. 299–315. Available at: https://doi.org/10.1080/08839514.2017.1315516.

Brown, J.K. (2015) 'Durable resistance of crops to disease: a Darwinian perspective', *Annual review of phytopathology*, 53, pp. 513–539.

Brown, J.K.M. (2021) 'Achievements in breeding cereals with durable disease resistance in Northwest Europe', in formerly Curtin University, Australia and R. Oliver (eds) *Burleigh Dodds Series in Agricultural Science*. Burleigh Dodds Science Publishing, pp. 825–872. Available at: https://doi.org/10.19103/AS.2021.0092.39.

Brown, J.K.M. and Wulff, B.B.H. (2022) 'Diversifying the menu for crop powdery mildew resistance', *Cell*, 185(5), pp. 761–763. Available at: https://doi.org/10.1016/j.cell.2022.02.003.

Carter, S. *et al*. (2019) 'Activation Atlas', *Distill*, 4(3), p. e15. Available at: https://doi.org/10.23915/distill.00015.

Chen, S. *et al*. (2021) 'An Approach for Rice Bacterial Leaf Streak Disease Segmentation and Disease Severity Estimation', *Agriculture*, 11(5), p. 420. Available at: https://doi.org/10.3390/agriculture11050420.

Chen, W. *et al*. (2014) 'Wheat stripe (yellow) rust caused by Puccinia striiformis f. sp. tritici', *Molecular Plant Pathology*, 15(5), pp. 433–446. Available at: https://doi.org/10.1111/mpp.12116.

Chollet, F. (2017a) *Deep Learning with Python*. Manning Publications.

Chollet, F. (2017b) 'Xception: Deep Learning with Depthwise Separable Convolutions', in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI: IEEE, pp. 1800–1807. Available at: https://doi.org/10.1109/CVPR.2017.195.

Chollet, F. and others (2015) 'Keras'. Github. Available at: https://github.com/fchollet/keras.

Cowger, C. *et al*. (2020) 'Role of Effector-Sensitivity Gene Interactions and Durability of Quantitative Resistance to Septoria Nodorum Blotch in Eastern U.S. Wheat', *Frontiers in Plant Science*, 11. Available at: https://www.frontiersin.org/articles/10.3389/fpls.2020.00155 (Accessed: 26 September 2022).

Day, L. *et al*. (2006) 'Wheat-gluten uses and industry needs', *Trends in Food Science & Technology*, 17(2), pp. 82–90. Available at: https://doi.org/10.1016/j.tifs.2005.10.003.

Deng, J. *et al*. (2009) 'ImageNet: A large-scale hierarchical image database', in *2009 IEEE Conference on Computer Vision and Pattern Recognition. 2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. Available at: https://doi.org/10.1109/CVPR.2009.5206848.

Divyanth, L.G., Ahmad, A. and Saraswat, D. (2023) 'A two-stage deep-learning based segmentation model for crop disease quantification based on corn field imagery', *Smart Agricultural Technology*, 3, p. 100108. Available at: https://doi.org/10.1016/j.atech.2022.100108.

Dubin, H.J. and Duveiller, E. (2011) 'Fungal, bacterial and nematode diseases of wheat: breeding for resistance and other control measures.', in *The World Wheat Book, A History of Wheat Breeding*. Angus, W., Bonjean, A. and van Ginkel, M., pp. 1131–1191.

European Commission (no date) *Neonicotinoids*. Available at: https://food.ec.europa.eu/plants/pesticides/approval-active-substances/renewal-approval/neonicotinoids_en (Accessed: 26 September 2022).

Feldman, M. (1995) 'Wheats', in Smartt, J. and Simmonds, N. W., *Evolution of crop plants*. Longman Scientific and Technical, Harlow, UK, pp. 185–192.

Ferentinos, K. (2018) 'Deep learning models for plant disease detection and diagnosis', *Computers and Electronics in Agriculture*, 145, pp. 311–318. Available at: https://doi.org/10.1016/j.compag.2018.01.009.

