Norwegian University
of Life Sciences

Master's Thesis 2019 30 ECTS
Faculty of Science and Technology

Detection and Quantification of Rot in Harvested Trees using Convolutional Neural Networks

Tyrone Carlisle Nowell
Master of Science in Data Science

Preface

This thesis is written at the Faculty of Science and Technology at the Norwegian University of Life Sciences (NMBU) in 2019. The thesis consists of 30 ECTS credits and marks the conclusion of a two-year masters degree in Data Science. This thesis has been carried out in collaboration with the Department of Forest Production and Technology at the Norwegian Institute of Bioeconomy Research (NIBIO).

First of all, I would like to thank my supervisors, Associate Professors Kristian Hovde Liland and Oliver Tomic, Faculty of Science and Technology (REALTEK), NMBU for their excellent guidance, support, feedback and Greek. I would also like to thank my fellow Master students Jon Nordby, Jarand Hornseth Pollestad and Vegard Solberg for giving me regular feedback, encouragement and inspiration.

Furthermore, I'd like to thank my colleagues at NIBIO, in particular, Head of Research Rasmus Astrup and Research Professor Bruce Edward Talbot for giving me the means and opportunity to research a topic I have found useful, interesting and exciting.

And to my family and friends, very far and very wide, thank you for your feedback, your encouragement, and your support during my Master's thesis.

Ås, 14th May, 2019

Tyrone Carlisle Nowell

Abstract

Root and Butt-Rot (RBR) is having a significant economic impact on the forest industry and is expected to increase with climate change. The current management strategies are becoming less effective, and little data on RBR distribution is available to develop new ones. In Europe, approximately half of the timber production is using Cut-To-Length timber harvesters which store a considerable amount of data on each tree. Being able to supplement this data with the presence and quantity of RBR in the tree would add significant value to both the forest industry and to the scientific community in developing new strategies for RBR management.

This Master's thesis explored the feasibility of embedding a computer vision system on the harvester for autonomous rot detection and quantification using state of the art Convolutional Neural Networks (CNNs). Among the potential applications of this system, this study assessed the possibilities to (1) provide real time feedback of this information to the harvester operator for faster, more accurate categorisation of the timber quality and (2) enable the collection of big data on RBR distribution for high spatial resolution mapping for the development of new management strategies.

The model developed to detect RBR achieved an F1 score of 97.1% accuracy (precision of 95.2% and recall of 99.0%) which is a significant improvement over previous techniques with an F1 score of 90.8% accuracy (precision of 90.8% and recall of 90.8%). Prediction of the RBR quantity as a percentage of the surface area attained an RMSE of 6.88%, and was reduced to 6.17% when aggregated with the RBR detector.

Evaluating the misclassifications of the detection system indicated that the model performance is at least on par with that of the author. These results indicate that there is significant potential in developing this technology further for both economic and environmental gains.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

As of 2015, the world had 4 billion hectares of forest covering approximately 31% of the global landmass of which 61% are classified as coniferous forests [1]. Coniferous trees, mostly evergreens with needle-shaped or scale-like leaves, are predominant in the Boreal forest - the worlds largest land biome - and make up a significant part of the temperate forests in North America, Europe and Asia.

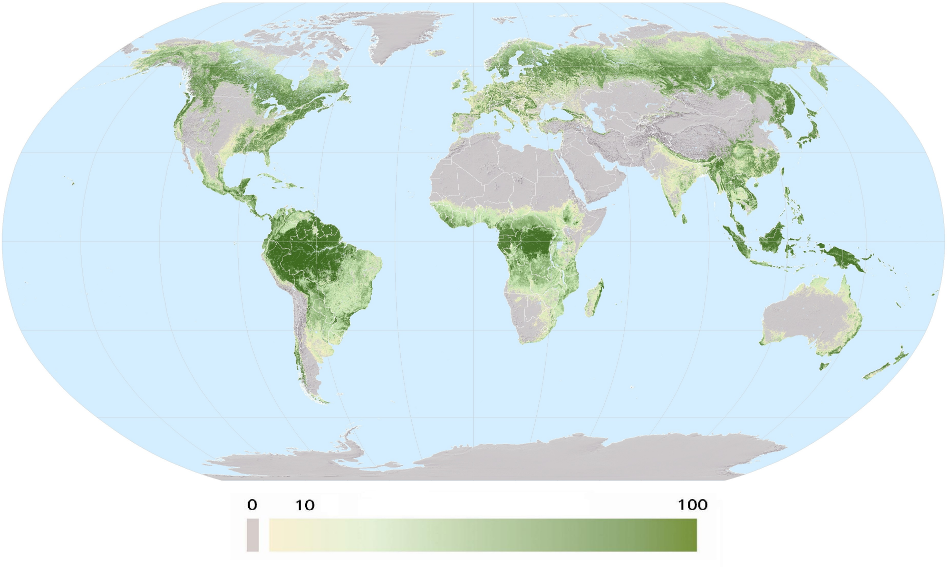


Figure 1.1: *Global percentage tree canopy cover [2]. The northern regions of the world are dominated by coniferous forests.*

Due to their abundance, they make up a large part of the production forests managed by the forestry industry. Approximately 400 million cubic meters of timber are harvested annually for industrial wood supply in Europe [3]. More than half of this is carried out in the Nordic / Baltic countries where Cut-to-length (CTL) harvesting systems are used. These are comprised of a timber harvester for felling and processing the logs and a forwarder for extracting the logs to the roadside.

Wood processing industries (sawmills, pulp mills) have different requirements on raw material quality which drives market demand. The CTL system uses real-time market demand to determine the dimensions of the harvested logs on the site of operation. The operator of the timber harvester has to recognise and assess defects, then make decisions as to the allocation of products to markets on the fly. The log is then classed as one of three general categories depending on the quality: sawlog, pulpwood or biofuel. Harvesting operations in these forests collect a significant amount of data, such as diameters, lengths and volumes of the whole tree and of the sections into which it is cut. This data is stored on onboard computers in a format described by the Standard for Forest machine Data and Communication (StanForD), which has become the global de facto standard for data storage from CTL harvesting operations [4].

Root and Butt-Rot (RBR) is considered a major defect in the production of sawlogs. It is caused by fungi of the *Heterobasidion* genus and is considered one of the most destructive diseases of conifers, particularly in Europe [5]. A comprehensive study on the economic impact of this genus estimated an annual loss of €790 million in Europe alone [6]. The distribution map shown in figure 1.2 highlights the global nature of the problem.

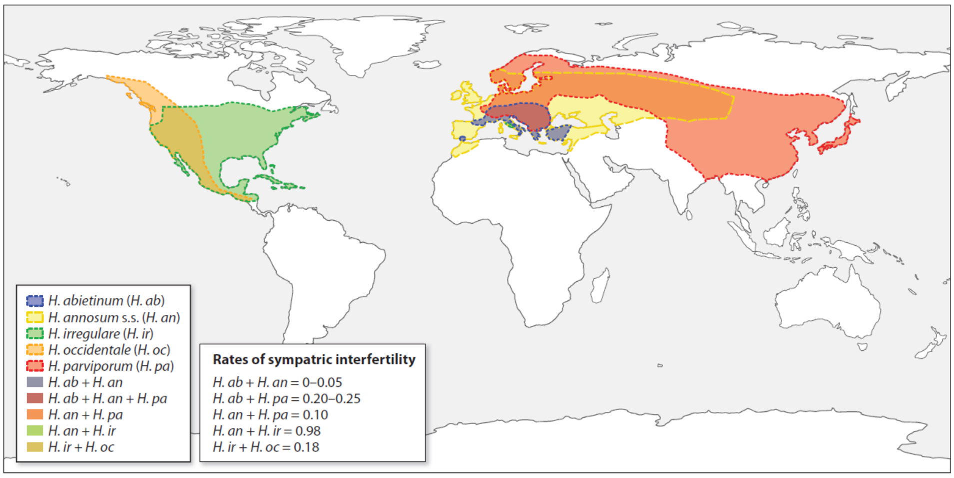


Figure 1.2: *Global distribution of Heterobasidion species [7]*

Harvesting operations are conducted all year round but above 5°C, stumps need to be treated to stop the spread of RBR. This is not always effective and recent mild winters have forced more harvesting operations to be carried out during the unfrozen seasons [8]. Therefore, mitigation strategies currently employed for RBR may become less effective as the sub zero period of the year shortens.

RBR is particularly problematic because infected trees show no outward symptoms. The only non-invasive methods available at present for detection in a living tree use sensors distributed manually around the stem to measure the acoustic or electrical conductivity. This can be used to generate a map of the internal wood density or humidity [9], an indicator of the degradation of the wood. The process is slow and expensive and does not scale commercially. Boring into the tree to obtain samples for analysis is relatively cheap and effective but can render the tree vulnerable to the very diseases being controlled for.

As the name suggests, RBR attacks the tree starting at the roots or through damaged bark, and spreads up the stem in the heartwood, the non-living core of the tree. It can infect healthy uninjured trees by growth through root contacts or grafts and is one of the very few fungal pathogens that is able to infect conifer roots of all ages [5]. It can survive and remain infectious in stumps for over 60 years after felling [10, 11] and can therefore efficiently spread from one forest generation to the next [5]. Norway Spruce (*Picea abies*) and Scots Pine (*Pinus sylvestris*) are the most common conifers harvested in northern Europe. While an infection in pine will be limited to a relatively small area due to more effective immune response mechanisms, the decay column in Spruce can extend 10-12 meters up the stem.

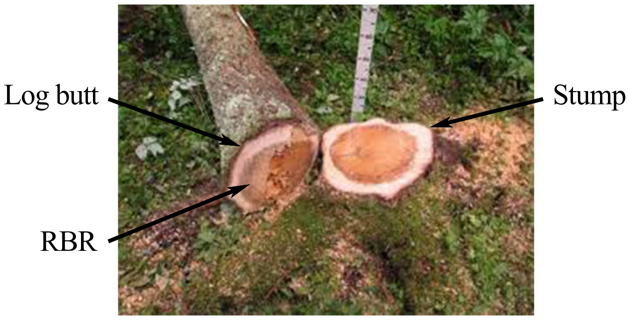


Figure 1.3: *Terminology of a felled tree. Adapted from [12].*

RBR has a significant economic impact due to its degradation of the quality of the timber. Sawlogs are used for aesthetic or structural purposes and thus must be free of RBR. Pulpwood can have RBR present at a maximum of 50% of the surface area, while the rest is designated as biofuel.

Both the forest industry and the research community have been attempting to build models to predict the quality of timber in a given area. These efforts have been met with limited success as the intrinsic properties of the trees are not feasible to measure with current technology before felling, such as disease and insect damage, making data collection expensive.

Subsequently, models predicting the quality of the timber - a large part of which is determined by the presence of RBR - have high variance [13, 14] making it hard to plan management and harvesting operations. The models for the prediction of RBR distribution are made using environmental conditions, such as soil type and hydrology, and the yields from similar sites, but the error is still significant. For example, in Norway, 20% of Norway Spruce show signs of decay due to RBR [15] and error in the site yield in quality timber is frequently above 10%.

Research indicates that the presence of RBR needs to be mapped at much higher spatial resolution to improve the models for quality predictions. New research exploring more accurate localisation of harvested trees has almost succeeded [16] but digitisation of the presence and quantity of RBR on a sufficiently large scale is not possible with current technology.

1.2 Problem Statement

Misclassifying a log can have severe financial repercussions since the value of the categories differs substantially. Pulpwood is worth just 64% of an equal volume of saw logs and biofuel is often left with the forest owner as firewood due to its low market value. Subsequently, the operator is under a lot of pressure to make fast, accurate decisions about the quality of the log and the length to which it should be cut.

To make these decisions, the operator must observe the colour of the saw dust ejected from the cut - an indicator of discolouration - and, if it's dark, manipulate the felled tree into visual range for a manual inspection. This process is time consuming and hard to master. Even a small amount of RBR, which is easy to miss in the saw dust, will be cause for rejection of the log at the saw mill. Any RBR missed is likely to be caught during the forwarding of the timber to the roadside. However, the log is cut to length by the harvester operator which may be wrong for its actual category, reducing the value of the timber.

The roles of pathogens as disturbance agents are expected to increase due to climate change as their ability to adapt to new climatic conditions will be greater than that of their long-lived hosts and the induced stress will make the hosts more vulnerable [17]. The effects of climate change as measured by tree health and tree species composition will be virtually imperceptible over the next few years [18] but models and management strategies will need to adapt to mitigate the effects in the long term.

Forest machines are being fitted with ever more sophisticated sensors to supplement the StanForD data collected [19]. Cameras have been mounted on harvester heads to capture images of the butt end of the logs [20] but only to ascertain the log dimensions for optimisation of the CTL process. This thesis proposes using an automated computer vision system, based on images captured in a similar setup, for RBR detection and quantification using Convolutional Neural Networks (CNNs).

The proposed solution would reduce the mental workload for the operator by providing accurate, real time information about the presence and quantity of RBR. This would enable the operator to make more informed decision about the quality of the log. And, with the development of models of RBR growth through a tree, the extent of the RBR could be predicted so that sawlog recovery could be maximised.

The automated digitisation and integration of this information into the StanForD file associated with the tree would then be added to a database. The RBR distribution could then be used to more precisely model the distribution and modes of infection of RBR to mitigate the economic and environmental impacts.

1.3 Structure of thesis

This thesis starts with the theory behind the machine learning process of CNNs and current state of the art architectures in chapter 2. In chapter 3 the images are explored, the dataset is prepared and chapter 4 describes the methodology applied. Chapter 5 covers the results which are then be discussed in detail in chapter 6. The results of this study are summarised in chapter 7.

Chapter 2

Theory

Machine learning is a subset of artificial intelligence where an algorithm trains a model to fit a system in order to emulate it. This process can be used to get a better understanding of the system's properties or behaviour, but is more commonly used to make predictions about samples within the system. Machine learning consists of three components: the **model**, **optimiser**, and the **dataset**.

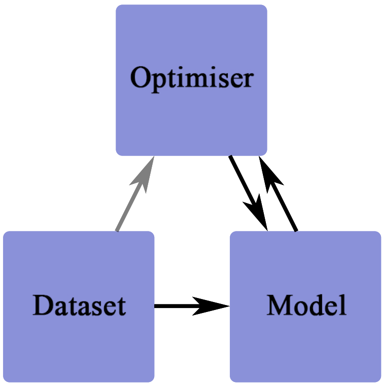
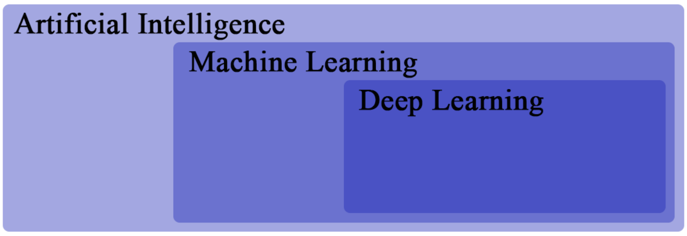


Figure 2.1: *Information flow in the machine learning process. Information flows from the dataset to the optimiser only for learning techniques that require feedback.*

The model defines how data is input, a set of parameters which can be modified to alter the model's behaviour and a function used to generate the output. The optimiser is an algorithm which updates the model parameters throughout the learning process with the aim of improving target performance characteristics. The dataset defines the real world information in a representative form accepted by the model.

The optimisation algorithm determines the type of learning: unsupervised, reinforcement, or supervised. Unsupervised learning techniques do not use feedback of ground truth labels or values, and are used to explore the structure of the data to extract meaningful information [21]. Reinforcement and supervised learning techniques use feedback to guide the learning process and can both be used to build predictive models. Reinforcement learning uses **evaluative** feedback from a reward system [22], which explores all possible inputs, actions or outputs. This is a relatively exhaustive process and convergence to a solution is generally very slow. Supervised learning uses **instructive** feedback which exploits the ground truth labels or values to converge on a solution much faster and will be the focus of this study.

Deep learning is a form of machine learning that requires learning parameters of more than one consecutive step. That is, the input signal is transformed through two or more consecutive trainable models to generate an output. Deep learning was integrated into the field of image analysis with the introduction of the convolutional neural network, a two stage network, comprising many steps, that extracts features and then finds the relationships between them.



This chapter will cover the theory pertaining to this study in the context of a convolutional neural network trained on an image dataset using supervised learning techniques. Convolutional neural networks will be described in section 2.1 and the optimisation process by supervised learning for these types of models will be described in section 2.2. The structuring and processing of datasets will be covered in section 2.3.

2.1 Model

2.1.1 Artificial neurons

The study of artificial neural networks started in 1943 with the development of a mathematical model of a biological neuron typically referred to as the *McCulloch-Pitts neuron* [23]. This was the first model to make assumptions about the activity of a neuron which made it possible to simplify and encode. These assumptions are as follows:

1. It has a binary output.
2. The sum of the inputs must surpass a threshold to trigger the neuron.
3. There is no delay in the system.
4. No input, no output.
5. The structure of the net does not change over time.

This neuron model can be summarised as a logic gate with a binary output where the input signals $\boldsymbol{x}$ are integrated and an output signal o is generated according to whether a threshold θ is reached as illustrated in figure 2.2.

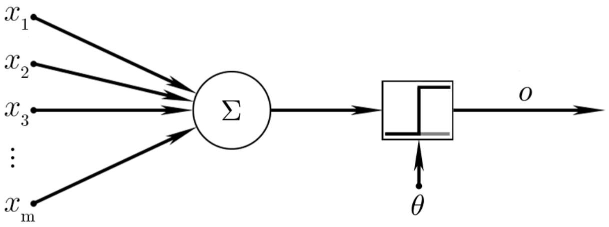


Figure 2.2: *McCulloch-Pitts model*

Perceptron

The Perceptron [24] model adopted and encoded the McCulloch-Pitts neuron into what can be considered the elemental building block of a modern artificial neural network. This model introduced the concept of applying weights to the input values in order to scale them according to their importance to the model. The net input of a function z is expressed as the linear combination of the input features $\boldsymbol{x}$ and the corresponding weights:

$$z = w_1 x_1 + \cdots + w_m x_m \tag{2.1}$$

where:

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \tag{2.2}$$

The Perceptron was developed in the context of a binary classification task where the two classes are defined as 1 (positive) and -1 (negative). The **activation function** of a Perceptron model $\phi(z)$ takes z as input and if z is greater than a defined threshold θ a class of 1 is predicted and -1 otherwise. This is implemented in the Perceptron model using a variant of the unit step function:

$$\phi(z) = \begin{cases} 1 & z \geq \theta \\ \text{-1} & \textit{otherwise} \end{cases} \tag{2.3}$$

By adding a new constant term $w_0 x_0$ to z, the definition can be simplified to:

$$z = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x} \tag{2.4}$$

Where $w_0 = -\theta$ and $x_0 = 1$ and therefore:

$$\phi(z) = \begin{cases} 1 & z \geq 0 \\ \text{-1} & \textit{otherwise} \end{cases} \tag{2.5}$$

w_0 is normally referred to as the bias unit. A functional diagram of this model is shown in figure 2.3 on the facing page.