Genaev, M.A. *et al*. (2021) 'Image-Based Wheat Fungi Diseases Identification by Deep Learning', *Plants*, 10(8), p. 1500. Available at: https://doi.org/10.3390/plants10081500.

Glorot, X., Bordes, A. and Bengio, Y. (2011) 'Deep Sparse Rectifier Neural Networks', in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, pp. 315–323. Available at: https://proceedings.mlr.press/v15/glorot11a.html (Accessed: 18 May 2023).

Godfray, H.C.J. *et al*. (2010) 'Food Security: The Challenge of Feeding 9 Billion People', *Science*, 327(5967), pp. 812–818. Available at: https://doi.org/10.1126/science.1185383.

Goutte, C. and Gaussier, E. (2005) 'A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation', in D.E. Losada and J.M. Fernández-Luna (eds) *Advances in Information Retrieval*. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science), pp. 345–359. Available at: https://doi.org/10.1007/978-3-540-31865-1_25.

Goyeau, H. *et al*. (2006) 'Distribution of pathotypes with regard to host cultivars in French wheat leaf rust populations', *Phytopathology*, 96(3), pp. 264–273. Available at: https://doi.org/10.1094/PHYTO-96-0264.

Hancock, J.F. (2004) *Plant Evolution and the Origin of Crop Species*. CABI Pub.

Haque, M.A. *et al*. (2022) 'Deep learning-based approach for identification of diseases of maize crop', *Scientific Reports*, 12(1), p. 6334. Available at: https://doi.org/10.1038/s41598-022-10140-z.

Hardwick, N.V., Jones, D.R. and Slough, J.E. (2001) 'Factors affecting diseases of winter wheat in England and Wales, 1989–98', *Plant Pathology*, 50(4), pp. 453–462. Available at: https://doi.org/10.1046/j.1365-3059.2001.00596.x.

Hartley, M. (no date) 'qtagger'. Available at: https://github.com/JIC-Image-Analysis/qtagger.

He, K. *et al*. (2016) 'Deep Residual Learning for Image Recognition', in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. *2016 IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778. Available at: https://doi.org/10.1109/CVPR.2016.90.

Hinton, G. and Tieleman, T. (2012) 'Lecture 6.5 - rmsprop, COURSERA: Neural Networks for Machine Learning'.

Hochreiter, S. (1998) 'The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions', *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02), pp. 107–116. Available at: https://doi.org/10.1142/S0218488598000094.

Hovmøller, M.S. *et al*. (2016) 'Replacement of the European wheat yellow rust population by new races from the centre of diversity in the near-Himalayan region', *Plant Pathology*, 65(3), pp. 402–411. Available at: https://doi.org/10.1111/ppa.12433.

Howard, A.G. *et al*. (2017) 'MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications', *arXiv:1704.04861 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1704.04861 (Accessed: 3 March 2021).

Huang, G. *et al*. (2018) 'Densely Connected Convolutional Networks', *arXiv:1608.06993 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1608.06993 (Accessed: 28 September 2021).

Hughes, D.P. and Salathe, M. (2016) 'An open access repository of images on plant health to enable the development of mobile disease diagnostics', *arXiv:1511.08060 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1511.08060 (Accessed: 5 March 2021).

Inoue, Y. (2020) 'Satellite- and drone-based remote sensing of crops and soils for smart farming – a review', *Soil Science and Plant Nutrition*, 66(6), pp. 798–810. Available at: https://doi.org/10.1080/00380768.2020.1738899.

Ioffe, S. and Szegedy, C. (2015) 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift', *arXiv:1502.03167 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1502.03167 (Accessed: 21 May 2021).

Jia, Y. *et al*. (2014) 'Caffe: Convolutional Architecture for Fast Feature Embedding', *arXiv:1408.5093 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1408.5093 (Accessed: 27 September 2021).

Jin, X. *et al*. (2018) 'Classifying Wheat Hyperspectral Pixels of Healthy Heads and Fusarium Head Blight Disease Using a Deep Neural Network in the Wild Field', *Remote Sensing*, 10(3), p. 395. Available at: https://doi.org/10.3390/rs10030395.