Adaline

Linear activation functions were introduced with the conceptualisation of the **Adaptive Linear Neuron (Adaline)** model [25]. The Adaline model uses a linear activation function instead of the unit step function which is simply the identity function of the net input, such that:

$$\phi(\boldsymbol{w}^T \boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} \tag{2.6}$$

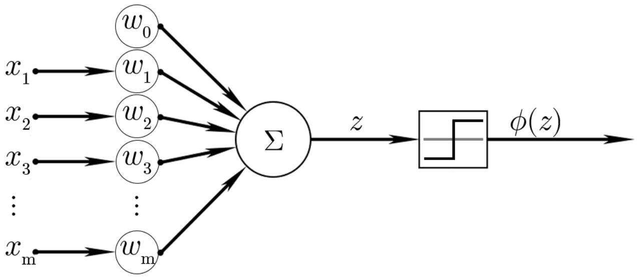


Figure 2.3: *Perceptron model*

In the Adaline model, $\phi(z)$ is used to update the weight vector, the advantages of which will be discussed in section 2.2.1 on page 26, but the activated signal can still be thresholded for binary classification.

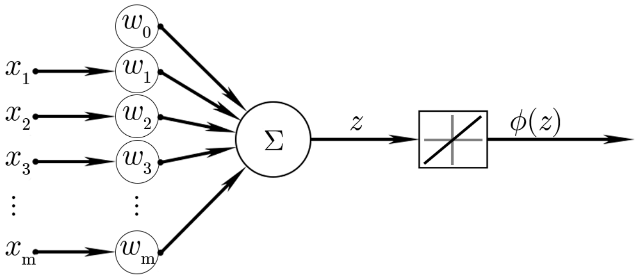


Figure 2.4: *Adaline model*

More broadly, an artificial neuron can be represented by the general model illustrated in figure 2.5 on the following page for any activation function ϕ.

Non-linear activations

A model using a linear activation function will always output a linear combination of the inputs and thus, will only be capable of modelling linearly separable data in a classification task or linearly distributed data in a regression task. For more complex data, the model must be able to compensate for non-linearities, which is accomplished by using a non-linear activation function. Virtually any monotonic function can be used as an activation function, however the properties required for gradient descent optimisation, to be discussed further in section 2.2.1 on page 26, and computational efficiency, are normally limiting factors.

One of the first non-linear activation functions was the **Sigmoid** function, equation 2.7 on the following page, a special case of a logistic function that takes a real

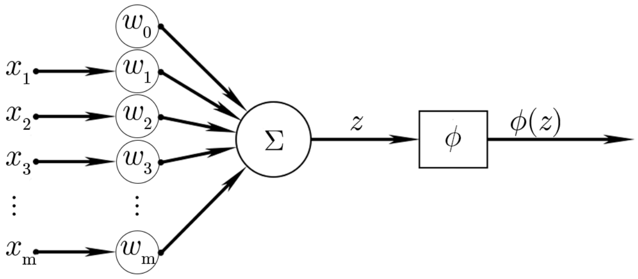


Figure 2.5: *Generalised neuron model*

value as input and returns a value in the range $(0, 1)$.

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{2.7}$$

The **Hyperbolic Tangent (tanh)** function, equation 2.8, is a similar *S*-shaped function which returns a value in the range $(-1, 1)$ giving a zero-centred output. Both functions are illustrated in figure 2.6.

$$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.8}$$

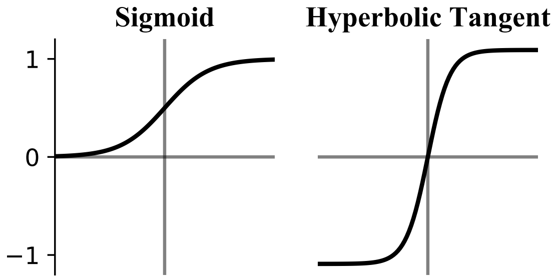


Figure 2.6: *Sigmoid and Hyperbolic Tangent activation functions*

These activations are not commonly used in neural networks because of the computational load of solving these functions and the vanishing gradient problem, which will be discussed further in section 2.2.3 on page 35. These issues lead to the introduction and popularisation of *rectified linear* functions. In the context of neural networks, *rectified linear* means that the positive part of the activation function is linear. The most commonly used rectified linear functions are shown in figure 2.7 on the facing page.

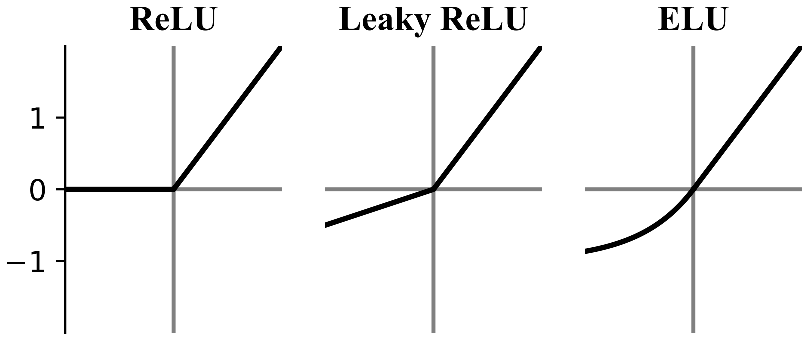


Figure 2.7: Rectified linear activation functions

Rectified Linear Unit (ReLU) [26] can be expressed as a piece-wise linear function simply as $max(0, z)$. This simple function is very efficient to compute but can also suffer from the vanishing gradient problem since it has a gradient of zero when $z < 0$.

Leaky ReLU [27] is a variant of ReLU with a small slope below zero expressed as $max(az, z)$ for $0 < a < 1$. This adjustment avoids the vanishing gradient problem and is still relatively simple to compute but the discontinuity at zero means that the gradient at this point cannot be derived and must be predefined (normally as zero).

The **Exponential Linear Unit (ELU)** [28] is a rectified activation function that smooths slowly below zero converging on a constant value $-\alpha$. This is a smooth, continuously differentiable function, but the extra complexity of the function adds to the computational workload. The benefits of using these function on a larger scale are discussed further in section 2.2.3 on page 36.

Regularisation

A common problem in machine learning with non-linear functions is overfitting the training data. When a model overfits the training data it results in high variance in predictions for similar samples and poor performance on previously unseen data. One method to avoid overfitting a dataset is through regularisation of the model parameters. Regularisation penalises large parameter values in a model by adding a term to the cost function.

The most common form of regularisation in machine learning models is L2 regularisation, which can be summarised as follows:

$$\lambda||\boldsymbol{w}||^2 = \lambda \sum_{j=1}^{m} w_j^2 \tag{2.9}$$

where λ is the regularisation parameter. By increase the cost of using large weights, the weights shrink decreasing the complexity of the model. L1 regularisation is more severe, as it tends to shrink the weights of less important features to zero:

$$\lambda||\boldsymbol{w}|| = \lambda \sum_{j=1}^{m} |w_j| \tag{2.10}$$

There are other methods to reduce overfitting that have the same effect but generally apply to a network of neurons, not to single neurons.

2.1.2 Artificial neural networks

In the context of an Artificial Neural Network (ANN), an individual neuron is referred to as a **unit** and a single layer of a neural network can be regarded as a group of parallel neurons that all receive the same input, but are differentiated by their weight vectors. All the units in a layer use the same activation function. A multilayer neural network structure is illustrated in figure 2.8.

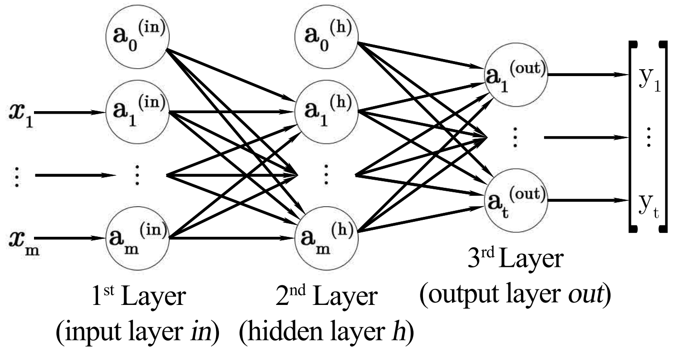


Figure 2.8: *Multilayer feedforward neural network. An input layer* ***in****, a hidden layer* ***h*** *and an output layer* ***out*** *where* ***m*** *is the number of input values and* ***t*** *is the number of classes in a multi-class classification task. The vector $[y_1, \ldots, y_t]$ contains the predictions of the network* $\hat{\boldsymbol{y}}$.

Hidden layers

A **hidden** layer is a layer in a multilayer neural network that is neither the first layer of the network (the **input** layer) nor the last layer of the network (**output** layer). A network that has more than one hidden layer is called a Deep Artificial Neural Network (DNN). The number of layers in a network and the number of units in a layer are referred to as its **depth** and **width** respectively.

A hidden layer is comprised of an arbitrary number of units and one bias unit. A **dense** layer is one where every unit is connected to all units in the previous layer. Defining a dense layer is normally limited to two parameters: the number of units and the activation function used. The bias unit acts as the bias for the whole of the next layer.

Input and output layers

The first and last layers define the format of the input and output signals. The input layer has one unit for every value in the input signal and one bias unit.

The activation function of a machine learning model determines whether it performs a classification or regression task and the same can be said for a neural network. However, only the activation function of the last layer of an ANN defines the format of the output and therefore the task the model is suited for.

The number of units in the last layer determines the number of outputs from the network. In a classification task, the number of units is the number of classes except for binary classification where the number of units is exactly one. Typically, a binary classification task will use a sigmoid activation function to output class probabilities, while a multiclass task will use a **softmax** activation which normalises the outputs of all units so that the sum of all outputs equals one. These are the probability of the sample belonging to each class, the highest of which will determine the models class prediction.

Regression tasks require one neuron per continuous variable and typically use linear activation functions.

Activation by forward propagation

Forward propagation is a series of tensor multiplications with activations in between. The inputs to each unit in a layer are multiplied by their respective weights and summed together. The results are then activated and fed into the next layer where the process is repeated.

Trainable parameters

The trainable parameters of a dense layer consist of the weights of each neuron. The number of parameters can be expressed as:

$$n_p = (n_i + 1) \times u_l \tag{2.11}$$

where n_i is the number of input values and u_l is the number of units in the dense layer in question.

Network Capacity

The **capacity** of a network is its ability to emulate complex models. The capacity of a network to overfit a dataset increases with increasing number of trainable parameters since they define the complexity of the model. A dataset rarely covers the full distribution of all potential samples so an overfitted model could have unpredictable results when applied to new unseen data. How to evaluate the network capacity and whether it is overfitted will be discussed further in section 2.3.1 on page 39.

2.1.3 Convolutional neural networks

Since their introduction in the 1990's where they showed outstanding performance in image classification tasks [29], Convolutional Neural Networks (CNNs) have been a popular field of research, which has lead to substantial improvements in machine learning and computer vision applications. Unlike traditional machine learning models which require domain expertise or computational feature extraction techniques, CNNs are able to learn which features are salient and extract them from raw data. Another advantage is that the extracted features are translationally invariant. This means that the location of the feature in the image does not affect the output, making the network more robust when dealing with unstructured data.

Deep CNNs, with stacked convolutional layers, were first popularised by AlexNet [30] before which convolutional layers had only been used as the input layer. They have been found to emulate the primate visual system [31] with its sequence of processing stages: edge detection, primitive shapes, gradually moving up to more complex visual shapes [32]. Low-level features are extracted by the first convolutional layer, after which, every subsequent convolutional layer combines them into progressively higher-level features constructing a feature hierarchy.

These features are represented in the **feature maps** of each layer where each element in the map corresponds to a local patch of values from the original input, or **signal**. The dimensions of the local patch are defined by the kernel, or **filter**. Therefore, feature maps are essentially activation maps output from a convolution operation of a filter on an input signal. This concept is illustrated in figure 2.9.

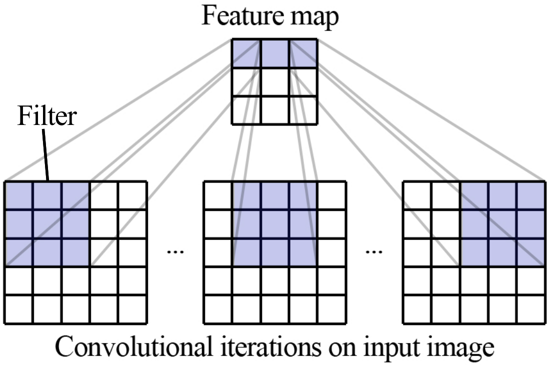


Figure 2.9: *Discrete convolutions of a 2D image. The filter is convolved iteratively with each pixel patch in the input image.*

Discrete Convolutions

The basis of a CNN is a discrete convolution. For one dimensional data, a discrete convolution can be defined as follows:

$$\boldsymbol{y} = \boldsymbol{x} * \boldsymbol{w} \tag{2.12}$$

Where $\boldsymbol{x}$ is the input, $\boldsymbol{w}$ is the filter and $*$ denotes the convolutional operation. For every input unit i this can be simplified to:

$$\boldsymbol{y}[i] = \sum_{k=0}^{k=m-1} \boldsymbol{x}^p[i+m-k]\boldsymbol{w}[k] \tag{2.13}$$

Where m is the filter length and p denotes the **padding** mode which is discussed and defined below. The padded vector $\boldsymbol{x}^p$ has size $n+2p$ when n is the number of input elements. The operation can be simplified by flipping the filter to get a dot product notation which can be repeated like a sliding window:

$$y[i] = x[i : i+m] \cdot w^r \tag{2.14}$$

where w^r is the rotated filter. Equation 2.13 on the previous page can easily be extrapolated to two dimensions, defined as:

$$\boldsymbol{Y}[i,j] = \sum_{k_1=0}^{k_1=m-1} \sum_{k_2=0}^{k_2=m-1} \boldsymbol{X}^P[i+m-k_1, j+m-k_2]\boldsymbol{W}[k_1,k_2] \tag{2.15}$$

which is essentially nesting equation 2.13 on the preceding page within itself to increase the dimensionality.

Padding

Computing a discrete convolutions of an input of a fixed size results in uneven representation of the input values. The outer most values will only be evaluated once, while the inner most values will be evaluated m times where m is the dimension of a 1D filter, or $m \times n$ times where $m \times n$ are the dimensions of a 2D filter. By padding the borders of the input with zeros, the filter can be shifted further out, increasing the number of convolutions of the original input values.

There are three modes of padding: **full**, **same** and **valid** shown in figure 2.10 on the next page.

- For full padding $p = m - 1$. This means that convolutions will be evaluated when any section of the filter overlaps with the image. Therefore, the output feature map will be larger than the input image which means it is rarely used in CNNs.
- Same padding results in a feature map the same size as the input image by calculating p using the filter size, $p = \lfloor \frac{m}{2} \rfloor$.
- Valid padding is when $p = 0$ so the feature map is smaller that the input image. For deep CNNs, valid padding is rarely used because the feature space reduces too quickly limiting the information flow through the network, which can have a negative impact on the network performance.

Full padding is the only mode that gives equal weight to all pixels in the input image. Same and valid padding give less weight to the outer most pixels since they are part of fewer pixel patches that are evaluated.

Note that the dimensions of the output are also largely affected by **stride**, which is the number of pixels the filter is moved between each iteration. A stride of one is most commonly used as larger strides can have the same impact on dimensions as valid padding. The size of the feature map is the number of iterations of the filter

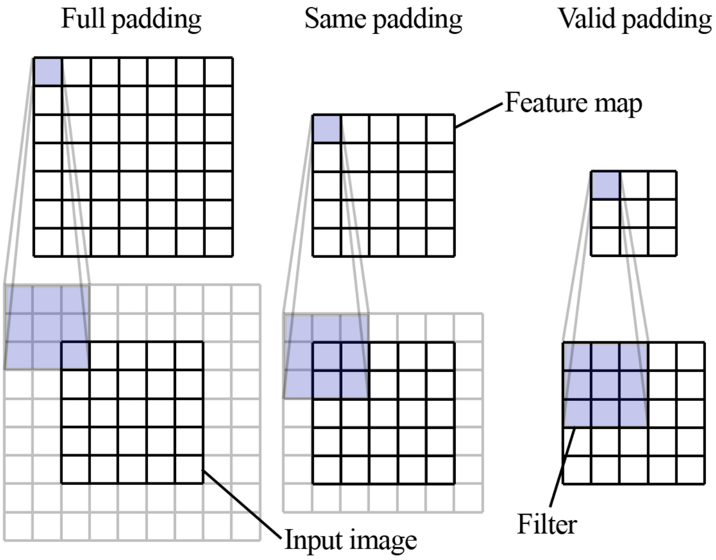


Figure 2.10: Padding of a 2D image.

in a given dimension. The size o can be calculated as follows:

$$o = \left\lfloor \frac{n + 2p - m}{s} \right\rfloor + 1 \tag{2.16}$$

Where n is the number of input elements, p denotes the padding mode, m is the filter size and s is the stride length. This equation can be applied to each dimension to get the feature map size.

Convolutional layer

A convolutional layer is generally defined by four properties: the number of filters, the size of the filters, the padding mode and the stride length. A convolutional layer does not activate the output and so is normally paired with an activation layer, which is often defined as the fifth property of the layer. While a stride length of one and same padding is standard for the previously mention reasons, the number of filters and filter size should be tailored according to the number and size of the features expected in the input signal. Optimising these layer parameters is incredibly complex so an iterative trial and error approach is normally used.

A standard convolutional layer outputs one feature map per filter even if the input signal has multiple channels, like an RGB image (3 channels) or the feature maps from the previous convolutional layer. This is because each convolutional filter consists of one kernel per channel and the result of the individual convolutions are summed together, collapsing the channels to one as shown in figure 2.11.

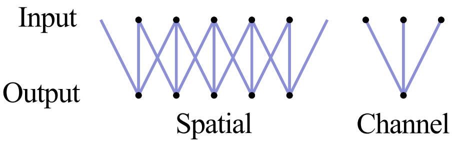


Figure 2.11: *Standard convolution of a single filter of width* 3 *on an input of width* 5 *with* 3 *channels and same padding.*

The computational complexity of this filter operation can be expressed as:

$$O_{filter} = w_i \times w_f \times c_i \tag{2.17}$$

Where w_i is the input width, w_f is the width of the filter and c_i is the number of input channels. This can be extrapolated to 2D images:

$$O_{filter} = (w_i h_i) \times (w_f h_f) \times c_i \tag{2.18}$$

Where h_i is the input height and h_f is the height of the filter. The complexity of multilayer convolutional operations can increase drastically as shown in figure 2.12. In standard CNNs, the number of channels increases after the first layer

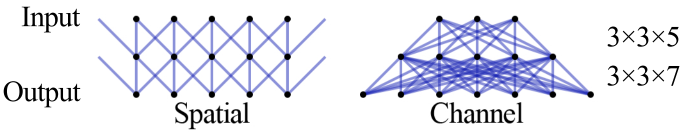


Figure 2.12: *Complexity of multichannel multifilter convolutions. The first layer consists of five* 3×3 *filters and the second consists of seven* 3×3 *filters.*

since each channel represents a feature. An advantage of CNNs is the vast number of salient features that can be extracted, however this can lead to an rapid increase in the computational load since the complexity of each layer depends on the number of channel output by the previous one as shown in equation 2.19.

$$O_{layer} = O_{filter} \times n_f = (w_i h_i) \times (w_f h_f) \times c_i n_f \tag{2.19}$$

Where n_f is the number of filters.

Trainable parameters

Other than the ability to learn salient features from raw data, CNNs are also much more computationally efficient due to the much smaller number of trainable parameters. A fully connected layer of n_u units with an input of p pixels in n_c channels would have $n_u \times p \times n_c + 1$ weights to train. However, the number of trainable parameters n_p in a convolutional layer is determined by the filter size f, the number of filters n_f and the number of input channels n_c as defined in equation 2.20.

$$n_p = n_f(n_c \times f + 1) \tag{2.20}$$

Hence, the number of trainable parameters in a convolutional layer is not related to the number of pixels in the input signal, drastically reducing the number of parameters. Both forward and backward propagation operations are computed on a pixel level so the computational workload is not reduced as drastically as the number of parameters.

Subsampling

Instead of increasing the filter size of each consecutive convolutional layer to construct a feature hierarchy, most neural networks will use **pooling** operations to subsample the feature maps and keep the filter size constant.

This operation is normally applied as either **max-pooling** or **mean-pooling**. These operations use a sliding window to look at a pixel patch of size $P_{n_1 \times n_2}$ and return either the max value or mean value respectively. The window is normally non-overlapping so that every pixel is only evaluated once. Therefore, a window of size 2×2 will reduce the input signal by half in both dimensions, i.e. a 2×2 window applied to an input of size 16×16 will have an output of size 8×8.

Max-pooling is often used in CNNs because it reduces the effect of small local changes making the features more robust to noise. Pooling layers also decrease the size of features increasing computational efficiency and since this type of layer has no trainable parameters, it does not add to the computational workload.

Batch normalisation

To increase the stability of a neural network, a batch normalisation layer normalises, scales and shifts the signal from the previous layer. The mechanism and benefits of this are covered in section 2.2.1 on page 30. This is particularly useful for very deep networks as it limits the signal variance throughout the network making it more robust.

Kernel stacking

Large filters are more expensive to compute, but are able to extract larger features. **Kernel stacking** is a method of finding larger features using computationally cheaper smaller filters. This is done by stacking two convolutional filters sequentially. This can either be done using two smaller filters of the same shape or by addressing one dimension at a time as shown in figure 2.13.

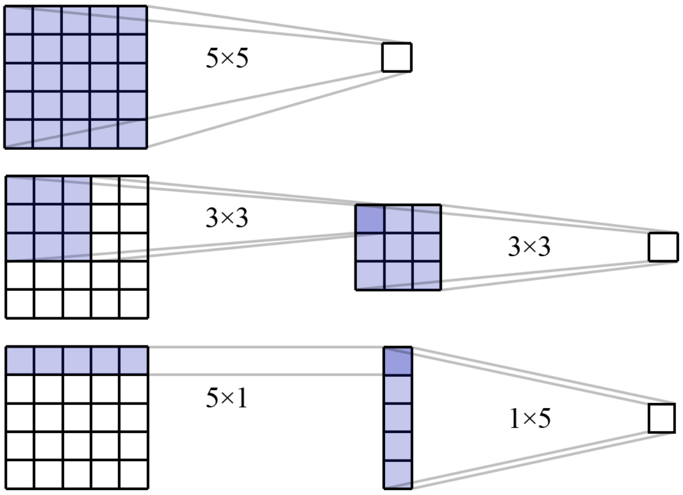


__Figure 2.13:__ An example of different kernel stacking methods.

Kernel stacking yields the most accurate representation when a linear activation function is used between the kernel, but achieves the best performance with ReLU. The advantage of kernel stacking is that the number of parameters is reduced significantly as expressed in equation 2.21.

$$\begin{aligned} 5\times 5 &\xrightarrow{5\times5} 1\times 1 & n_p &= 5\times 5 = 25 \\ 5\times 5 \xrightarrow{3\times3} 3\times 3 &\xrightarrow{3\times3} 1\times 1 & n_p &= 3\times 3 + 3\times 3 = 18 \\ 5\times 5 \xrightarrow{5\times1} 1\times 5 &\xrightarrow{1\times5} 1\times 1 & n_p &= 5\times 1 + 1\times 5 = 10 \end{aligned} \tag{2.21}$$

where n_p is the number of trainable parameters for the given filter combination.

Depthwise separable convolution

A popular way to address this issue is by using depthwise separable convolutions. This method convolves the channels separately followed by a pointwise convolution. This method is illustrated in figure 2.14 on the next page.

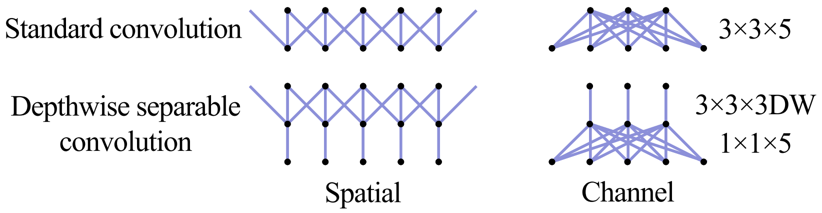


Figure 2.14: *Depthwise separable convolution. A* $3 \times 3 \times 5$ *layer can be replaced by a* $3 \times 3 \times 3DW$ *followed by a* $1 \times 1 \times 5$*. The DW notation means that the filter convolutions are not performed across channels.*

Using equation 2.19 on page 20, this can be shown to be more computationally efficient considering the depthwise separable convolution:

$$O_{3\times3\times5} = (5 \times 5) \times (3 \times 3) \times 3 \times 5 = \mathbf{3375} \text{ computations}$$

$$\begin{aligned} O_{3\times3\times3DW} &= (5 \times 5) \times (3 \times 3) \times 3 \times 1 &&= 675 \\ O_{1\times1\times5} &= (5 \times 5) \times (1 \times 1) \times 3 \times 5 &&= 375 \\ O_{DS} &= O_{3\times3\times3DW} + O_{1\times1\times5} &&= \mathbf{1050} \text{ computations} \end{aligned}$$

Global pooling

The interface between the feature maps of the final convolutional layer and first dense layer has a large effect on the total number of trainable parameters in a model. Connected directly to the feature maps, a unit in the dense layer would require one trainable weight per pixel in every feature map. A number of different methods are used to mitigate this effect, the most popular of which is the global pooling layer. Just like a regular pooling layer, a global pooling layer returns the **mean** or **max** but of the whole feature map. This reduces every feature map to a single value so that a unit in the dense layer will only have as many weights to train as there are feature maps.

2.1.4 Non-sequential layers

With the development of applied CNNs, the focus has shifted from a sequential topology to a more modular approached based on non-sequential network architectures. These modules, or **blocks**, typically consist of convolutional layers, pooling layers and batch normalisation layers. Most of these modules attempt to address the issue of information loss in deep networks. Information is lost through each convolutional layer as the signal that is not picked up by one of the filters is discarded. Therefore, the objective of non-sequential blocks has been to maintain or enhance the complexity of the neural network while reducing the distance the signal has to travel between the input and the output layers.

Inception block

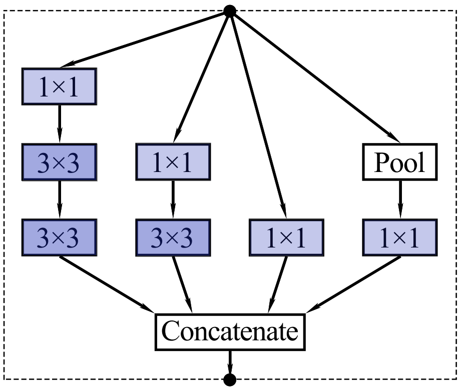


Figure 2.15: *An example of an Inception block.*

Convolutional layers use linear transformations with non-linear activation functions to achieve the flexibility required of feature extraction tasks. The Inception [33] block was found to produce non-linear transformations by using parallel convolutional layers and concatenating the output. The newest versions of the module incorporate pointwise convolutions for dimension reduction before computing the more expensive 3×3 or 5×5 convolutions. An example is shown in figure 2.15.

Residual block

A neural network is a universal function approximator and, generally, adding layers increases the accuracy of the approximation. However, this has diminishing returns due to effects such as the vanishing gradient problem and the curse of dimensionality. Information is lost through each convolutional layer because only that which is picked up by the filters is passed on. The rest, the **residual**, is discarded. A Residual block is a simple concept: sum the input with the output of a layer so that the residuals are passed on to the next layer as shown in figure 2.16.

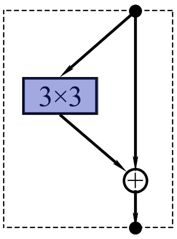


Figure 2.16: *An example of a Residual block.*

Residual inception block

The Residual Inception block is a combination of the two previous concepts. The block uses parallel filters of different sizes to extract features of different sizes and then sums the results with the input.

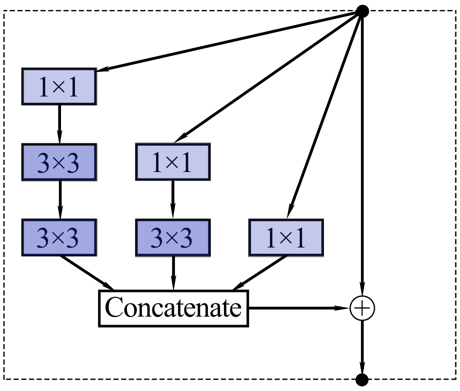


Figure 2.17: An example of a Inception Residual block.

This retains more information in the signal while adding features of multiple scales.

Densely connected convolutions

To address the issue of information loss in deep CNNs, the Densely Connected Convolutional block connects the input signal and the output of every convolutional layer to the input of every successive convolutional layer. As opposed to summing the signal like the residual blocks, the feature maps are concatenated, which leads to less correlation in the features of subsequent layers. It also means that the densely connected neural network has access to the entire feature hierarchy instead of just the higher level features.

2.1.5 Modern architectures

Fine-tuning pretrained models has become common practice in applied computer vision solutions using CNNs. A CNN can be considered as two separate sections: the feature extractor and the densely connected neural network. The feature extractor is comprised of the convolutional layers, which convert an image into the feature space. The densely connected neural network then uses the relationships between the features to make a prediction.

A feature extractor fitted to a large enough dataset with varied enough feature requirements will generally extract a large variety of features that can be used for

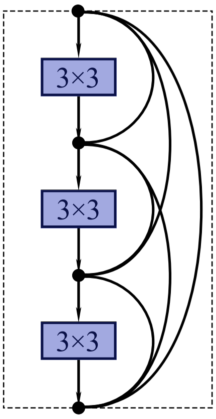


Figure 2.18: *An example of a densely connected convolutional block.*

any task. To re-purpose these networks, a new densely connected neural network can be connected to the existing feature extractor and tailored to the new task, a process which will be discussed further in section 2.2.4 on page 37.

2.2 Optimiser

2.2.1 Gradient descent optimisation

The first learning rule defined for the application of machine learning was the Perceptron learning rule. The learning process utilises the error to update the weight vector **w** and can be defined by the following steps:

1. Initialise the weight vector **w** to zeros or small random numbers.
2. For a given training sample $x^{(i)}$:
    (a) Compute the output $\hat{y}$.

    (b) Update the weight vector ***w***.

Where $\hat{y}$ is the output of the unit step activation function defined in equation 2.5 on page 10 and the simultaneous update of each weight w_j in the weight vector ***w*** can be expressed as:

$$w_j := w_j + \Delta w_j \tag{2.22}$$

Δw_j is derived from the Perceptron learning rule:

$$\Delta w_j = \eta\Big(y^{(i)} - \hat{y}^{(i)}\Big)x_j^{(i)} \tag{2.23}$$

Where η is the learning rate, $y^{(i)}$ is the true class label, $\hat{y}^{(i)}$ is the predicted class label and $x_j^{(i)}$ is the input value for the corresponding weight w_j. In the event of a misclassification, the weights are pushed in the direction of the true class:

$$\Delta w_j = \eta(1 - -1)x_j^{(i)} = 2\eta x_j^{(i)}$$
$$\Delta w_j = \eta(-1 - 1)x_j^{(i)} = -2\eta x_j^{(i)}$$

The Adaline algorithm is of particular importance because it was the first to illustrate the key concepts of defining and minimising continuous cost functions. The objective of the learning process must be formulated in order for the algorithm to optimise its weight vector. This is referred to as the objective function, which often takes the form of a **cost** (or **loss**) function to be minimised. In the case of Adaline, the cost function *J* is defined as the Sum of Squared Error (SSE) between the computed output $\hat{y}$ and the true class label y:

$$J(\mathbf{w}) = \frac{1}{2}\sum_i \Big(y^{(i)} - \phi\big(z^{(i)}\big)\Big)^2 \tag{2.24}$$

The constant term $\frac{1}{2}$ is added for simplification of the derivative for the following reasons. The key advantage of having a differentiable convex cost function is that the **gradient descent** optimisation algorithm can be used to find the optimal weights to minimise the cost function. The general principle behind gradient descent, illustrated in figure 2.19 on the next page, is that for each learning iteration a step is taken *down* the cost function slope proportional to the magnitude of the slope in an attempt to find the cost minimum.

The weight vector is updated by taking a step in the opposite direction of the gradient $\nabla J(\boldsymbol{w})$ of the cost function $J(\boldsymbol{w})$ where the weight vector change $\Delta\boldsymbol{w}$ is defined as:

$$\Delta\boldsymbol{w} = -\eta\nabla J(\boldsymbol{w}) \tag{2.25}$$

The gradient of the cost function is the partial derivative of the cost function with respect to each weight w_j. Therefore, the update applied to each weight w_j can be written as:

$$\Delta w_j = \eta\sum_i \Big(y^{(i)} - \phi\big(z^{(i)}\big)\Big)x_j^{(i)} \tag{2.26}$$

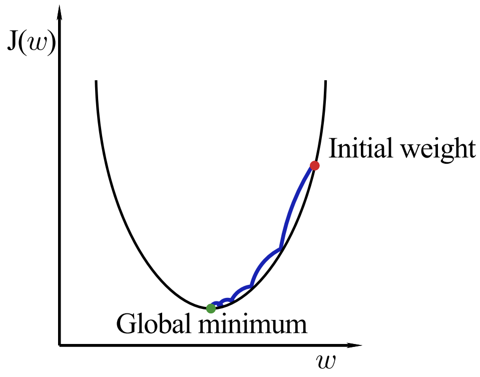


Figure 2.19: *Gradient Descent optimisation. The step size is proportional to the gradient of the cost function.*

This weight vector update is calculated based on all samples in the batch being used for training, as opposed to on a per sample basis (**online** learning), which is referred to as **batch gradient descent** or **full batch** learning. Datasets that are too large to fit in the computer memory can be processed using a technique called **mini-batch** learning that iterates through smaller subsets, updating the weights at each iteration.

However, equation 2.26 on the preceding page requires that $y^{(i)}$ be a continuous value which would be the case in a regression model but not in a classification model where the labels are binary values. For example, predicting if rot is present or not can only have a binary output, true or false. For classification tasks the error can be transformed to a continuous value using a **binary cross-entropy** function:

$$D\big(\hat{y}, y\big) = -\big(ylog\big(\hat{y}\big) + (1-y)log(1-\hat{y})\big) \tag{2.27}$$

Or a **categorical cross-entropy** function for a multiclass problem:

$$D\big(\hat{y}, y\big) = -\sum_{i=1}^{k} y^{(i)} log\big(\hat{y}^{(i)}\big), \tag{2.28}$$

where k is the number of classes.

While regression models output a continuous variable which can be used directly as the loss, using a more sophisticated function can lead to faster convergence. The most commonly used loss functions are **mean squared error (MSE)** and **mean absolute error (MAE)**.

$$MSE = \frac{1}{n}\sum_{i=0}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 \tag{2.29}$$

$$MAE = \frac{1}{n}\sum_{i=0}^{n}|y^{(i)} - \hat{y}^{(i)}| \tag{2.30}$$

where n is the number of samples. For regression models, these function can also be used to evaluate the performance characteristics which will be discussed further in section 2.2.5 on page 37.

Learning rate

The learning rate η determines the size of the weight updates in each training cycle. In gradient descent optimisation the learning rate is multiplied by the gradient of the cost function as shown in equation 2.25 on page 27.

One of the key challenges of gradient descent learning is finding the correct learning rate. As depicted in figure 2.20, the learning rate will determine the time for the model to converge and whether it will ever find the global minimum.

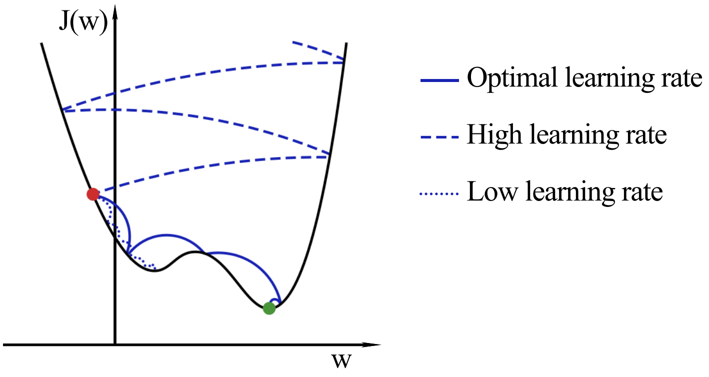


Figure 2.20: *Effect of learning rate on gradient-based learning*

Feature scaling

Since weights are initialised at or very close to zero, a cost function with a global minimum far from zero will take a long time to converge to. Increasing the learning rate would mean faster convergence but it would be at the expense of finer tuning. One method to mitigate this effect is **feature scaling**. The most common method of feature scaling is **standardisation** which shifts the mean of a feature to zero and scales the feature to get a standard deviation of one as shown in equation 2.31.

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x} \tag{2.31}$$

Where $x^{(i)}$ is a particular sample, μ_x is the mean and σ_x is the standard deviation of the feature. The effect of standardisation is depicted in figure 2.21.

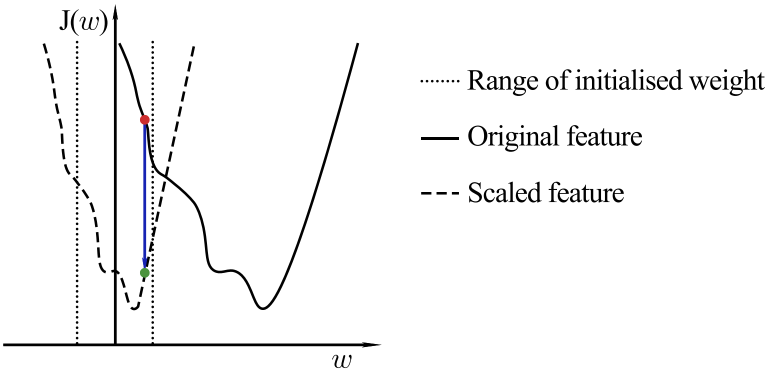


Figure 2.21: *Effect of feature scaling on gradient-based learning.*

This can also increase the rate of convergence since the gradient is more likely to be in the direction of the global minimum as illustrated in figure 2.22 on the facing page.

Activation functions that are not zero centred have a negative effect on multilayer neural networks [34] which is one reason why sigmoid functions are not used. This is partially due to the fact that during the training process the output of a layer will change and thus the input to the next layer. This causes a shift in the signal distribution that will then have to be learned by the successive layer. This stochasticity in the input signal makes it difficult for the layer to converge to good weights. This problem is known as internal covariate shift.

A **batch normalisation** layer mitigates this problem through a form of learned feature scaling, or signal normalisation. This is done by calculating the batch mean and variance:

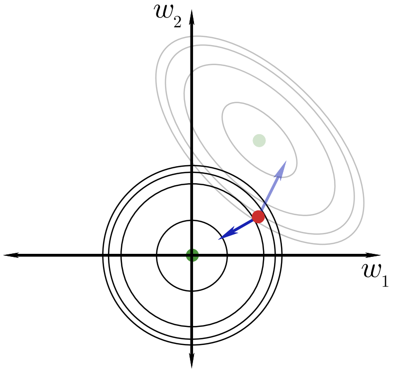


Figure 2.22: *Effect of feature scaling on gradient direction. w_1 and w_2 are two weights of the neuron.*

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

where μ_B is the batch mean and σ_B^2 is the batch variance. The input is then normalised using the calculated batch mean and variance:

$$\bar{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

where x_i is the input signal, ϵ is a small float added to the variance to avoid dividing by zero and $\bar{x}_i$ is the normalised input. The learn parameters γ and β are then used to scale and shift the signal:

$$y_i = \gamma \bar{x}_i + \beta$$

where y_i is the output signal.