Kingma, D.P. and Ba, J. (2017) 'Adam: A Method for Stochastic Optimization'. arXiv. Available at: https://doi.org/10.48550/arXiv.1412.6980.

Krizhevsky, A. (2014) 'One weird trick for parallelizing convolutional neural networks', *arXiv:1404.5997 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1404.5997 (Accessed: 27 September 2021).

Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012) 'ImageNet classification with deep convolutional neural networks', in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. Red Hook, NY, USA: Curran Associates Inc. (NIPS'12), pp. 1097–1105.

Kulkarni, P. *et al*. (2021) *Plant Disease Detection Using Image Processing and Machine Learning*.

Lecun, Y. *et al*. (1998) 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE*, 86(11), pp. 2278–2324. Available at: https://doi.org/10.1109/5.726791.

LeCun, Y. *et al*. (1999) 'Object recognition with gradient-based learning', in *Shape, Contour and Grouping in Computer Vision. International Workshop on Shape, Contour and Grouping in Computer Vision*, Springer Verlag, pp. 319–345. Available at: https://doi.org/10.1007/3-540-46805-6_19.

Li, L., Zhang, S. and Wang, B. (2021) 'Plant Disease Detection and Classification by Deep Learning—A Review', *IEEE Access*, 9, pp. 56683–56698. Available at: https://doi.org/10.1109/ACCESS.2021.3069646.

Lin, K. *et al*. (2019) 'Deep Learning-Based Segmentation and Quantification of Cucumber Powdery Mildew Using Convolutional Neural Network', *Frontiers in Plant Science*, 10, p. 155. Available at: https://doi.org/10.3389/fpls.2019.00155.

Liu, B. *et al*. (2018) 'Identification of Apple Leaf Diseases Based on Deep Convolutional Neural Networks', *Symmetry*, 10(1), p. 11. Available at: https://doi.org/10.3390/sym10010011.

Liu, J. and Wang, X. (2021) 'Plant diseases and pests detection based on deep learning: a review', *Plant Methods*, 17(1), p. 22. Available at: https://doi.org/10.1186/s13007-021-00722-9.

Liu, M. and Hambleton, S. (2010) 'Taxonomic study of stripe rust, Puccinia striiformis sensu lato, based on molecular and morphological evidence', *Fungal biology*, 114, pp. 881–99. Available at: https://doi.org/10.1016/j.funbio.2010.08.005.

Lu, Y. *et al*. (2017) 'Identification of rice diseases using deep convolutional neural networks', *Neurocomputing*, 267, pp. 378–384. Available at: https://doi.org/10.1016/j.neucom.2017.06.023.

Lydia, A. and Francis, S. (2019) 'Adagrad - An Optimizer for Stochastic Gradient Descent', Volume 6, pp. 566–568.

Mi, Z. *et al*. (2020) 'Wheat Stripe Rust Grading by Deep Learning With Attention Mechanism and Images From Mobile Devices', *Frontiers in Plant Science*, 11. Available at: https://doi.org/10.3389/fpls.2020.558126.

Mohanty, S.P., Hughes, D.P. and Salathé, M. (2016) 'Using Deep Learning for Image-Based Plant Disease Detection', *Frontiers in Plant Science*, 7. Available at: https://doi.org/10.3389/fpls.2016.01419.

Nagasubramanian, K. *et al*. (2018) 'Explaining hyperspectral imaging based plant disease identification: 3D CNN and saliency maps'.

Oppenheim, D. *et al*. (2018) 'Using Deep Learning for Image-Based Potato Tuber Disease Detection', *Phytopathology®*, 109(6), pp. 1083–1087. Available at: https://doi.org/10.1094/PHYTO-08-18-0288-R.

Pandian, J.A. *et al*. (2022) 'Plant Disease Detection Using Deep Convolutional Neural Network', *Applied Sciences*, 12(14), p. 6982. Available at: https://doi.org/10.3390/app12146982.

Pedregosa, F. *et al*. (2011) 'Scikit-learn: Machine Learning in Python', *Journal of Machine Learning Research*, 12(85), pp. 2825–2830.