2.2.2 Back propagation of error

In 1960, the basics of a continuous back propagation model emerged from control theory [35] but only became the basis for modern neural networks in 1988

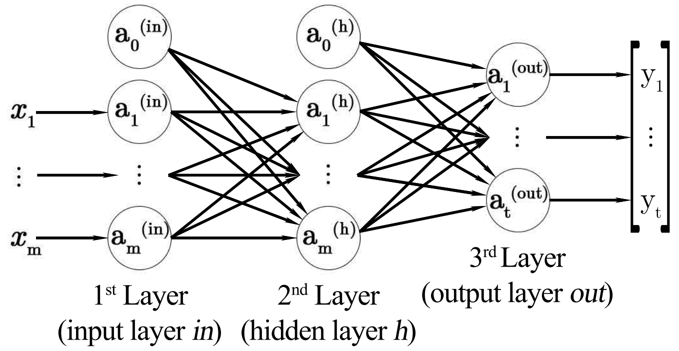


Figure 2.23: *ANN with one hidden layer.*

[36]. To explain back propagation, an ANN with a single hidden layer as shown in figure 2.23 will be used as an example.

The error is computed using the derivative of the cost function as discussed in section 2.2.1 on page 26 and can be expressed as:

$$\delta^{(out)} = \boldsymbol{A}^{(out)} - y \tag{2.32}$$

where $\boldsymbol{A}^{(out)}$ is the activated signal of the output layer, or $\hat{y}$. Working backwards through the network, the activations of each layer can be calculated using the following relationships:

$$\begin{aligned}
\boldsymbol{A}^{(out)} &= \phi\big(\boldsymbol{Z}^{(out)}\big) && \text{Activation of the output layer}\\
\boldsymbol{Z}^{(out)} &= \boldsymbol{A}^{(h)}\boldsymbol{W}^{(out)} && \text{Net input of the output layer}\\
\boldsymbol{A}^{(h)} &= \phi\big(\boldsymbol{Z}^{(h)}\big) && \text{Activation of the hidden layer}\\
\boldsymbol{Z}^{(h)} &= \boldsymbol{A}^{(in)}\boldsymbol{W}^{(out)} && \text{Net input of the hidden layer}
\end{aligned}$$

These relationships repeat all the way back to the input layer so that the whole network can be expressed as a series of nested functions. The chain rule is used to facilitate the back propagation of error through a network by collapsing the complexity of the required computations. The main building block of the chain rule is the derivative of a nested function:

$$\frac{d}{dx}[f(g(x))] = \frac{df}{dg}\cdot\frac{dg}{dx}$$

The error vector, $\delta^{(out)}$ of the output layer is used to calculate the error of the hidden layers, $\delta^{(h)}$, by propagation, transforming the errors backwards through the network:

$$\delta^{(h)} = \delta^{(out)} \left(\boldsymbol{W}^{(out)} \right)^T \odot \frac{\partial \phi(z^{(h)})}{\partial z^{(h)}} \tag{2.33}$$

After obtaining the error terms, δ, for every layer, the derivative of the cost function can be calculated:

$$\frac{\partial}{\partial w_{i,j}^{(out)}} J(\boldsymbol{W}) = a_j^{(h)} \delta_i^{(out)}$$

$$\frac{\partial}{\partial w_{i,j}^{(h)}} J(\boldsymbol{W}) = a_j^{(in)} \delta_i^{(h)}$$

The partial derivatives are then accumulated for every node in each layer and the regularisation term is added:

$$\Delta^{(out)} = \left(\boldsymbol{A}^{(h)} \right)^T \delta^{(out)} + \lambda^{(out)}$$

$$\Delta^{(h)} = \left(\boldsymbol{A}^{(in)} \right)^T \delta^{(h)} + \lambda^{(h)}$$

The weights can then be updated using the computed gradients:

$$\boldsymbol{W}^{(l)} := \boldsymbol{W}^{(l)} - \mu \Delta^{(l)} \tag{2.34}$$

for every layer l.

2.2.3 Convergence

Dropout

Dropout [37] layers were developed to address the challenging task of optimising a network's capacity. If a network has too little capacity it cannot emulate a complex system and with too much capacity it may overfit the data impacting its performance on unseen data.

Dropout layers build redundancy in a network by randomly dropping the hidden units of a higher level and scaling the remaining activations to compensate. This forces the network to learn a redundant representation of the data making it more robust. The dropout layer is only active during training so that all redundant neural pathways are used when processing new data.

Dropout layers are commonly employed in large networks that have too much capacity. By optimising the dropout rate, the extra capacity of the network can be used for redundant pathways instead of overfitting.

Decay

Higher learning rates allow a model to bypass local minima and converge faster on the global minimum but is unlikely to get close due to the larger weight updates. When close to the global minimum, a smaller learning rates lead to better *fine tuning*. This concept is illustrated in figure 2.25 on page 27.

Decay decreases the learning rate gradually during training to converge faster on the global minimum and then facilitate fine tuning once its there. This is normally done by reducing the learning rate by a fixed proportion after every training iteration or it can be done by reducing it in response to the training metrics. However, the decay rate needs to be selected carefully; too large and the weight updates will reduce too quickly effectively terminating the learning process prematurely, too small and the model will converge on the global minimum quickly but will take a long time to finish fine tuning.

Momentum

Classical momentum (Polyak, 1964) is a technique to accumulate a velocity vector in the directions of consistent reduction in the objective across training iterations. This accelerates gradient descent by boosting weight updates using the velocity vector, reducing the risk of converging to a local minima.

The drawback of momentum based techniques is that this ability to avoid local minima will also effect its convergence on the target minimum. *Nesterov momentum* adds a partial update of the gradient before adding momentum to the weight update. This limits the effect of momentum by using the previous gradient to anticipate the next update, allowing for smoother convergence.

Optimisers

DNNs are generally much harder to train than shallow learning algorithms due to the much larger number of parameters that need to be optimised. The cost function can be rough and the optimiser can get trapped in a local minimum. Subsequently, the gradient descent optimisation algorithm described previously can converge very slowly, if at all. While the general principle has remained the same, new techniques have been developed to address this issue.

Stochastic Gradient Descent (SGD) [38] optimisation uses approximations of the cost from single training samples (online learning) or small subsets of samples (mini-batch learning). Updating more frequently can lead to faster convergence and the stochasticity introduced by the cost estimations is beneficial to the training process as it can help escape local minima.

Adaptive Gradient (Adagrad) [39] optimisation tracks the frequency at which individual weights are updated and modifies the learning rate accordingly. A weight which is updated frequently will have its learning rate reduced and one which is rarely updated will have a large learning rate. **Adadelta** is based on the same concept but the update frequency is only tracked through a moving window, limiting the extent of the optimisers memory.

The central idea of **Root Mean Square Propagation (RMSProp)** is to keep a moving average of the gradients squared for each individual weight. For each update, the gradient is divided by the square root of this average so that all weight updates are scaled according their respective average gradients. This means the resultant gradient magnitude will be relative to the local area so that the size of the weight updates, irrespective of the actual gradient magnitude, will always be relatively similar.

Adaptive Moment Estimation (Adam) tracks the rate of change of the gradient (second moment) over a fixed time period in the form of a velocity vector. The Adam optimiser was the first to integrate classical momentum to accelerate gradient descent. **Nesterov Adaptive Moment Estimation (Nadam)** [40] uses Nesterov moment instead.

The redundancy of the dataset and the complexity of the system the model is learning to emulate will determine which of the optimisers performs best. Due to the size and complexity of most datasets used in machine learning, choosing an optimiser is best accomplished through trial and error rather than by inference.

DNNs are general highly over parameterised which means that there are many minima in the cost function. While there is only one global minimum, many others may yield a low enough cost to satisfy the model requirements. Therefore, the model does not need to find the global minimum, only *a* minimum that meets the performance requirements.

Vanishing gradient problem

Activation functions which converge on a constant value, like the sigmoid and the tanh functions, suffer from the vanishing gradient problem. With gradient-based learning methods, the weights are updated by a factor proportional to the gradient. These functions have gradients approaching zero for large positive or negative values of z. This slows down and may even stop the learning process. The vanishing gradient problem makes training a model based on these functions more difficult as extra caution must be taken when initialising weights and learning rate or the learning process might effectively terminate prematurely.

For this reason sigmoid activation functions are not used in neural networks. However, while tanh activation functions suffer from the same drawbacks, it is still used in recurrent neural networks (RNN) due to other properties of the network that compensates for the vanishing gradient problem. RNNs are outside of the scope of this thesis and the reasons behind why tanh works will not be discussed here.

One solution to this problem which has been widely adopted are rectified linear activation functions. It has been shown that using a rectifying non-linearity is the single most important factor in improving the performance of a recognition system [41]. The rectifier activation function also allows a network to easily obtain sparse representations which leads to mathematical advantages [42]:

- **Information disentanglement** - a dense representation is highly entangled because almost any change in the input modifies most of the entries in the representation vector. Instead, through sparse representation the set of non-zero features is almost always preserved during small changes in input.
- **Linear separability** - sparse representations are more likely to be linearly separable, or more easily separable, simply because the information is represented in a higher-dimensional space.
- **Distributed but sparse** - while dense distributed representations are the richest representations, sparse representations' efficiency is still exponentially greater, where the power of the exponent is the number of non-zero features.

Despite the hard threshold at 0, networks trained with the rectifier activation function can find local-minima of greater or equal quality than those obtained with smooth activation functions, and computations are generally cheaper as there is no need to compute an exponential function [42].

Hyperparameters

An ANN is limited in its capacity to learn by its architecture. While the learning process can optimise the trainable parameters to fit a task, the architecture itself does not change. **Hyperparameters** are properties of the model or optimiser that must be selected by the user. For a model, these include: number of units in a dense layer, filter sizes, padding, etc. And for the optimiser this includes: optimisation algorithm, decay, training time, etc.

2.2.4 Fine tuning pretrained networks

The feature extractor of a pretrained network is finely tuned to find salient features in the original dataset. If the dataset is large and diverse enough, this feature set can be applied to a wide range of new image analysis tasks. As discussed in section 2.1, the feature extractor converts the image into a feature space and the densely connected neural network finds the relationships between these features and the ground truth of the samples. For a new task, the neural network must be retrained to find new relationships that means it need to be retrained from scratch.

During the early stages of the training process the error rates will be large because, even if the feature extractor is finding salient features, the neural network is still learning the relationships. If all parameters in the network are trainable, the large error rates due to the neural network will cause weight updates in the feature extractor that are detrimental to the system in the long term. Therefore, the parameters of the feature extractor are frozen during training of the neural network. Once the neural network has been sufficiently optimised, the parameters of the feature extractor could be unfrozen and the whole network could be fine tuned. However, this can have unpredictable results as the response of neural network to a change in the feature space is unknown.

The feature extractor is optimised on a different dataset than that which may be used for the new task. Therefore, the signal distribution will be different. To compensate for this, the parameters of the batch normalisation layers in the feature extract are often unfrozen during the learning process so that the feature space adapts to the new dataset.

2.2.5 Metrics

Metrics, unlike loss, are used purely for model interpretation and evaluation. Metrics are normally used to derive a form of error that is more easily interpretable to a human being. The type of metric used is determined by the format of the model output and the domain in which it's being used.

Classification

A confusion matrix is a table that is used to describe the performance characteristics of a classification model. The table is a comparison of the of the predicted classes and the actual classes. It consists of four values: true positives **TP**, true negative **TN**, false positives **FP**, and false negatives **FN**. The true values indicate the number of samples that were classified correctly, while the false values are those that weren't. The positive or negative notations reflect the actual class of the sample. The most common metrics of a classification model can be calculated using these values.

	Predicted True	Predicted False
Actual True	True Positive (**TP**)	False Negative (**FN**)
Actual False	False Positive (**FP**)	True Negative (**TN**)

Table 2.1: *Confusion matrix.*

Accuracy of a classification model is measured as the percentage of correctly predicted samples. For a binary classification task, **binary accuracy** is simply the number of true positives and true negatives over the total number samples:

$$Accuracy_{binary} = \frac{TP + TN}{total} \tag{2.35}$$

where the total number of samples is $TP + FP + TN + FN$. For k-classes classification, **categorical accuracy** is the sum of the true positives of all the classes over the total number of samples:

$$Accuracy_{categorical} = \frac{\sum_{i=1}^{k} TP_i}{total} \tag{2.36}$$

However, accuracy metrics assume that a misclassifications are equally important in all cases. In cases where a false negative is acceptable but a false positive is not, or vice versa, accuracy provides little information. To differentiate in these cases, **recall** and **precision** are used.

$$Precision = \frac{TP}{TP + FP} \tag{2.37}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.38}$$

Precision is the percentage of the positive predictions that were correct while recall is the percentage of the positive class that were correctly predicted. For example, if there are 20 samples and 10 are of a positive class, a model that predicts only 5 samples are positive but they are all correct would have a precision of 100% but a recall of 50%. On the other hand, if the model predicted all samples are positive it would have a precision of 50% but a recall of 100%. These metrics are commonly used when a classification task places different levels of importance on different mistakes. These metrics are consolidated in the **F1** metric.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.39}$$

Yielding a single value, the F1 metric is much easier to use to compare models unless model precision or recall are the only characteristic being optimised. **Custom** metrics are also commonly uses, generally as different combinations of TP, FP, FN and TN for more domain specific applications. For example, if a value can be assigned to each classification results, the metric could be uses to design the model to maximise value.

Regression

Since the output of a regression model is a continuous value, the loss functions can be interpreted directly as a metric. However, other than minimising this value it has no meaning in the real world and can be hard to interpret. Both MSE and MAE, discussed in section 2.2.1 on page 28, are common metrics used to evaluate regression model performance characteristics. MSE is most commonly used because it penalises outliers more severely, which tends to lead to better model generalisability, a concept that will be discussed in the next section. While MAE is easier to interpret since the units are the same as the value predicted, it does not emphasize the error due to outliers. Root Mean Square Error (RMSE) is the square root of the MSE but the units are the same as those of the predicted value making it more easily interpretable while retaining the benefits of MSE.

2.3 Dataset

The quality of a dataset is determined by how closely the distribution of data matches that which the system could be exposed to. As per standard practice in data driven modelling, a dataset should be split into two sets. A **training** dataset and a **test** dataset. The reason behind this and the uses of these sets will be discussed in this section.

2.3.1 Test dataset

The creation of a test dataset not used in the training process is an essential check for an overfitted model. Consistent performance of a model when exposed to new data - its **generalisability** - is one of the key performance indicators used to evaluate a model.

The size of the test dataset, as a percentage of the total dataset, depends on the total number of samples and how evenly they are distributed. As long as the training dataset is a representative set of real world data, the test dataset can be arbitrarily large. Therefore, the larger the dataset the more data can be set aside for testing

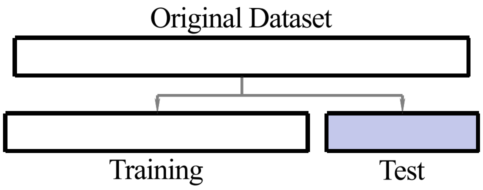


Figure 2.24: *Creating a test dataset.*

assuming constant distribution. In normal scenarios, where data is limited, the size of the test dataset must be tailored to the task.

Evaluating a model's generalisability requires the test dataset to be sequestrated for the duration of the training process and used only in the final step. The model should be evaluated on test dataset as sparingly as possible since each evaluation yields information about the test dataset which may be used, even inadvertently, in any further design iterations. The model will then *know* something about the test dataset which voids the assumption that the evaluation on this dataset represents the models generalisability.

2.3.2 Cross-validation

To avoid overusing the test dataset, a validation set is made as a subset of the training set. This is called **holdout cross-validation** and is used to evaluate the model during the training process thus limiting the information leakage from the test set. While the evaluation on the test dataset defines a models generalisability, iterative testing on the validation set throughout the training process offers key insights into when a model starts to overfit the data.

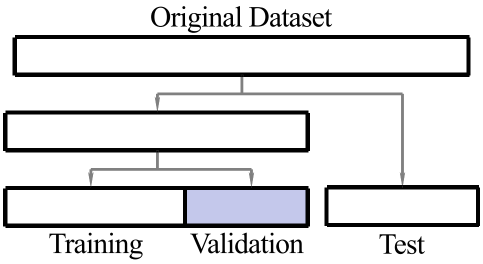


Figure 2.25: *Holdout cross-validation.*

Using the same validation dataset throughout the training process can lead to more rapid overfitting of the training dataset due to its reduced size and is sensitive to the partitioning of the dataset. **K-fold cross-validation** avoids this problem by using a different validation dataset for every iteration. The training dataset is split into

k equal sets where one set is used for performance evaluation and $k-1$ are used for model training. The process iterates through using each fold as the validation set.

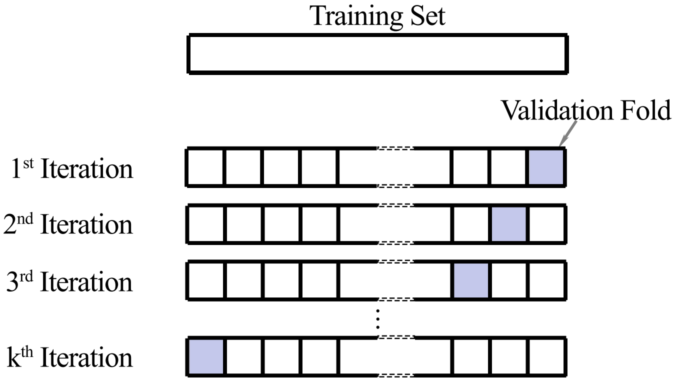


Figure 2.26: *K-fold cross-validation.*

2.3.3 Data preparation

Image preparation

Before applying machine learning to an image dataset, it is common practice to enhance the images. These processes tend to adjust image intensities to enhance contrast in the hope of making features more defined and, therefore, easier to find. **Histogram equalisation** is a common technique to increase the global contrast of an image. The adjustment aims to equalise the pixel intensities using the image histogram and is particularly effective on images which have either too high or too low exposure. It does so by defining each channels cumulative distribution function which can be used to transform the image into a new image with a flat histogram.

Image augmentation

Image augmentation is a process of generating synthetic samples to be used in the training process. Images from the original dataset are modified to create new images which are treated as separate samples. These augmentations include transforming the image by: zooming, shearing, rotating, and scaling the original image.

The new image will have similar feature to those found in the original image but not identical. This procedure expands the distribution of features to which the model is exposed to in the training process improving its generalisability. However, radical image augmentation could result in impossible features which could have unpredictable results on the model performance. Therefore, image augmentation can be used to boost an existing dataset but care must be taken in defining the augmentation parameters to that the new images still fit within the real world distribution.

2.3.4 Class imbalance

If samples from one or more classes are over-represented in a dataset - the number of samples per class differ substantially - it is considered imbalanced. RBR in Norway has been observed in approximately 20% of Norway Spruce. Without accounting for class imbalance, predicting that every tree is rot free would achieve 80% accuracy which may be the solution the machine learning process finds. Therefore, a class imbalance can affect the learning process creating a bias towards the majority class or classes but there are methods to mitigate these effects by class weighting, upsampling the minority class or downsampling the majority class, or by generation of synthetic training samples to balance the classes.

Chapter 3

Materials

The scope of this thesis was to evaluate the potential of RBR identification through image analysis from a harvester head mounted camera. Due to the technical complexity, time and rough conditions, mounting a camera on an operational harvester head was deemed to be too risky without preliminary results to validate the investment. Subsequently, images which emulate the perspective of a camera in this position were produced by cropping the images already acquired for the crane mounted camera perspective.

Images of stumps of recently harvested trees were collected as described in section 3.2 on the next page, for the original purpose of stump detection as part of the project described in section 3.1. Masks were created according to criteria outlined in section 3.3 on the next page for pixel-wise labelling of the stump and, if present, RBR. The dataset was then modified to simulate the perspective investigated in this study as described in section 3.4 on page 45.

3.1 PRECISION Project

Access to this dataset was granted by NIBIO as coordinator of the PRECISION project - *Precision forestry for improved resource utilization and reduced wood decay in Norwegian Forests* (project number NFR281140) - which is studying all aspects of RBR in the Norwegian forest industry and investigating potential solutions through methodology and technology. One work package within the project has been investigating the potential of a crane mounted camera for identifying and classifying stumps with the aim of mapping the presence and location of RBR. The dataset acquired from this work package was made available for this thesis study.

Study area

The study was carried out in Ås municipality in Norway (59°39'N 10°47'E, 0-220m above sea level). The forest in this area is boreal and dominated by Norway Spruce (*Picea abies* (L.) H. Karst), with Scots Pine (*Pinus sylvestris* L.) and Birch (*Betula pubescens* Ehrh.) as the other primary species.

3.2 Data collection

Several sites were used for data collection within the Ås municipality. All images were collected from these sites within a few days of harvesting to ensure that the stump surface had not yet been altered by environmental conditions. The images were taken at differing distances, angles, times of day and with varying degrees of occlusion to simulate a crane mounted camera, as shown in figure 3.1.

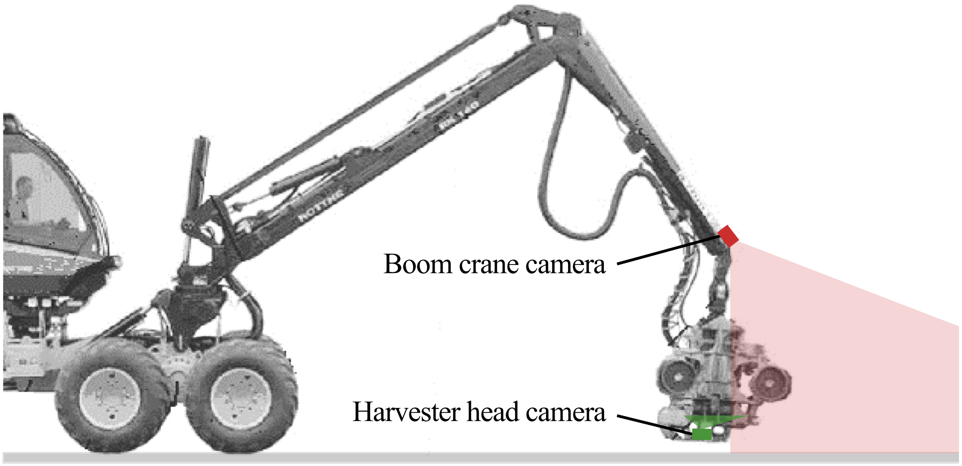
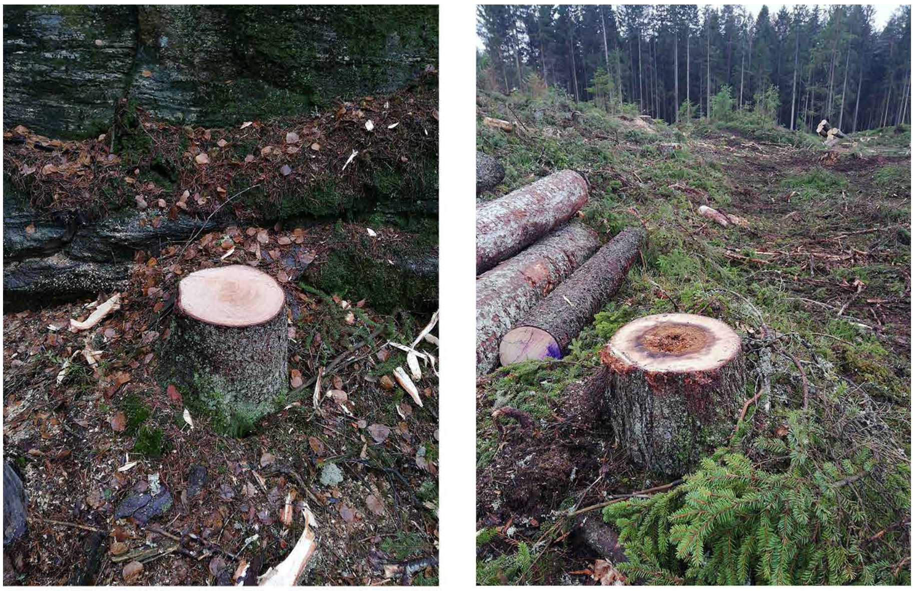


Figure 3.1: *Crane and harvester head cameras. Figure adapted from [43].*

These images were collected using a Canon EOS 100D, an Apple iPad Air and a Huawei Mate 10 Lite mobile phone. All three devices had different resolutions, capabilities in low light conditions and image processing techniques that resulted in images with a wide range of exposures, levels of contrast and focus.

3.3 Mask creation

The masks for this image set were created using GIMP raster graphics editor whereby pixels belonging to the stump, any wood discolouration, and the background were labelled separately. Any discolouration present on the stump was



No RBR

RBR

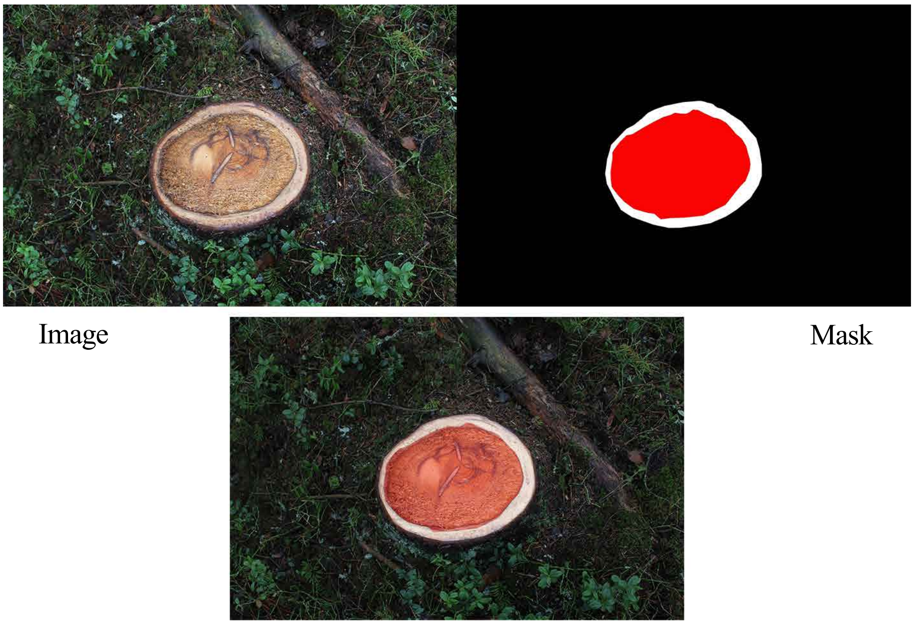
Figure 3.2: *Original image examples*

presumed to be due to RBR, however, obvious contamination was excluded, e.g oil marks from the chain saw, tracks from tire chains. Scots Pine has naturally darker colouration in the heartwood, which can be misinterpreted as RBR. To minimise label mistakes, a domain expert was consulted to outline the physiological indicators of RBR and all masks were cross checked. An example of an image with the label mask is shown in figure 3.3 on the following page.

3.4 Data preparation

A camera mounted in the harvester head would be very close to the butt of the log giving it an unobstructed view. By carefully selecting the field of view of the lens, an image could be acquired covering the full extent of, but limited to, the butt of the log.

The images acquired for the PRECISION project are much more varied in perspective than those of a harvester head mounted camera. And, they are images of the stump, not the butt of the log, but the saw only removes a few centimetres of wood during felling so the stump and butt of the log are identical for all practical purposes. By using the mask created to delineate the stump, the image can be cropped to reduce the field of view to that which can be expected from the harvester head. This procedure is covered in section 4.2.1 on page 48



Image

Mask

Mask overlaid on image

Figure 3.3: *Image/mask pair*

The resulting dataset is comprised of 1010 RGB images of which 507 have areas considered RBR.

Chapter 4

Methods

The methodology used in this study is made up of four phases: preprocessing, data selection, data handling, and model selection and evaluation. The preprocessing of the data covers the generation of the cropped images from the original dataset and image enhancement in section 4.2. Data selection will cover the creation of a test set and the creation of folds for cross-validation in section 4.3 on page 50. Data handling will be discussed with the implementation of mini-batch learning and data augmentation in section 4.4 on page 51. The training process involving model comparison, selection and optimisation is covered in section 4.5 on page 52. The code used for this study can be found on GitHub at `https://github.com/tynowell/rotdetection-CNN.git`.

4.1 Software

This study used Python version 3.6.4 on an Anaconda platform with Numpy version 1.15.4. Scikit-learn [44] version 0.20.2 and Keras [45] version 2.2.4 - with Tensorflow [46] version 1.12 as the backend - were used for data handling and machine learning tasks. This process used GPU acceleration with CUDA 9.0 and cuDNN 7.4.2.

4.2 Data preprocessing

The dataset was relatively well prepared as it had already been used for a machine learning application [47]. All data had been consolidated into two folders for images and masks and the masks had been given the same file name as the originals.

4.2.1 Image cropping

The masks created for each of the original images were used to define a bounding box limited to the extent of the cross-section of the stump. This area was then cropped to create a new image and a new mask as shown in figure 4.1.

Bounding box

The masks were stored as JPEG images where background, stump and RBR were labelled black, white and red respectively. To obtain a bounding box, the image was binarised to consolidate the area designated as the stump including any RBR. The image was read into a Numpy array and all non-zero (non-black) pixels were identified. The row and column minimums and maximums were then used to define the extent of the bounding box.

Cropping

The bounding box was then used to crop both the image and the mask, which were saved to separate folders. The files were also named using a zero padded four digit numerical sequence so that order could be checked more easily.

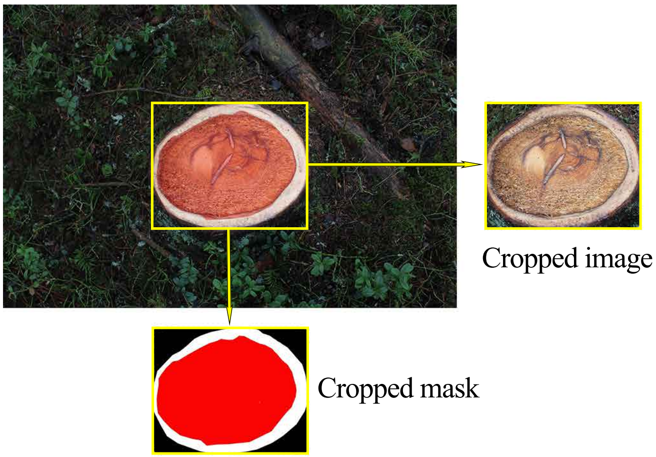


Figure 4.1: *Cropping the images and masks*

4.2.2 Target extraction

The cropped mask was then used to extract label data for the images: a Boolean value indicating the presence of RBR and the percentage of RBR by surface area.

Antialiasing mask

The masks were stored as JPEG files and the compression process caused aliasing along the borders of the designations. When analysing masks for the presence of RBR, it was found that the aliasing noise was at times closer to red than white or black. This meant red pixels were detected in masks with no RBR labelled. To fix the problem, the masks were thresholded to eliminate the noise and separate the labels using OpenCV's binary threshold function, *threshold*.

Extracting targets

Once the mask labels had been separated, the presence of RBR and the percentage of RBR were extracted. The percentage of RBR was calculated using equation 4.1.

$$RBR\% = \frac{p_{rot}}{p_{rot} + p_{clear}} \tag{4.1}$$

Where p_{rot} is the number of red pixels and p_{clear} is the number of white pixels. The presence of RBR was determined simply as a Boolean value from the percentage RBR. This data and the image names were aggregated in a Pandas dataframe, which was used later for dataset selection as described in section 4.3.1 on the following page.

4.2.3 Image enhancement

Due to the variation between cameras used to collect the images, a new dataset was generated with normalised images. Histogram equalisation was applied to the entire dataset of cropped images in an attempt to limit the range of exposures and levels of contrast. This was applied to the cropped images using OpenCV's histogram equalisation function, *equalizeHist*. However, since the pretrained networks are trained on the ImageNet dataset without image enhancement, both the original and enhanced dataset were kept to be compared.

4.3 Data selection and cross-validation

As discussed in section 2.3 on the topic of datasets, creating training, validation and test sets are essential for evaluating a model and ensuring generalisability.

4.3.1 Set selection

The datasets used in this study were chosen using Scikit-learn's *train_test_split* function. This function splits a dataframe into two, which contain the training and test sets. This function was also set to stratify the datasets, that is, it ensures that both sets have the same proportion of a given feature, which is the Boolean value for the presence of RBR in this case. A test set of 200 images was deemed suitable for evaluating generalisation without compromising the distribution of the training dataset. Once the test set had been chosen, the dataframes were used to split the images into a *train* folder and a *test* folder.

4.3.2 K-Fold cross-validation

Cross-validation should be nested within the training process such that the validation set changes for every training epoch. However, this functionality has not been integrated into the training tools provided by Keras. Subsequently, the folds were treated as independent evaluations of the model architecture rather than as a means to improve the generalisation of a single model.

To ensure statistically robust results for comparison, 10-fold cross-validation is used wherever possible. Ten dataframes were created using *train_test_split* with stratification on the training set to create new training and validation sets. These were then concatenated back into a single dataframe. As will be discussed in section 4.4.1 on the next page, the mini-batch learning function used accepts a dataframe and uses a given proportion as the validation set. The validation set is always selected from the end of the dataframe so by passing it a dataframe with the desired validation set concatenated to the end, the correct selection is always used. A validation set size of 160 images was chosen (19.8%) as around 20% is a common size in machine learning work flows, leaving 650 images for training.

4.4 Data handling

4.4.1 Mini-batch learning

This study aimed to test a wide range of architectures which required a large number of training cycles. The computation time involved in this process would have been impractical on a standard CPU. Therefore, GPU accelerated computations were enabled using Tensorflow's GPU package. While this had a large impact on computation time, the memory space was restricted to that of the GPU RAM (8 GB). The models and the dataset were too large to load into this memory space so mini-batch learning was used. This was implemented using the *flow_from_dataframe* function in the Keras package, one of the *flow* functions that use mini-batch learning. Larger batch sizes normally lead to better generalisability of the model. However, the batch must fit into the GPU memory along with the model and optimisation parameters. Due to constraints in GPU memory, the batch size had to be limited. Along with the chosen image resolution of 128×128, maximising the memory usage allowed for a batch size of 10.

This function uses a Pandas dataframe containing the file name and target value for each sample. By defining the file names in the dataframe, the training images do not need to be copied into separate folders with different splits for 10-fold cross-validation unlike the *flow_from_directory* function.

4.4.2 Data generation

The *flow_from_dataframe* function accepts an *ImageDataGenerator* to do mini-batch learning. The generator converts the images into batches of tensor image data and a set of augmentations to be applied for each training iteration. The augmentations are applied at random within the defined constraints for each batch. The values chosen from test runs and by visual inspection are shown in table 4.1.

Augmentation	Value
Rotation range	90
Shear range	0.3
Zoom range	0.3
Horizontal flip	True
Vertical flip	True
Rescale	1/255

Table 4.1: *Augmentation parameters for the data generator*

The *rescale* value is constant across training iterations. It is a scaling factor applied to all pixels to reduce the range of values from 8 bit (0 to 255) to values between 0 and 1. In defining the ImageDataGenerator, the validation split is also required. This was set to 0.198, the same used before when defining the validation folds for cross validation. A separate ImageDataGenerator can be defined with no augmentations, only rescaling, for the test dataset.

4.4.3 File structure

The images were split into two folders for the originals and for those enhanced with histogram equalisation. The enhanced images were given the same name as the original so that only the file path needed to be changed in the dataframe fed to the flow_from_dataframe function.

4.5 Deep learning

CNNs capable of learning subtle textural or structural differences in images are generally very deep. Designing a network with so many layers can be very time consuming due to the complexity and iterative nature of the process. Therefore, this study started with preexisting architectures, which have proven successful at similar tasks, to identify the best core architecture.

The full process will be described in the rest of this section but the following is a short summary of the model selection and training methodology:

1. Acquire pretrained networks to test.
2. Modify the top layers for the new task.
3. Compare performance of all architectures on limited training.
4. Compare performance of the best architectures.
5. Optimise the best architecture.
6. Retrain with optimal parameters on the full training set.

4.5.1 Architectures

ImageNet

ImageNet [48, 49] is an open source image database containing 14,197,122 labelled images in 21,841 synsets at the time of writing. It has been used extensively for training CNNs both due to the availability and scope of the dataset and because of the ImageNet Large Scale Visual Recognition Competition (ILSVRC), which has been a proving ground for all state of the art architectures.

Keras Applications

Keras Applications are deep learning models that are made available along with weights of a model trained on the ImageNet dataset. For the purpose of comparison, these models are evaluated using the *Top-1* and *Top-5* metrics. The output of a multiclass classifier is a vector of the probabilities that the sample belong to each class. The Top-1 metric can be regarded as a standard accuracy metric - its the accuracy that it makes the correct class prediction. Top-5 is the accuracy of the model where a prediction is considered correct if the true class is one of the 5 with the highest probability in the output vector. However, assuming these metrics are directly applicable to any image analysis task would be a mistake. The feature space required for a particular task, like RBR detection, could be very different to that which would result from training on the ImageNet dataset. Subsequently, all models must be evaluated on the new task independent of their stated performance. Table 4.2 on the next page shows the details of the models used in this study.

The VGG [50] neural network designed by the Visual Geometry Group was the first runner up in the 2014 ILSVRC classification task, but still a significant improvement over the 2013 winner. The VGG network was the first to use stacked kernels with ReLU activation functions separated by pooling layers to reduce the number of parameters in the convolutional layers. The VGG-16 is 23 layers deep, 19 of which are the feature extractor making it one of the deepest networks at the time. The VGG-19 network is deeper but did not perform as well, which was the first indication to the community that increased depth does not always correlate with increased performance.

GoogLeNet [51], aka InceptionV1, was the winner of the 2014 ILSVRC and was improved in the following year to become InceptionV3 that came second in the ILSVRC by a very small margin. GoogLeNet saw the introduction of inception blocks of parallel convolutions, the intermediate step of the InceptionV2 network introduced batch normalisation layers and InceptionV3 saw the reduction in the number of parameters by stacking kernels to reduce filter sizes.

Microsoft's ResNet [52] network was the 2015 ILSVRC winner having seemingly solved the problem of limited network depth by using residual blocks. Subsequent improvements have seen the introduction of the *bottleneck* design which uses a pointwise convolutions before and after the normal convolutional layers to reduce the dimensionality of the operation to decrease the computational complexity. InceptionResNetV2 is based on an architecture introduced in 2016 [53], which was the first to incorporate the inception and residual blocks.