Perez, L. and Wang, J. (2017) 'The Effectiveness of Data Augmentation in Image Classification using Deep Learning'. arXiv. Available at: https://doi.org/10.48550/arXiv.1712.04621.

Rangarajan, A.K., Purushothaman, R. and Ramesh, A. (2018) 'Tomato crop disease classification using pre-trained deep learning algorithm', *Procedia Computer Science*, 133, pp. 1040–1047. Available at: https://doi.org/10.1016/j.procs.2018.07.070.

Refaeilzadeh, P., Tang, L. and Liu, H. (2009) 'Cross-Validation', in L. LIU and M.T. ÖZSU (eds) *Encyclopedia of Database Systems*. Boston, MA: Springer US, pp. 532–538. Available at: https://doi.org/10.1007/978-0-387-39940-9_565.

Russakovsky, O. *et al*. (2015) 'ImageNet Large Scale Visual Recognition Challenge', *arXiv:1409.0575 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1409.0575 (Accessed: 21 September 2021).

Russell, B.C., Torralba, A. and Murphy, K.P. (2008) 'LabelMe: A database and web-based tool for image annotation', 77, pp. 157–153.

Rutkoski, J.E., Krause, M.R. and Sorrells, M.E. (2022) 'Breeding Methods: Line Development', in M.P. Reynolds and H.-J. Braun (eds) *Wheat Improvement: Food Security in a Changing Climate*. Cham: Springer International Publishing, pp. 69–82. Available at: https://doi.org/10.1007/978-3-030-90673-3_5.

Saleem, M.H. *et al*. (2020) 'Image-Based Plant Disease Identification by Deep Learning Meta-Architectures', *Plants*, 9(11), p. 1451. Available at: https://doi.org/10.3390/plants9111451.

Saunders, D.G.O., Pretorius, Z.A. and Hovmøller, M.S. (2019) 'Tackling the re-emergence of wheat stem rust in Western Europe', *Communications Biology*, 2(1), pp. 1–3. Available at: https://doi.org/10.1038/s42003-019-0294-9.

Savary, S. *et al*. (2019) 'The global burden of pathogens and pests on major food crops', *Nature Ecology & Evolution*, 3(3), pp. 430–439. Available at: https://doi.org/10.1038/s41559-018-0793-y.

Schirrmann, M. *et al*. (2021) 'Early Detection of Stripe Rust in Winter Wheat Using Deep Residual Neural Networks', *Frontiers in Plant Science*, 12, p. 475. Available at: https://doi.org/10.3389/fpls.2021.469689.

Selvaraju, R.R. *et al*. (2020) 'Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization', *International Journal of Computer Vision*, 128(2), pp. 336–359. Available at: https://doi.org/10.1007/s11263-019-01228-7.

Sermanet, P. *et al*. (2013) 'OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks', *International Conference on Learning Representations (ICLR) (Banff)* [Preprint].

Shewry, P.R. *et al*. (2002) 'The structure and properties of gluten: an elastic protein from wheat grain.', *Philosophical Transactions of the Royal Society B: Biological Sciences*, 357(1418), pp. 133–142. Available at: https://doi.org/10.1098/rstb.2001.1024.

Shewry, P.R. (2009) 'Wheat', *Journal of Experimental Botany*, 60(6), pp. 1537–1553. Available at: https://doi.org/10.1093/jxb/erp058.

Shewry, P.R. and Hey, S.J. (2015) 'The contribution of wheat to human diet and health', *Food and Energy Security*, 4(3), pp. 178–202. Available at: https://doi.org/10.1002/fes3.64.

Simonyan, K. and Zisserman, A. (2015) 'Very Deep Convolutional Networks for Large-Scale Image Recognition', *arXiv:1409.1556 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1409.1556 (Accessed: 3 March 2021).

Singh, D. *et al*. (2020) 'PlantDoc: A Dataset for Visual Plant Disease Detection', in, pp. 249–253. Available at: https://doi.org/10.1145/3371158.3371196.

Sladojevic, S. *et al*. (2016) 'Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification', *Computational Intelligence and Neuroscience*, 2016, p. e3289801. Available at: https://doi.org/10.1155/2016/3289801.