Model	Size	Top-1	Top-5	Parameters	Depth
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
InceptionV3	92 MB	0.779	0.937	23,851,784	65
ResNet50	98 MB	0.749	0.921	25,636,712	50
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
Xception	88 MB	0.790	0.945	22,910,480	126
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201

Table 4.2: *Keras Application models. The depth refers to the topological depth of the network which includes activation layers, batch normalisation layers, etc. Due to the complexity of the learned architecture of NASNet, the depth is hard to define.*

MobileNet [54] was design specifically for mobile and embedded vision applications. Its a relatively small model with reduced complexity primarily due to extensive use of depthwise separable convolutions. MobileNet (V1) generally outperforms GoogLeNet with fewer parameters and a third of the computations while achieving similar accuracy to VGG-16 with almost 27 times less computations.

The Extreme version of Inception, or Xception [55], is based on the InceptionV3 network, but uses modified depthwise separable convolutions. The original depthwise separable convolution used a depthwise convolution followed by a pointwise convolutions with a non-linear activation in between. The Xception network uses them in the reverse order, but without the non-linear activation. The model obtains similar results to the InceptionV3 model, but the reduction in the number of non-linear activation functions makes it more efficient.

Google Brain released a new concept called Neural Architecture Search (NAS) which uses a recurrent neural network to learning its own architecture using reinforcement learning thereby optimising the architecture. As a result, the NASNet [56] model was developed, but this methodology could not use the ImageNet dataset because it is too large. Instead, the CIFAR-10 dataset [57] was used.

DenseNet [58] models, based on densely connected convolutional blocks, have achieved state of the art performance on the CIFAR-10 dataset.

Top layers

The architecture of the top layers adopted the standard used for CNNs as shown in table 4.3.

Layer	Parameters
Global Average Pooling	-
Dense	**256** units
Activation	**ReLU**
Dropout	**0.5**
Output	Task specific

Table 4.3: *Top layer architecture for model comparison. The parameters to optimised in the final model are in bold.*

The parameters to optimise to the task are the number of units in the dense layer, the type of activation function and the proportion of dropout. The initial values were chosen for the model selection process with the aim of giving the top layers enough capacity to not be the limiting factor on the performance, while keeping the number of parameters down to decrease training time. The values chosen were 256 units in the dense layer, ReLU activation and 0.5 dropout.

Both the classifiers and regressors have a single unit in the output layer, but the classifier uses a sigmoid activation function to generate the output while the regressor uses a linear activation function.

4.5.2 Optimisation

To ensure a level playing field for model comparison the same optimiser was used in all cases. To decrease the chance of stochastic behaviour, whether it be advantageous or disadvantageous to the process, a momentum based optimiser was deemed more stable and the Nadam optimisation algorithm was chosen. This optimiser was chosen for the comparative phases of this process but others were tested for the final model optimisation.

4.5.3 Metrics

Classification

The metrics used during training of the classifier models were part of the *keras_metrics* package. These were the *precision* and *recall* functions. To simplify the comparison process, a single metric, F1 score, would be more convenient, but since the precision and recall of the models are useful for a more detailed comparison and the F1 score could be calculated post-training. Therefore, only the the precision and recall were computed during training. Models were evaluated using the highest F1 score attained throughout the training process.

A custom metric, referred to as the Value metric, was also created for evaluation that would be more interpretable to those who may end up using the model. A misclassification of RBR would results in the miscategorisation of the log. A sawlog categorised as pulpwood will fetch 64% of the market value and the reverse will be rejected at the mill with very little, if any, compensation. The value retained is shown in table 4.4.

	Predicted true	Predicted false
Actual true	1.00	0.00
Actual false	0.64	1.00

Table 4.4: *Custom value metric.*

This metric can be defined as:

$$valueretained = \frac{TP + TN + (FP \times 0.64)}{TP + TN + FP + FN} \tag{4.2}$$

Regression

MSE is the most commonly used metric for regression models, but it is not easily interpretable because the units are the square of the predicted value. RMSE, a standard metric in forestry research, contains the same information but the units are the same as the target value. Since there is no reason to compute both MSE and RMSE, a custom Keras metric was created to compute RMSE to be used for model evaluation. The lowest RMSE attained by the model was used in the model comparison process.

4.5.4 Model selection

All training progress was logged using *TensorBoard* from which the data was extracted and analysed.

General comparison

For a general comparison of the model architectures, a model was trained on each of the folds for 100 epochs for a total of 10 models trained for each of the 13 architectures tested. The validation results were then used to create a subset of the most promising models.

Selection comparison

The same process was repeated with the subset of models until a maximum of 300 epochs. This process also included a *learning rate plateau* and an *early stop* callback.

The learning rate plateau callback uses a sliding window to monitor a given metric. When the metric does not meet a defined criterion, the callback reduces the learning rate by a given factor. This is a form of adaptive decay. For the selection comparison, this callback was set to monitor the validation loss of the last 25 epochs, decreasing the learning rate by a factor of 0.5 if the validation loss over this time period had not decreased.

Early stop does just that. It also monitors a metric over a given period but it terminates the learning process if the criteria is not met. In this case, it was also set to monitor validation loss, but over a 50 epoch span. This means that at least one decrease in the learning rate due to the learning rate plateau callback would have to have no effect before the criteria of the early stopping callback is void and the process terminates.

The final architecture chosen to optimise is based on the validation results of this comparison.

4.5.5 Model optimisation

Hyperparameter tuning

Once the final architecture was selected the top layers of the network were optimised. The three hyperparameters to optimise were: number of dense units, the activation function and the dropout proportion as shown in table 4.3 on page 55.

The optimisation process used a grid search approach to tune the hyperparameters. This entails trying all possible combinations of the hyperparameters. Instead of including the dropout proportion in a grid search, it is added after the fact. Dropout improves the generalisation of the model through using excess capacity to build redundant pathways. By taking this into consideration, the optimal number of dense units is found first without dropout and then increased with the addition of dropout. This decreases the number of iterations required in the grid search by a substantial factor.

Preliminary tests were conducted to ascertain the range of the grid search. This involved training models on three randomly selected folds over a wide range of parameters. Once the range had been reduced significantly, a standard grid search was performed.

While there are only two hyperparameters to tune, an exhaustive grid search was deemed too time consuming with 10-fold cross-validation. Therefore, this process was limited to 4 folds, which may not be as robust a representation of the models generalised performance, but is still useful for comparison. Since the models will be relatively similar, the loss curves are expected to have similar responses. Because of this, the performance difference should be clear after relatively few epochs. Preliminary tests found that by 100 epochs the models had diverged sufficiently to evaluate their relative performance characteristics. Thus, these models were limited to 120 epochs of training.

Final training

Once these parameters were tuned, a model was trained on each of the 10 cross-validation folds to establish the optimal training time. The final model was then trained on the full training set for the designated time and evaluated on the test set.

4.6 Combined model

The combined model was set up as a pipeline where the classifier and regressor evaluated the same test dataset outputting one prediction vector each. These vectors are then multiplied element-wise to obtain a prediction vector where the percentage of any sample classified as clear of RBR is set to zero.

Chapter 5

Results

This chapter will review the results but a more detailed analysis will be covered in chapter 6. The results of the data preparation process and a description of the chosen datasets will be shown in section 5.1. The results of the image enhancement will be covered in section 5.2 on page 61. The results of the classification task will be shown in section 5.3 on page 62, and of the regression task in section 5.4 on page 64. The combine model will be covered in section 5.5 on page 67.

5.1 Data preparation

5.1.1 Dataset distribution

The RBR is reasonably well distributed throughout the dataset as shown in figure 5.1. The mean percentage of RBR present in all the images, including those without RBR, was 19.43%.

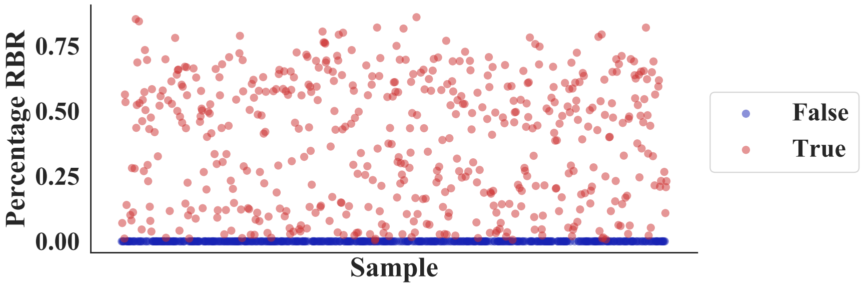


Figure 5.1: *Target distribution.*

Doing a split without randomisation of this dataset could have yielded sets with similar distributions, however random selection with stratification was used to ensure it.

5.1.2 Test dataset

The split with stratification, used to create the test dataset, yielded the distribution shown in figure 5.2.

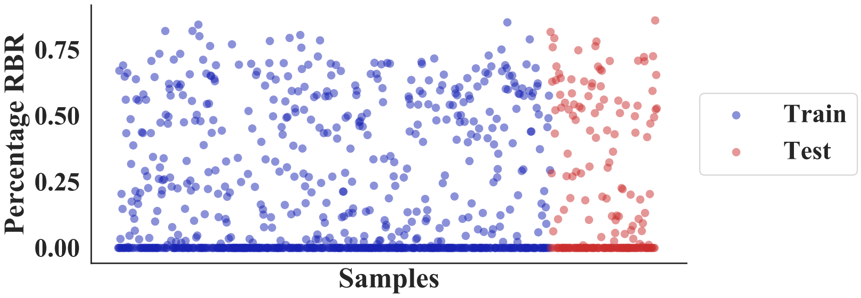


Figure 5.2: *Training and test set distribution.*

The training and test datasets have similar distributions as shown in table 5.1.

	Training	Test
Total number of images	810	200
Number of RBR images	407	100
Proportion of RBR images	0.5025	0.5000
%RBR mean	0.1916	0.2051
%RBR variance	0.0647	0.0717

Table 5.1: *Distribution of RBR presence in Train/Test split.*

The variation between the sets was deemed to be negligible.

5.1.3 Cross-validation folds

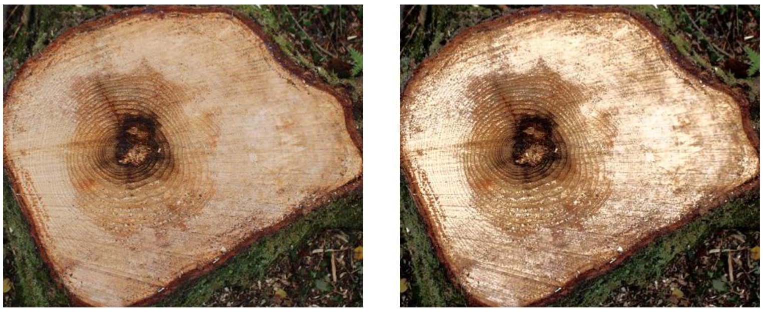
The cross validation folds were also created using stratification to ensure similar distributions. A statistical description of the folds is show in table 5.2 on the next page.

	RBR images	%RBR mean	%RBR variance
Fold 1	81	0.1968	0.0640
Fold 2	81	0.1906	0.0665
Fold 3	81	0.1819	0.0603
Fold 4	81	0.1896	0.0648
Fold 5	81	0.1764	0.0589
Fold 6	81	0.1920	0.0683
Fold 7	81	0.2033	0.0686
Fold 8	81	0.1979	0.0699
Fold 9	81	0.2132	0.0695
Fold 10	81	0.1782	0.0630

Table 5.2: *Distribution of RBR in the validation folds. The total number of images in the validation folds and the proportion of RBR images were constant at 160 and 50.62%, respectively.*

5.2 Image enhancement

The effect of the histogram equalisation function on the images can be seen in the example in figure 5.3.



Original Image

Histogram Equalised

Figure 5.3: *Histogram equalised image.*

Preliminary results shown in figure 5.4 on the next page indicated that enhancing the images had a negative effect on the performance of the pretrained networks. Some architectures had a high tendency to diverge rapidly when training on enhanced images. These outliers were excluded from the figure since the large loss values made it difficult to compare with that of the original images. Therefore, the rest of the study was conducted on the original images.

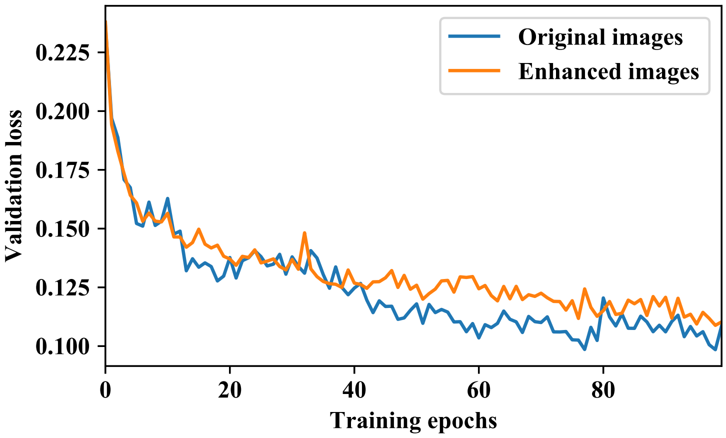


Figure 5.4: *Classifier performance on equalised images.*

5.3 Classifier

5.3.1 Pretrained models

Of the thirteen tested architectures, five were selected for further study. These results are shown in figure 5.5.

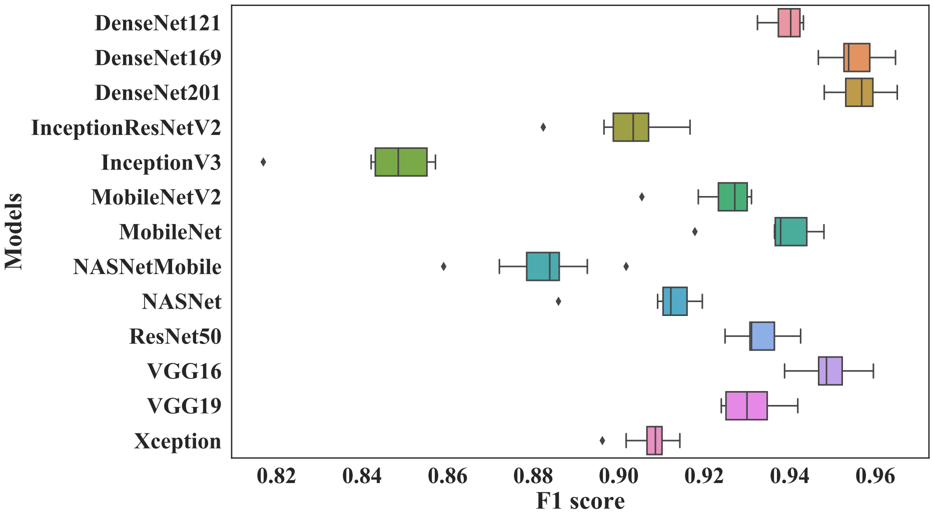


Figure 5.5: *Comparison of all classification models.*

The top five architectures were DenseNet121, DenseNet169, DenseNet201, MobileNet and VGG16.

5.3.2 Selected models

The selected architectures were then trained on all 10 folds for a maximum of 300 epochs. The results are shown in figure 5.6 where DenseNet201 had the highest mean F1 score.

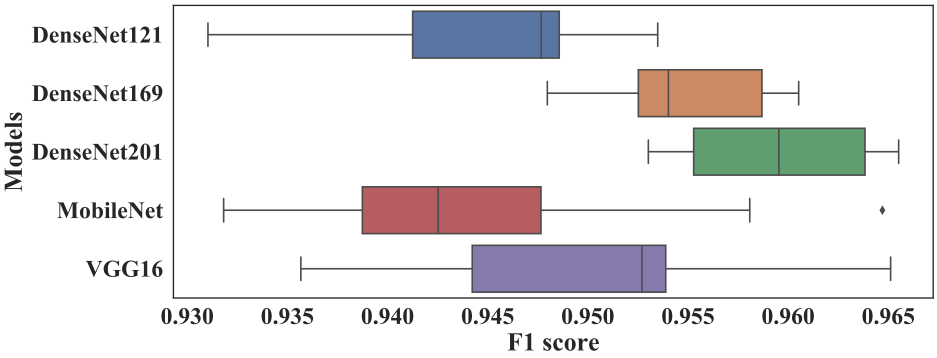


Figure 5.6: Comparison of selected classification models. DenseNet201 had the highest mean F1 score.

5.3.3 Best model

Preliminary tests found that the range of 32 to 128 dense units yielded the best results, so a grid search within this range paired with different activation function was conducted. As shown in figure 5.7, a 32 unit dense layer with ELU activation achieved the highest mean F1 score and was selected as the configuration for the final model architecture.

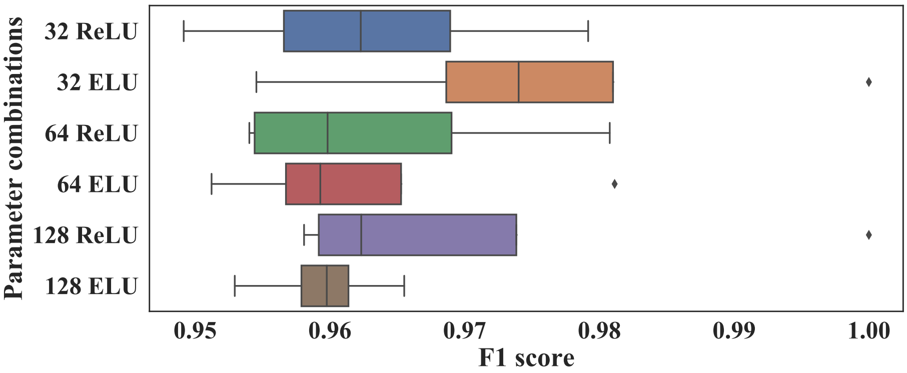


Figure 5.7: Parameter selection for classification model.

This architecture was then trained on all 10 folds for 200 epochs to determine the optimal amount of training required. As shown in the validation loss curve in figure 5.8, the optimal training time was 100 epochs.

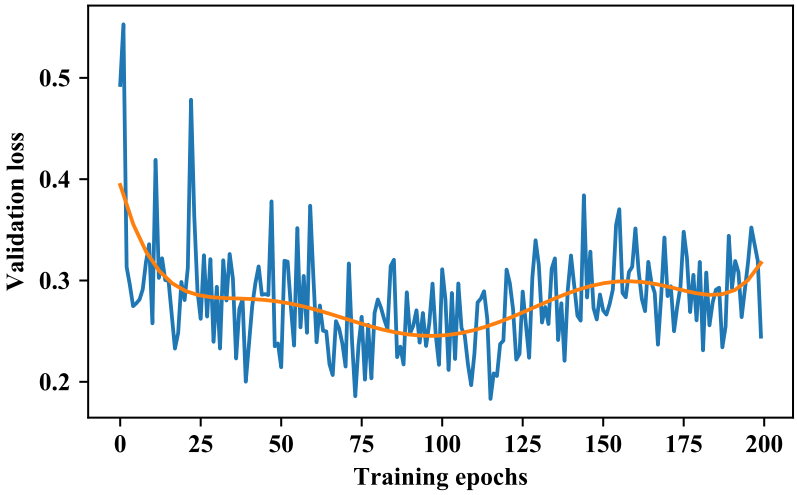


Figure 5.8: *Validation loss of chosen classifier.*

The final model was then trained on the full training dataset for the allotted time with a dropout of 0.4. The results of the evaluation of the model on the test dataset are shown in table 5.3.

Metric	Train	Test
Precision	0.9669	0.9519
Recall	0.9429	0.9900
F1	0.9547	**0.9706**

Table 5.3: *Results of the classification model.*

Using the custom metric describe in equation 4.2 on page 56, the value retained by this model was **98.6%**.

5.4 Regressor

5.4.1 Pretrained models

InceptionResNetV2 was excluded due to poor performance. The six best models were chosen for further investigation: DenseNet121, DenseNet169, DenseNet201, MobileNet, ResNet50 and Xception.

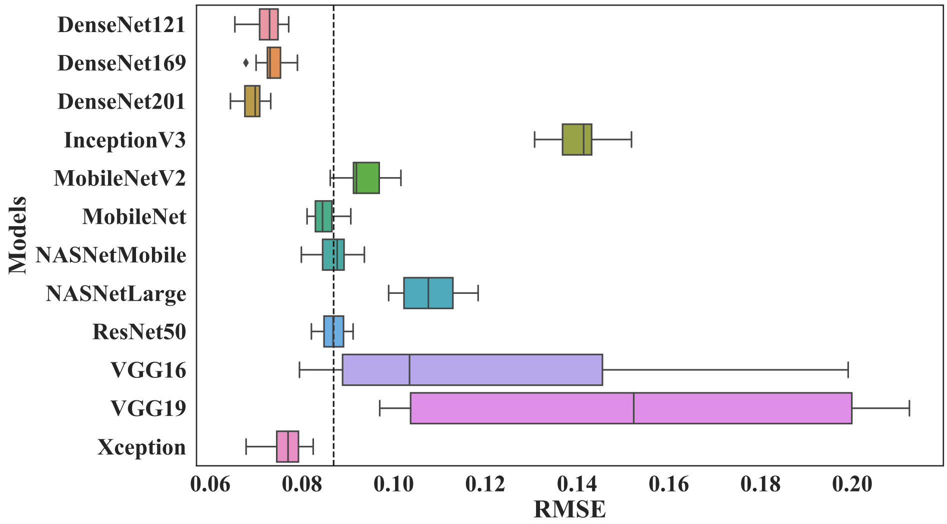


Figure 5.9: Comparison of all regression models. The model selection was limited to the top six.

5.4.2 Selected models

Of the selected models, DenseNet201 had the lowest RMSE score and was selected for the final model architecture.

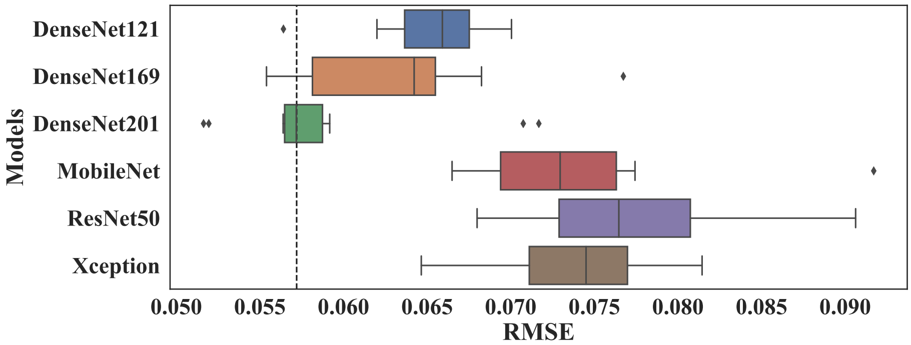


Figure 5.10: Comparison of selected regression models. DenseNet201 had the lowest mean RMSE.

5.4.3 Best model

Preliminary tests found that the optimal number of dense units fell in the range of 32 to 128. All three models based on ReLU performed better that those with ELU, with the 64 unit dense layer performing the best. Therefore, a 64 unit dense layer with ELU activation was chosen for the final model architecture.

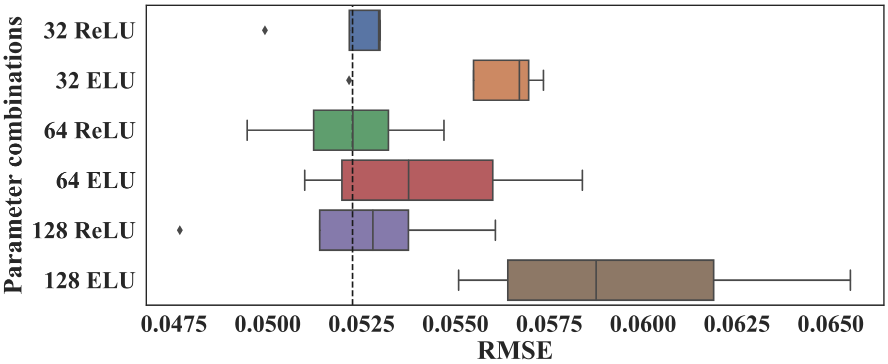


Figure 5.11: *Parameter selection for regression model. The 64 unit dense layer with ReLU activation had the best performance characteristics.*

This architecture was then trained on all 10 folds for 200 epochs to determine the optimal amount of training required. As shown in the validation loss curve in figure 5.12, the optimal training time was 190 epochs.

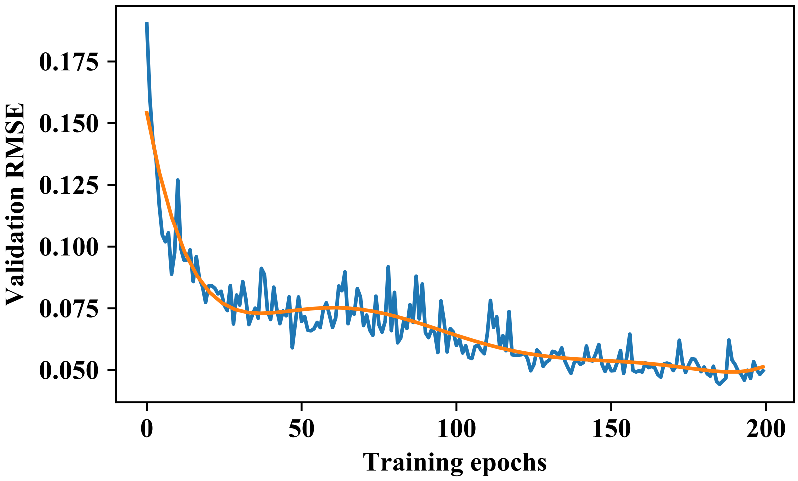


Figure 5.12: *Validation RMSE of chosen regressor.*

It was found that adding any dropout to the network had a detrimental effect of the model performance so it was excluded from the final model. It was then trained on the full training dataset and evaluated on the test dataset. The model attained a RMSE of **0.0688**.

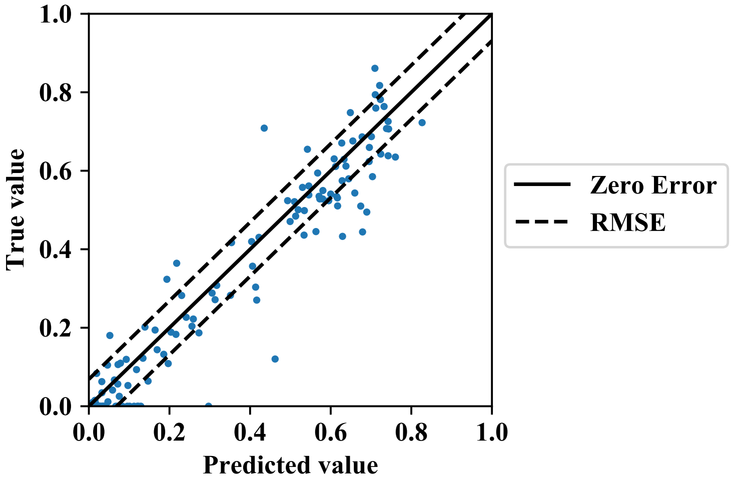


Figure 5.13: *Regression model error. The dashed lines delineate the values that fall within the RMSE.*

Figure 5.13 shows the regression prediction plotted against its true value.

5.5 Combined model

The element-wise multiplication of the prediction vectors is illustrated in table 5.4.

Sample	Regressor	Classifier	Combined	True value
0	0.001429	0	0.000000	0.0000
1	0.571155	1	0.571155	0.5349
2	0.007030	0	0.000000	0.0000
3	0.533893	1	0.533893	0.4360
4	0.025214	1	0.025214	0.0000
5	0.120256	0	0.000000	0.0000
⋮	⋮	⋮	⋮	⋮

Table 5.4: *Combined model sample predictions. This depicts the element-wise multiplication process to obtain a combined model prediction.*

Combining the models reduced the RMSE from 0.0688 to **0.0617**.

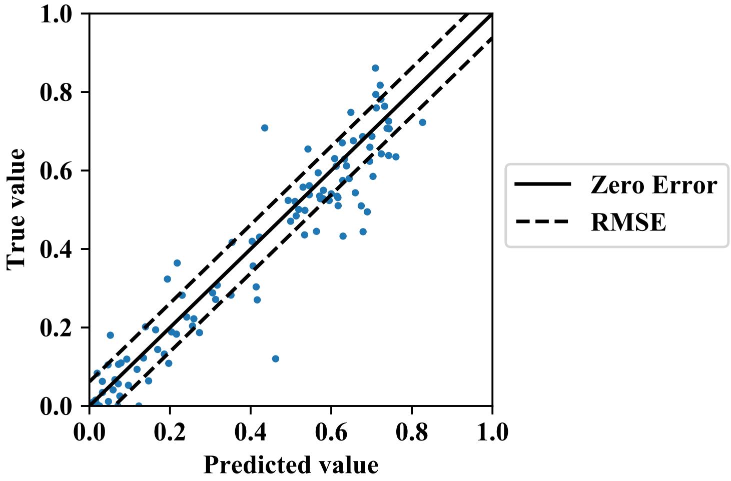


Figure 5.14: *Combined model regression error. The dashed lines delineate the values that fall within the RMSE.*

Chapter 6

Discussion

6.1 Dataset

6.1.1 Images

The original images were collected using a hand held camera in an attempt to simulate the perspective of a crane mounted camera. The surface of the stump and that of the log butt are very similar but the environments in which they are located are not. Therefore, the assumption that an image cropped from the original is a reasonable representation of this view may impact the generalisability of the model.

The criteria used for the collection of the original image dataset required that noise be added to the images. This included variable lighting conditions, saw dust and wood chips, branches and oblique angles. While a harvester head mounted camera would not have constant conditions, it would not be subject to the same extremes. An example of one of many challenging images can be seen in figure 6.1 on the next page.

The variation in distance from the stump in particular may have had a large effect on the results of this study. The relative size of the stump in the cropped image is consistent throughout the dataset since the bounding box was limited to its extent, but the cropped images themselves have a wide range of different resolutions as shown in figure 6.2 on the following page.

When the image generator resizes the images to fit the network, the resulting images can have very different features. In particular, textural features could be lost or modified in the process, introducing noise or losing information the model could otherwise use.

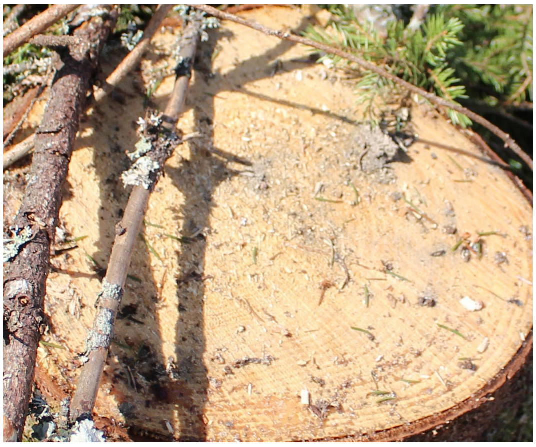


Figure 6.1: *Hard image example*

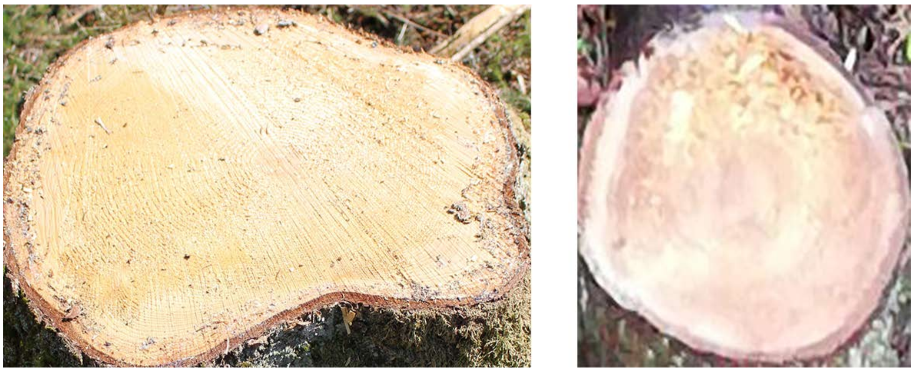


Figure 6.2: *Cropped image resolutions*

6.1.2 Masks

The label masks were produced according to a criterion stipulated by a domain expert but some discoloration not due to RBR may have been mislabelled. In some instances it is very hard, even for an expert, to classify it without more invasive tests especially when there is occlusion due to saw dust, leaves or shadows. For example, occlusion due to saw dust or wood chips can make it very hard to see discolouration of the stump or even to tell whether the discolouration is due to RBR as illustrated by figure 6.1.

The aliasing caused by the file compression for conversion to JPEG may have introduced errors in the RBR percentage value assigned to the images. Masks that returned very small value were double checked and none were found to give a false positive of RBR. However, the effect this may have had on the percentage assigned to those that do have RBR is unknown and may be significant.

6.1.3 Set selection

Using SciKit-learn's *train_test_split* function with stratification only ensures that the proportion of samples with the stratified feature are the same in both sets. In this case, that means that the proportion of images classed as RBR is the same in all sets, not that the percentage rot or any other characteristics of the images are equally balanced. Combined with the noise already present in the dataset, this caused significant variation in the cross-validation folds. In particular, the results of the first fold show significant variation to the rest. This can be seen clearly in figure 6.3.

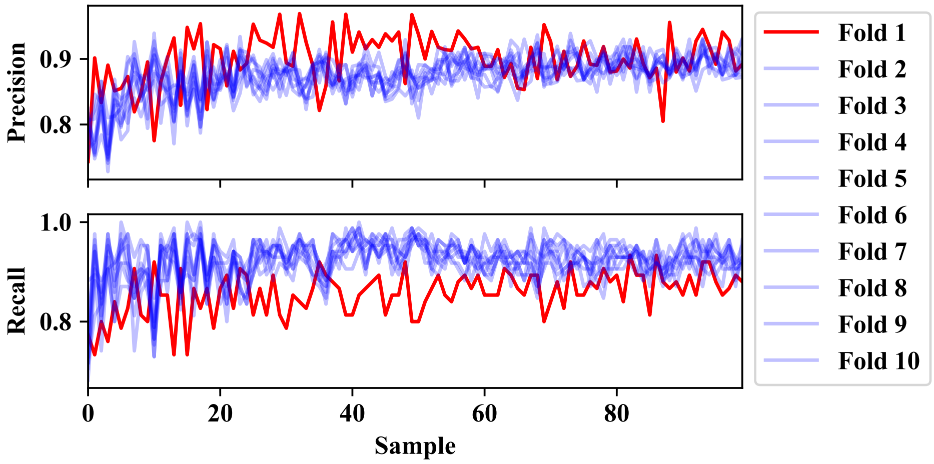


Figure 6.3: *Variation in cross-validation folds.*

6.2 Enhanced images

The enhanced images saw little to no difference in performance for the regression models and had a negative impact on the classification models. A negative impact is perhaps not that surprising since the weights for the pretrained networks used in this study were learning through training on the ImageNet dataset. The images in that set are not enhanced through histogram equalisation or by any globally applied enhancement. Perhaps the feature extractor could be retrained on enhanced images and get better results, but using the original weights seemed to render it unstable on enhanced images.

6.3 Model selection and optimisation

Using a small batch size of ten, due to memory limitations, the learning rate was relatively high since it is related to the frequency of weight updates which would be higher with more batches to process. This also introduced noise as the smaller batch sizes are less likely to be of similar distribution to the dataset as a whole. This means that the graphs of the loss and metrics of the learning process were very noisy and hard to interpret, even when averaged over all the folds, as shown in figure 6.4.

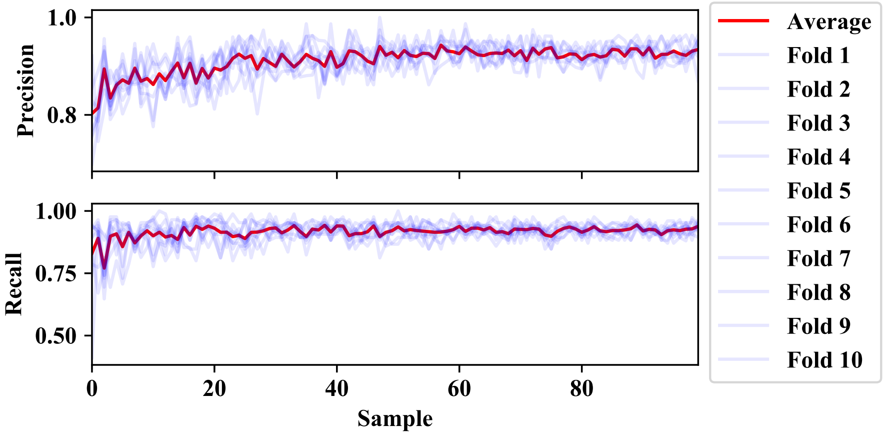


Figure 6.4: *Noise in training metrics.*

The comparison between the DenseNet models implies that an increase in depth is directly correlated with an increase in performance. Initially, this study aimed to determine the ideal architectures for these tasks with the intent of designing a network with a feature extractor tailored to the task using the defining features as a foundation. However, the findings indicated that the complexity of the task could not be limited in feature space and the original feature extractor was kept.

6.3.1 Classifier

The results of the classifier are very promising. However, a closer look at the misclassified images shows that closing the gap to 100% accuracy may be a lot more complicated. Six of the two hundred test images were misclassified. One of which, shown in figure 6.5 on the facing page, was a false positive that seems to have been misclassified due to occlusion. This was the only occluded image to be misclassified. Occluded images may not account for a significant portion of the error but it is still worth considering that occlusions such as this are unlikely to occur for a harvester head mounted camera.

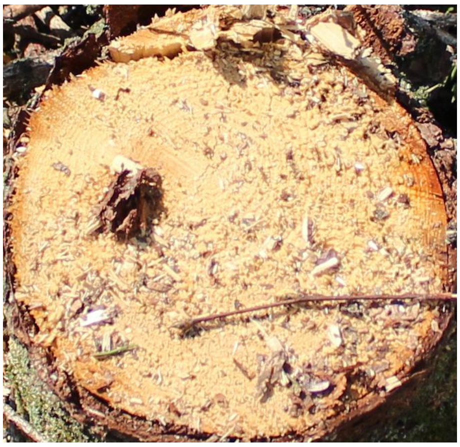


Figure 6.5: *Potential misclassification due to occlusion.*

The rest of the false positives, shown in figure 6.6 and in appendix A, have some degree of discolouration on the surface of the stump which can easily be interpreted as RBR by the untrained eye. Whether these images were mislabelled, or if the discolouration is not due to RBR, is unknown to the author.