Srivastava, N. *et al*. (2014) 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting', *Journal of Machine Learning Research*, 15(56), pp. 1929–1958.

Strange, R.N. and Scott, P.R. (2005) 'Plant Disease: A Threat to Global Food Security', *Annual Review of Phytopathology*, 43(1), pp. 83–116. Available at: https://doi.org/10.1146/annurev.phyto.43.113004.133839.

Szegedy, C. *et al*. (2014) 'Going Deeper with Convolutions', *arXiv:1409.4842 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1409.4842 (Accessed: 28 May 2021).

Szegedy, C. *et al*. (2015) 'Rethinking the Inception Architecture for Computer Vision'. arXiv. Available at: https://doi.org/10.48550/arXiv.1512.00567.

Szegedy, C., Ioffe, S., *et al*. (2016) 'Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning', *arXiv:1602.07261 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1602.07261 (Accessed: 27 September 2021).

Szegedy, C., Vanhoucke, V., *et al*. (2016) 'Rethinking the Inception Architecture for Computer Vision', in. Available at: https://doi.org/10.1109/CVPR.2016.308.

Tiley, A.M.M., Foster, G.D. and Bailey, A.M. (2018) 'Exploring the Genetic Regulation of Asexual Sporulation in Zymoseptoria tritici', *Frontiers in Microbiology*, 9. Available at: https://www.frontiersin.org/articles/10.3389/fmicb.2018.01859 (Accessed: 26 September 2022).

Too, E.C. *et al*. (2019) 'A comparative study of fine-tuning deep learning models for plant disease identification', *Computers and Electronics in Agriculture*, 161, pp. 272–279. Available at: https://doi.org/10.1016/j.compag.2018.03.032.

Wang, D. *et al*. (2019) 'Early Detection of Tomato Spotted Wilt Virus by Hyperspectral Imaging and Outlier Removal Auxiliary Classifier Generative Adversarial Nets (OR-AC-GAN)', *Scientific Reports*, 9(1), p. 4377. Available at: https://doi.org/10.1038/s41598-019-40066-y.

Wang, F. *et al*. (2017) 'Residual Attention Network for Image Classification'. arXiv. Available at: https://doi.org/10.48550/arXiv.1704.06904.

Wang, G., Sun, Y. and Wang, J. (2017) 'Automatic Image-Based Plant Disease Severity Estimation Using Deep Learning', *Computational Intelligence and Neuroscience*, 2017, pp. 1–8. Available at: https://doi.org/10.1155/2017/2917536.

Yamashita, R. *et al*. (2018) 'Convolutional neural networks: an overview and application in radiology', *Insights into Imaging*, 9(4), pp. 611–629. Available at: https://doi.org/10.1007/s13244-018-0639-9.

Yang, G. *et al*. (2017) 'Unmanned Aerial Vehicle Remote Sensing for Field-Based Crop Phenotyping: Current Status and Perspectives', *Frontiers in Plant Science*, 8. Available at: https://www.frontiersin.org/articles/10.3389/fpls.2017.01111 (Accessed: 30 September 2022).

Zhang, S., Huang, W. and Zhang, C. (2018) 'Three-Channel Convolutional Neural Networks for Vegetable Leaf Disease Recognition', *Cognitive Systems Research*, 53. Available at: https://doi.org/10.1016/j.cogsys.2018.04.006.

Appendix 1

The residual attention model code by Deontae Pharr:

```
from keras.layers import Input, Conv2D, Lambda, MaxPool2D,
UpSampling2D, AveragePooling2D, ZeroPadding2D
from keras.layers import Activation, Flatten, Dense, Add, Multiply,
BatchNormalization, Dropout

from keras.models import Model

# Todo: Make scalable/all-encompassing
class ResidualAttentionNetwork():

    def __init__(self, input_shape, n_classes, activation, p=1, t=2,
r=1):
        self.input_shape = input_shape
        self.n_classes = n_classes
        self.activation = activation
        self.p = p
        self.t = t
        self.r = r

    def build_model(self):
         # Initialize a Keras Tensor of input_shape
        input_data = Input(shape=self.input_shape)

        # Initial Layers before Attention Module

        # Doing padding because I'm having trouble with img dims
that are <= 28
        if self.input_shape[0] <= 28 or self.input_shape[1] <= 28:
            x_dim_inc = (32 - self.input_shape[0]) // 2
            y_dim_inc = (32 - self.input_shape[1]) // 2

            # Pad the input data to 32x32
            padded_input_data = ZeroPadding2D( (x_dim_inc,
y_dim_inc) )(input_data)
            conv_layer_1 = Conv2D(filters=32,
                                  kernel_size=(3,3),
                                  strides=(1,1),
                                  padding='same')(padded_input_data)
        else:
            conv_layer_1 = Conv2D(filters=32,
                                  kernel_size=(3,3),
                                  strides=(1,1),
                                  padding='same')(input_data)

        max_pool_layer_1 = MaxPool2D(pool_size=(2, 2),
                                     strides=(2, 2),
                                     padding='same')(conv_layer_1)

        # Residual Unit then Attention Module #1
        res_unit_1 = self.residual_unit(max_pool_layer_1,
filters=[32, 64, 128])
```

```
        att_mod_1 = self.attention_module(res_unit_1, filters=[32,
64, 128])

        # Residual Unit then Attention Module #2
        res_unit_2 = self.residual_unit(att_mod_1, filters=[32, 64,
128])
        att_mod_2 = self.attention_module(res_unit_2, filters=[32,
64, 128])

        # Residual Unit then Attention Module #3
        res_unit_3 = self.residual_unit(att_mod_2, filters=[32, 64,
128])
        att_mod_3 = self.attention_module(res_unit_3, filters=[32,
64, 128])

        # Ending it all
        res_unit_end_1 = self.residual_unit(att_mod_3, filters=[32,
32, 64])
        res_unit_end_2 = self.residual_unit(res_unit_end_1,
filters=[32, 32, 64])
        res_unit_end_3 = self.residual_unit(res_unit_end_2,
filters=[32, 32, 64])
        res_unit_end_4 = self.residual_unit(res_unit_end_3,
filters=[32, 32, 64])

        # Avg Pooling
        avg_pool_layer = AveragePooling2D(pool_size=(2, 2),
strides=(2, 2))(res_unit_end_4)

        # Flatten the data
        flatten_op = Flatten()(avg_pool_layer)

        # FC Layers for prediction
        fully_connected_layer_1 = Dense(256,
activation='relu')(flatten_op)
        dropout_layer_1 = Dropout(0.5)(fully_connected_layer_1)
        fully_connected_layer_2 = Dense(256,
activation='relu')(dropout_layer_1)
        dropout_layer_2 = Dropout(0.5)(fully_connected_layer_2)
        fully_connected_layer_3 = Dense(256,
activation='relu')(dropout_layer_2)
        dropout_layer_3 = Dropout(0.5)(fully_connected_layer_3)
        fully_connected_layer_last = Dense(self.n_classes,
activation=self.activation)(dropout_layer_3)

        # Fully constructed model
        model = Model(inputs=input_data,
outputs=fully_connected_layer_last)

        return model

    # Pre-Activation Identity ResUnit Bottleneck Architecture
    def residual_unit(self, residual_input_data, filters):

        # Hold input_x here for later processing
        identity_x = residual_input_data

        filter1,filter2,filter3 = filters

        # Layer 1
        batch_norm_op_1 = BatchNormalization()(residual_input_data)
```

```
        activation_op_1 = Activation('relu')(batch_norm_op_1)
        conv_op_1 = Conv2D(filters=filter1,
                           kernel_size=(1,1),
                           strides=(1,1),
                           padding='same')(activation_op_1)

        # Layer 2
        batch_norm_op_2 = BatchNormalization()(conv_op_1)
        activation_op_2 = Activation('relu')(batch_norm_op_2)
        conv_op_2 = Conv2D(filters=filter2,
                           kernel_size=(3,3),
                           strides=(1,1),
                           padding='same')(activation_op_2)

        # Layer 3
        batch_norm_op_3 = BatchNormalization()(conv_op_2)
        activation_op_3 = Activation('relu')(batch_norm_op_3)
        conv_op_3 = Conv2D(filters=filter3,
                           kernel_size=(1,1),
                           strides=(1,1),
                           padding='same')(activation_op_3)

        # Element-wise Addition
        if identity_x.shape[-1].value != conv_op_3.shape[-1].value:
            filter_n = conv_op_3.shape[-1].value

            identity_x = Conv2D(filters=filter_n,
                                kernel_size=(1,1),
                                strides=(1,1),
                                padding='same')(identity_x)

        output = Add()([identity_x, conv_op_3])

        return output

    def attention_module(self, attention_input_data, filters):
        # Send input_x through #p residual_units
        p_res_unit_op_1 = attention_input_data
        for _ in range(self.p):
            p_res_unit_op_1 = self.residual_unit(p_res_unit_op_1,
filters=filters)

        # Perform Trunk Branch Operation
        trunk_branch_op =
self.trunk_branch(trunk_input_data=p_res_unit_op_1, filters=filters)

        # Perform Mask Branch Operation
        mask_branch_op =
self.mask_branch(mask_input_data=p_res_unit_op_1, filters=filters)

        # Perform Attention Residual Learning: Combine Trunk and
Mask branch results
        ar_learning_op =
self.attention_residual_learning(mask_input=mask_branch_op,
trunk_input=trunk_branch_op)

        # Send branch results through #p residual_units
        p_res_unit_op_2 = ar_learning_op
        for _ in range(self.p):
            p_res_unit_op_2 = self.residual_unit(p_res_unit_op_2,
filters=filters)
```

```
        return p_res_unit_op_2

    def trunk_branch(self, trunk_input_data, filters):
        # sequence of residual units, default=2
        t_res_unit_op = trunk_input_data
        for _ in range(self.t):
            t_res_unit_op = self.residual_unit(t_res_unit_op,
filters=filters)

        return t_res_unit_op

    def mask_branch(self, mask_input_data, filters, m=3):
        # r = num of residual units between adjacent pooling layers,
default=1
        # m = num max pooling / linear interpolations to do

        # Downsampling Step Initialization - Top
        downsampling = MaxPool2D(pool_size=(2, 2),
                                 strides=(2, 2),

padding='same')(mask_input_data)

        for _ in range(m):
            # Perform residual units ops r times between adjacent
pooling layers
            for j in range(self.r):
                downsampling =
self.residual_unit(residual_input_data=downsampling,
filters=filters)

            # Last pooling step before middle step - Bottom
            downsampling = MaxPool2D(pool_size=(2, 2),
                                     strides=(2, 2),

padding='same')(downsampling)


#=================================================================
================================

        # Middle Residuals - Perform 2*r residual units steps before
upsampling
        middleware = downsampling
        for _ in range(2 * self.r):
            middleware =
self.residual_unit(residual_input_data=middleware, filters=filters)


#=================================================================
================================

        # Upsampling Step Initialization - Top
        upsampling = UpSampling2D(size=(2, 2))(middleware)

        for _ in range(m):
            # Perform residual units ops r times between adjacent
pooling layers
            for j in range(self.r):
                upsampling =
self.residual_unit(residual_input_data=upsampling, filters=filters)
```

```
            # Last interpolation step - Bottom
            upsampling = UpSampling2D(size=(2, 2))(upsampling)

        conv_filter = upsampling.shape[-1].value

        conv1 = Conv2D(filters=conv_filter,
                       kernel_size=(1,1),
                       strides=(1,1),
                       padding='same')(upsampling)

        conv2 = Conv2D(filters=conv_filter,
                       kernel_size=(1,1),
                       strides=(1,1),
                       padding='same')(conv1)

        sigmoid = Activation('sigmoid')(conv2)

        return sigmoid

    def attention_residual_learning(self, mask_input, trunk_input):
        # https://stackoverflow.com/a/53361303/9221241
        Mx = Lambda(lambda x: 1 + x)(mask_input) # 1 + mask
        return Multiply()([Mx, trunk_input]) # M(x) * T(x)
```