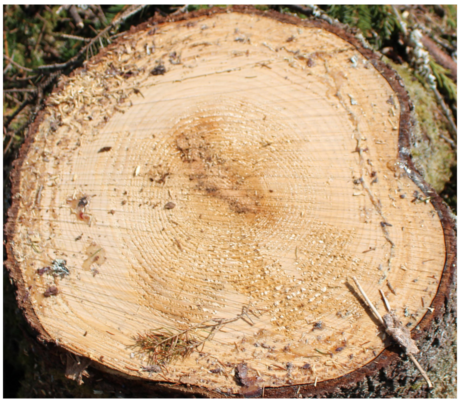


Figure 6.6: *Potential misclassification due to discolouration.*

One image was misclassified as a false negative. Shown in figure 6.7 on the following page, it appears that RBR on the stump may have been misclassified due to the uneven distribution. This error is concerning because uneven distribution of RBR is not uncommon. There are several other images that have similar patterns but were correctly classified. However, a false negative classification would have more severe financial consequences as it would be categorised as a sawlog but rejected at the sawmill with no financial compensation.

The 98.6% value score of the final classification model does not account for class imbalance. The dataset used is relatively balanced but the proportion of RBR im-

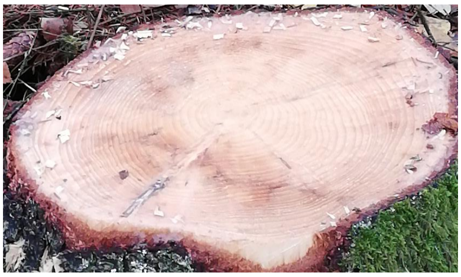


Figure 6.7: *Potential misclassification due to distribution.*

ages in the real world is not. In Norway, the proportion of trees with RBR is approximately 20%. The classifier metrics indicate that 99% of these trees and 95% of the healthy trees would be correctly classified as shown in table 6.1.

	Predicted True	Predicted False	Total
Actual True	99	1	100
Actual False	5	95	100

Table 6.1: *Classifier confusion matrix.*

This yields a retained value of 98.6%. However, this can be scaled to reflect expected proportions as shown in table 6.2.

	Predicted True	Predicted False	Total
Actual True	19.8	0.02	20
Actual False	4	76	80

Table 6.2: *Class balanced confusion matrix.*

The value retained then decreases to 98.36% since the error in the actual false samples has more weight. However, the difference is very small and the value retention rate is still very good.

6.3.2 Regressor

The selection of a subset of architectures for the regression task was somewhat challenging. InceptionResNetV2 performed poorly, diverged significantly on several of the folds and was excluded from the plots due to scaling issues. The VGG networks also had the tendency to diverge, however it appeared to be an issue with stable initialisation. In some cases, several attempts had to be made before they converged on any given fold.

The DenseNet architectures were clearly the best, but it was decided to include other types of architectures for the sake of comparison. Xception was included as it significantly outperformed the rest and MobileNet was included as a model optimised for low computational and memory requirements. NASNetMobile and MobileNet are both optimised for mobile platforms, and NASNetMobile and ResNet50 had similar score. Therefore, of the three, MobileNet and ResNet50 were selected for further optimisation since NASNetMobile has less interesting insight to offer.

The performance of the different configurations of the top layer architecture yielded similar results with the ReLU activation function. The variation within the results increased with the number of units implying some degree of overfitting. The 64 unit configuration was chosen since it had the lowest mean RMSE and because it was thought that adding dropout would limit any overfitting. However, dropout was then found to have a detrimental effect on the model and was not included. This could mean that the capacity of the top layer architecture could not incorporate any redundant pathways. Increasing the capacity by adding more dense units or decreasing the proportion of dropout used may have led to a more robust model.

When determining the ideal training time, a learning rate plateau callback was used. This decreased the learning rate periodically throughout the training process, an effect which can be seen in figure 5.12 on page 66. The drops in learning rate were consistently timed across all tested folds. Only after collecting this data, it was realised that this callback cannot be used during training without validation as was done to train the final model. This is because the callback monitors the validation metrics of which there are none. To compensate for this oversight, the model was trained in intervals between which the learning rate was decreased. This allowed for a replication of the same training conditions for the final model which appears to have been successful since the results coincide with those attained through validation.

The RMSE of the final model was affected, in part, by the prediction of small values of RBR in the images with no RBR. The errors were small, normally $<5\%$, but there were as many as there were images with no RBR. The accumulated effect caused a small but significant error, the effect of which is investigated in the next section.

6.4 Combined model

Using both models resulted in a decrease in the RMSE of percentage RBR predicted as show in figure 6.8 on the following page. This was expected, as some of the error was due to small values predicted for images with no RBR present.

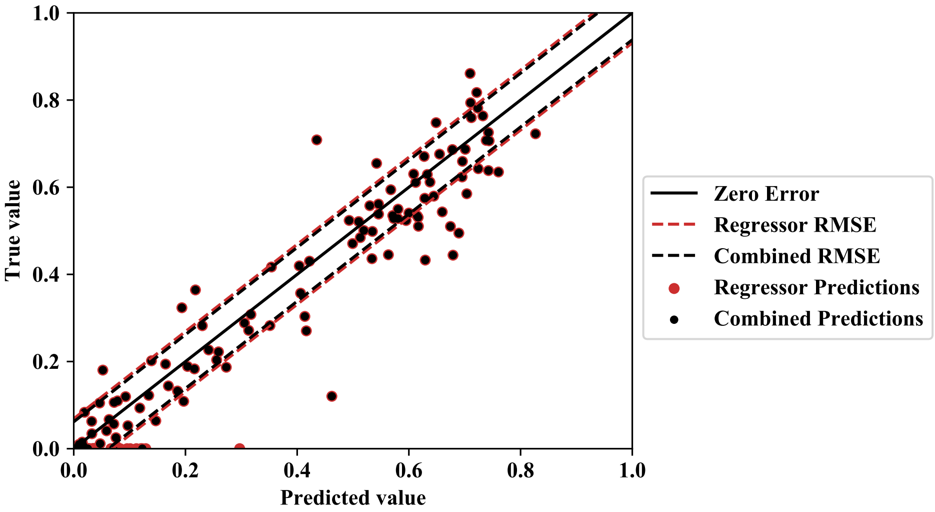


Figure 6.8: *Combined model regression comparison. By removing samples with a true value of zero, those on the x axis, the RMSE decreases.*

The DenseNet201 architecture was selected for both the classifier and the regressor. This would indicate that the features used for both tasks are very similar, which is not surprising since they are both looking for features common to RBR. This implies that a combining the model architectures is feasible. With two output layers, with linear and sigmoid activation functions, the model could be trained to classify and quantify RBR in an image. While this may halve the computational load of sample analysis, it may make optimisation of the model difficult since the required feature space may not be identical.

6.5 Implications for forest operations

The value of this system is two fold: giving real time feedback to the harvester operator and establishing a tree-level database for RBR distribution.

6.5.1 Real time feedback

The immediate value would be evident to the operator of the harvester by providing real time feedback. Harvesters are complex machines to operate, requiring many years of experience to reach a high level of competency and intense concentration for many hours at a time. The operator is expected to detect RBR in the log with a high degree of accuracy. An image of the log butt may suffice for the operator to make their decision but evaluating it would still require time and effort. An accurate, automated system could be used to inform the operators decisions for increased gains, both economic and ergonomic.

6.5.2 RBR database

Attempts have already been made to detect RBR with machine learning with the aim of building a database. Most recently, in the PRECISION project, using UAV based imaging [59] and a crane mounted camera [47]. For detection of RBR in a segmented image of the stump, these studies achieved F1 scores of 0.7926 (0.7665 precision and 0.8205 recall) and 0.9082 (0.9082 precision and 0.9082 recall), respectively. However, both of these studies included the challenging task of identifying the stump before RBR detection, reducing their overall accuracy, a source of error which is not relevant to the system proposed by this study. Linking the classification of the stump with the associated StanForD file with a reasonable degree of certainty also proved difficult due to the low accuracy GPS information in the file. Again, not an issue that would need to be addressed by the proposed system.

6.6 Further work

The largest source of uncertainty in the finding of this study was assumption that the image dataset used was an accurate representation of the perspective from a harvester head mounted camera. A continuation of this work would need to validate the model on images acquired from the correct perspective of log butt clear of occlusions that would be unlikely to happen in the real system. Additionally, integration of a model predicting the vertical RBR propagation from the quantity and distribution detected would have the most immediate benefits to those who are likely to adopt a system like this. There are also many other properties of the log which may have value to integrate into this system.

The methodology could also be improve with the development and implementation of an image data generator with integrated cross-validation. This would reduce the computational workload of the model selection process significantly. The scope of the proposed solution could also be expanded through development of a mobile application for data collection. The MobileNet architecture had promising results in the selection process and may be possible to optimise to a reasonable degree of accuracy.

Chapter 7

Conclusions

Overall, this study showed that a computer vision system using a CNN-based model may provide accurate information on the presence and quantity of RBR in real time to the harvester operator, and to inform the decisions of the forest industry and research community. By exploiting the availability of pretrained CNNs, the methodology employed in this study resulted in a highly accurate model with relatively little fine tuning required. The uncertainty in the labelling of the misclassified samples appears to indicate that the model is at least on par with the average human at classifying RBR. Therefore, it is the conclusion of this study that this system has shown sufficient potential for further development, the next step of which would be validation on a image dataset from a harvester head mounted camera.

Bibliography

[1] FAO, *Global Forest Resources Assessment 2015*. UN Food and Agriculture Organisation, 2015.

[2] ——, *Global Forest Resources Assessment 2010*. UN Food and Agriculture Organisation, 2010.

[3] "Forestry in the EU and the world," https://ec.europa.eu/eurostat/, accessed: 2019-02-28.

[4] J. Arlinger, M. Nordström, and J. Möller, "Stanford 2010," *Modern communication with forest machines, Working report*, vol. 785, p. 83, 2012.

[5] F. O. Asiegbu, A. Adomas, and J. Stenlid, "Conifer root and butt rot caused by Heterobasidion annosum (Fr.) Bref. sl," *Molecular plant pathology*, vol. 6, no. 4, pp. 395–409, 2005.

[6] S. Woodward, *Heterobasidion annosum: Biology, ecology, impact, and control*. CABI, 1998.

[7] M. Garbelotto and P. Gonthier, "Biology, epidemiology, and control of Heterobasidion species worldwide," *Annual review of phytopathology*, vol. 51, pp. 39–59, 2013.

[8] M. Holopainen, M. Vastaranta, and J. Hyyppä, "Outlook for the next generation's precision forestry in finland," *Forests*, vol. 5, no. 7, pp. 1682–1694, 2014.

[9] S. Al Hagrey, "Electrical resistivity imaging of tree trunks," *Near Surface Geophysics*, vol. 4, no. 3, pp. 179–187, 2006.

[10] B. Greig and J. Pratt, "Some observations on the longevity of Fomes annosus in conifer stumps." *European journal of forest pathology*, vol. 6, no. 4, pp. 250–253, 1976.

[11] M. Thor, J. D. Arlinger, and J. Stenlid, "Heterobasidion annosum root rot in Picea abies: Modelling economic outcomes of stump treatment in Scandi-

navian coniferous forests," *Scandinavian journal of forest research*, vol. 21, no. 5, pp. 414–423, 2006.

[12] N. Arhipova, "Heart rot of spruce and alder in forests of latvia," Ph.D. dissertation, Swedish University of Agricultural Sciences, 2012.

[13] Q. V. Cao, H. E. Burkhart, and T. A. Max, "Evaluation of two methods for cubic-volume prediction of loblolly pine to any merchantable limit," *Forest Science*, vol. 26, no. 1, pp. 71–80, 1980.

[14] O. M. Bollandsås, M. Maltamo, T. Gobakken, V. Lien, and E. Næssset, "Prediction of timber quality parameters of forest stands by means of small footprint airborne laser scanner data," *International journal of forest engineering*, vol. 22, no. 1, pp. 14–23, 2011.

[15] H. Solheim, K. J. Huse, and K. Venn, "A nation-wide inventory of root and butt rot in the annual cuttings of spruce in Norway," in *8. International Conference on Root and Butt Rots, Uppsala (Sweden), 9-16 Aug 1993*. Sveriges Lantbruksuniv., 1994.

[16] M. Hauglin, E. H. Hansen, E. Næsset, B. E. Busterud, J. G. O. Gjevestad, and T. Gobakken, "Accurate single-tree positions from a harvester: a test of two global satellite-based positioning systems," *Scandinavian journal of forest research*, vol. 32, no. 8, pp. 774–781, 2017.

[17] R. Sturrock, S. Frankel, A. Brown, P. Hennon, J. Kliejunas, K. Lewis, J. Worrall, and A. Woods, "Climate change and forest diseases," *Plant Pathology*, vol. 60, no. 1, pp. 133–149, 2011.

[18] N. La Porta, P. Capretti, I. M. Thomsen, R. Kasanen, A. M. Hietala, and K. Von Weissenberg, "Forest pathogens with higher damage potential due to climate change in Europe," *Canadian Journal of Plant Pathology*, vol. 30, no. 2, pp. 177–195, 2008.

[19] B. Talbot, M. Pierzchała, and R. Astrup, "Applications of remote and proximal sensing for improved precision in forest operations," *Croatian Journal of Forest Engineering: Journal for Theory and Application of Forestry Engineering*, vol. 38, no. 2, pp. 327–336, 2017.

[20] M. Miettinen, J. Kulovesi, J. Kalmari, and A. Visala, "New measurement concept for forest harvester head," in *Field and service robotics*. Springer, 2010, pp. 35–44.

[21] A. F. Atiya, "An unsupervised learning technique for artificial neural networks," *Neural Networks*, vol. 3, no. 6, pp. 707–711, 1990.

[22] D. Koller, N. Friedman, S. Džeroski, C. Sutton, A. McCallum, A. Pfeffer, P. Abbeel, M.-F. Wong, D. Heckerman, C. Meek *et al.*, *Introduction to statistical relational learning*. MIT press, 2007.

[23] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[24] F. Rosenblantt, “The perceptron, a perceiving and recognizing automation,” *Cornell Aeronautical Laboratory*, 1957.

[25] B. Widrow, “An adaptive ’ADALINE’ neuron using chemical ’memistors’, 1553-1552,” 1960.

[26] V. Nair and G. E. Hinton, “Rectified linear units improve restricted Boltzmann machines,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[27] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[28] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” *arXiv preprint arXiv:1511.07289*, 2015.

[29] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, “Handwritten digit recognition with a back-propagation network,” in *Advances in neural information processing systems*, 1990, pp. 396–404.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, pp. 84–90, 2012.

[31] H. Lee, C. Ekanadham, and A. Y. Ng, “Sparse deep belief net model for visual area v2,” in *Advances in neural information processing systems*, 2008, pp. 873–880.

[32] T. Serre, L. Wolf, and T. Poggio, “Object recognition with features inspired by visual cortex,” Massachusetts Institute of Technology Cambridge Department of Brain and Cognitive Sciences, Tech. Rep., 2006.

[33] M. Lin, Q. Chen, and S. Yan, “Network in network,” *arXiv preprint arXiv:1312.4400*, 2013.

[34] B. Karlik and A. V. Olgac, “Performance analysis of various activation functions in generalized MLP architectures of neural networks,” *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[35] H. J. Kelley, “Gradient theory of optimal flight paths,” *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[36] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[38] C. Darken, J. Chang, and J. Moody, "Learning rate schedules for faster stochastic gradient search," in *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*. IEEE, 1992, pp. 3–12.

[39] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[40] T. Dozat, "Incorporating nesterov momentum into adam.(2016)," *Dostupné z: http://cs229. stanford. edu/proj2015/054_report. pdf*, 2016.

[41] K. Jarrett, K. Kavukcuoglu, Y. LeCun *et al.*, "What is the best multi-stage architecture for object recognition?" in *2009 IEEE 12th International Conference on Computer Vision (ICCV)*. IEEE, 2009, pp. 2146–2153.

[42] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[43] J. Dvořák, R. Bystrickỳ, P. Hošková, M. Hrib, M. Jarkovská, J. Kováč, J. Krilek, P. Natov, and L. Natovová, *The use of harvester technology in production forests*. Lesnická Práce, 2011.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[45] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[46] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[47] A. Ostovar, B. Talbot, S. Puliti, R. Astrup, and O. Ringdahl, "Detection and classification of Root and Butt-Rot (RBR) in stumps of Norway Spruce using RGB images and machine learning," *Sensors*, vol. 19, no. 7, p. 1579, 2019.

[48] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[49] L. Fei-Fei, "Imagenet: crowdsourcing, benchmarking & other cool things," in *CMU VASC Seminar*, vol. 16, 2010, pp. 18–25.

[50] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[51] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[53] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[54] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[55] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[56] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[57] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[58] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[59] S. Puliti, B. Talbot, and R. Astrup, "Tree-stump detection, segmentation, classification, and measurement using unmanned aerial vehicle (UAV) imagery," *Forests*, vol. 9, no. 3, p. 102, 2018.

Appendix

Appendix A: Classifier Errors

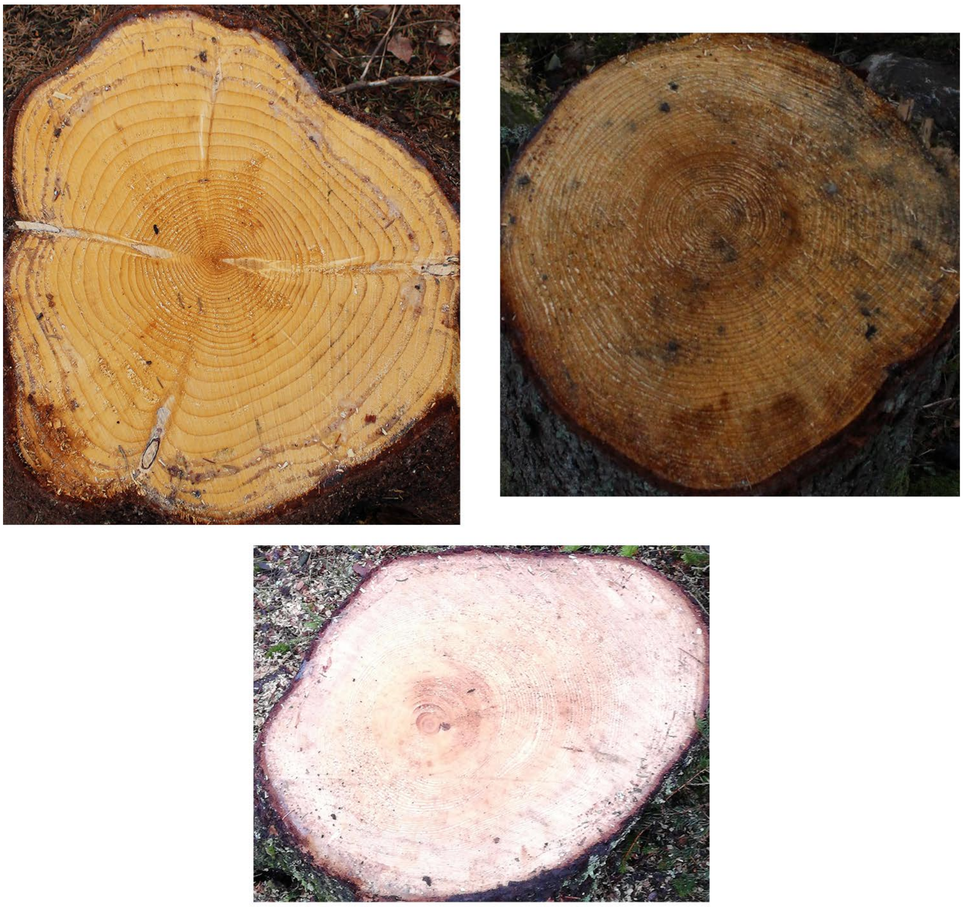


Figure 7.1: *False positive classifications. All the images have areas of discolouration that could be RBR.*

Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway