UNSW
SYDNEY

Deep Learning Methods for Lacune Detection in MRI

Melinda Mortimer

Supervisors: Dr Pierre Lafaye de Micheaux & A/Prof Wei Wen

School of Mathematics and Statistics

UNSW Sydney

November 2018

Submitted in partial fulfillment of the requirements of the degree of

Bachelor of Science with Honours

Plagiarism statement

I declare that this thesis is my own work, except where acknowledged, and has not been submitted for academic credit elsewhere.

I acknowledge that the assessor of this thesis may, for the purpose of assessing it:

- Reproduce it and provide a copy to another member of the University; and/or,
- Communicate a copy of it to a plagiarism checking service (which may then retain a copy of it on its database for the purpose of future plagiarism checking).

I certify that I have read and understood the University Rules in respect of Student Academic Misconduct, and am aware of any potential plagiarism penalties which may apply.

By signing this declaration I am agreeing to the statements and conditions above.

Signed: _____ Date: _____

Acknowledgements

I would like to thank my supervisors, Pierre and Wei, for their guidance and support throughout this year.

Thanks to Jiyang for all your assistance and helping with data preparation.

I would also like to thank Ramanan, Jeffrey, and Prosha for giving me helpful advice and endless proof-reading.

Thanks also to my parents and Kevin for looking after me over the past year.

Melinda Mortimer, 26 October 2018.

Abstract

Cerebral small vessel disease is a neurological disease that affects over 90% of elderly and is a major cause of stroke, dementia, and cognitive decline. Biomarkers of the disease are visible in MRI and are manually identified by trained clinicians. Lacunes are one such biomarker and are of particular interest for their potential role in the advancement of cerebral small vessel disease and in the incidence of stroke. However, the manual identification of lacunes is difficult, laborious, and largely inconsistent.

In this thesis, we present a convolutional neural networks model for the automated detection of lacunes in MRI. The model adapts an existing model by Ghafoorian et al. (2017) to the scenario in which location-based variables are unavailable. In particular, we remove the dependence on location-based variables, perform extraction of the brain matter prior to model input, and remove early dimension reduction. The resulting model exhibited a testing sensitivity of 99.9% and a specificity of 99.8%.

It should be noted that this model performance was a result of sampling correlations between the training, testing, and validation sets. A discussion of the final results and the corrected model is provided.

Notation

$\mathbf{a}^\ell$	Vector of activation values (outputs) of the ℓ-th layer.
a_j^ℓ	j-th element of $\mathbf{a}^\ell$, the activation value of the j-th neuron of the ℓ-th layer.
$\mathbf{w}^\ell$	Matrix of weights used to compute the vector $\mathbf{a}^\ell$.
$\mathbf{w}_j^\ell$	Vector of weights used to compute a_j^ℓ.
w_{ji}^ℓ	i-th element of $\mathbf{w}_j^\ell$, to be multiplied with $a_i^{\ell-1}$.
$\mathbf{b}^\ell$	Vector of biases of the ℓ-th layer.
b_j^ℓ	Bias of the j-th neuron of the ℓ-th layer.
z_j^ℓ	Value of the j-th neuron of the ℓ-th layer before the activation function $\sigma(\cdot)$ is applied. $\mathbf{w}_j^\ell \cdot \mathbf{a}^{\ell-1} + b_j^\ell = \sum_k w_{jk}^\ell a_k^{\ell-1} + b_j^\ell$.
$\mathbf{z}^\ell$	Vector of z_j^ℓ of the ℓ-th layer.
δ_j^ℓ	Gradient error in the j-th neuron of the ℓ-th layer when minimising the cost function C. $\dfrac{\partial C}{\partial z_j^\ell}$.
δ^ℓ	Vector of errors δ_j^ℓ for the ℓ-th layer.
Z	Set of all z_j^ℓ in the network.
W	Set of all weights w_{jk}^ℓ in the network.
B	Set of all biases b_j^ℓ in the network.
$\mathbf{v}$	Set of all weights and biases. (W, B).
L	Number of layers in the network.
X	Set of n training inputs. $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$.
Y	Set of n training responses. $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$.
η	Learning rate, controls the magnitude of adjustments made to weights during training.

Contents

Chapter 1

Introduction

Cerebral Small Vessel Disease (SVD) describes a set of abnormalities affecting small blood vessels in the deep grey and white matter of the brain. The changes are particularly prevalent amongst the elderly, with SVD biomarkers appearing in over 90% of MRI for those aged 60–90 years (de Leeuw et al., 2001). SVD is the primary cause of a quarter of ischaemic (oxygen-starved) strokes (Wardlaw et al., 2013b) and is a major cause of dementia and cognitive decline (Norrving, 2008).

Imaging markers of SVD include lacunes, enlarged perivascular spaces, white matter hyperintensities, microbleeds, recent small subcortical infarcts, and brain atrophy (Wardlaw et al., 2013b). Current research investigates the role of biomarkers in the advancement of SVD as the extent to which the disease affects cognition, or which particular events are to blame, is not clear. Consequently, clear identification of biomarkers is needed to ensure valid diagnosis and further research.

The identification of SVD biomarkers in MRI (known as image *rating*) is generally conducted by eye. The three-dimensional image constructed by an MRI scan is made up of numerous two-dimensional *slices*. Trained clinicians, with reference to the STRIVE criterion (Wardlaw et al., 2013b), examine each image slice for lesions and other features of interest. The locations and sizes of these features are logged manually and is therefore a very time consuming task. van den Heuvel et al. (2016) claim that the rating of a single brain scan takes approximately one hour. Additionally, neuroscientists have commented on the difficulty and reliability of these rating methods (Benjamin et al., 2018; Ghafoorian et al., 2017; Yokoyama et al., 2007).

Wardlaw et al. (2013a) advise caution when establishing research conclusions as many of the features are difficult to differentiate.

Lacunes are of particular interest as the role of lacunes in SVD and in the incidence of stroke is under review. For instance, Benjamin et al. (2018) argue that it is only lacunes, rather than perivascular spaces, that influence cognitive decline. They suspect that previous observed influence of perivascular spaces may have been the result of incorrect classification during the rating process.

Attempts at improving the accuracy, efficiency, and consistency of image rating has been undertaken through the use of automation. Yokoyama et al. (2007) determined candidate lacunes by considering the candidate's distance to central structures of the brain. False-positives were reduced by developing thresholds on the candidates' area, perimeter, and centre of gravity. The model had a lacune sensitivity of 90.1%, specificity of 30.0%, and had an average of 1.7 false-positives per image slice. Uchiyama et al. (2007a,b) used the method developed by Yokoyama et al. (2007) and further reduced false-positives by feeding twelve additional location and visual variables into two machine learning models: a support vector machine and a neural network. The resulting models had a sensitivity of 96.8%, and had an average of 0.76 false-positives per slice. These models were revised by (Uchiyama et al., 2015), with false-positives reduced by matching candidate lacunes with image templates. The resulting model had a sensitivity of 96.8% with 0.47 false-positives per slice. The sensitivities of these models are moderately high and demonstrate efficacy at identifying lacunes, however further improvements can be made to ensure that the models can run without clinician supervision.

Improvements in image classification have been made with the introduction of convolutional neural networks (CNNs). The AlexNet CNN model (Krizhevsky et al., 2012) performed most accurately of all model entries on the ImageNet data set, which contains 1.5×10^7 images to be classified from 2.2×10^5 categories. A CNN was successfully applied to the MRI context by Dou et al. (2016), resulting in accurate detection of cerebral microbleeds.

A successful CNN model for lacune detection was first developed by Ghafoorian et al. (2017). This model involved a two-dimensional CNN for candidate lacune detection, followed by three parallel three-dimensional CNNs connected with seven additional location-based variables, to reduce the number of false-positive classifications. This model exhibited a sensitivity of 97.4% and exhibited an average of 0.13 false-positives per slice. Although the model demonstrates highly accurate performance, it relies on the inclusion of location-based variables. Some of these variables include distances between the candidate points and particular structures of the brain. However, the precise locations and sizes of these brain structures are not constant between scans. These location-based variables must be estimated using existing algorithms or, in the case that a clinician does not have access to these algorithms, measured manually.

In this thesis, we adapt the CNN developed by Ghafoorian et al. (2017) to the scenario in which the location-based variables are unavailable. In particular, we make three major adjustments. The first is the *removal of the location-based variables* presumed to be inaccessible. The second adjustment is the *extraction of brain matter* to remove visual identifiers such as the skull and eyes. The third adjustment is the *removal of early dimension reduction* to ensure that adequate pixel information is being passed to the first layer of the CNN.

The resulting model exhibits a testing sensitivity of 99.9% and specificity of 99.8%. This improved sensitivity indicates that the model is as sensitive to lacunes as a trained clinician. Improving model specificity minimises the time spent checking the model results. A model with a very large number of false-positives may take just as long to correct as the rating of a whole scan. Improvement in specificity may be achieved by applying a three-dimensional CNN to improve image context, or by the inclusion of simple location-based variables, such as the x, y, and z-coordinates. It should be noted that these methods require additional data preprocessing before use in a data pipeline.

We now outline the structure of the remainder of this thesis.

In Chapter 2, we introduce terminology surrounding MRI and the structure of the brain. We formally define the biomarkers involved in SVD and how they are identified. We discuss current image rating methods and the need for automated image rating algorithms.

In Chapter 3, we introduce the structure of neural networks and discuss the choice of activation and cost functions. We show that the minimum of the cost function can be estimated using the Gradient Descent algorithm, and finally discuss methods that mitigate overfitting of the data.

In Chapter 4, we introduce convolutional neural networks for image classification tasks. We outline techniques that improve training efficiency given a large number of variables to be estimated.

In Chapter 5, we discuss the efficacy of existing models, outline the structure of the lacune identification model built by Ghafoorian et al. (2017), and outline the proposed changes to the model.

In Chapter 6, we describe the methods of data collection and the preprocessing applied to the collected scans. We also describe the methods of data sampling before model implementation.

In Chapter 7, we present the results of two CNNs. We test the significance of a greater proportion of negative samples during training, and determine the final testing sensitivity and specificity rates. We discuss the final behaviour of the model, including the characteristics of samples that incur model error.

In Chapter 8, we discuss existing sample correlations between the training, validation, and testing data sets. We outline the results of the revised models, discuss updated model improvements, and review the existing model false-positive rate.

Chapter 2

Neuroimaging background

An overview of Magnetic Resonance Imaging (MRI) and cerebral small vessel disease (SVD) is required to understand the motivations behind the model, the features that are being detected, and the structure of the data provided. This chapter introduces terminology surrounding the general structure of the brain, features of MRI image types, the definition and appearance of SVD biomarkers, and outlines existing rating methods.

2.1 Radiological axes

By convention, the MRI axis planes are referred to as coronal, sagittal and axial. These are shown in Figure 2.1.

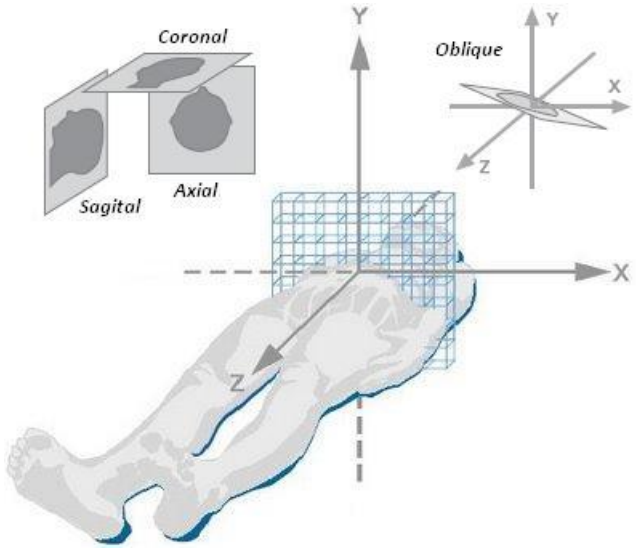

Figure 2.1: MRI axis planes [sic]: coronal, sagittal and axial.
Image taken from Bean (2014).

The resulting image from an MRI scan is in three-dimensions. The scan is therefore referred to as a *volume* and is represented as a three-dimensional array of *voxels*. A *slice* is a cross-sectional image taken from a volume. Hence an *axial slice* is a cross-sectional image with a fixed z-coordinate such that the image appears as if viewed from above the body.

Directional terminology in MRI include:

- *anterior* and *posterior*, referring to objects situated towards the front and back of the body respectively;
- *superior* and *inferior*, referring to objects situated above and below other parts of the body respectively; and
- *interior* and *exterior*, referring to objects situated closer to and further from the $x = 0$ sagittal plane respectively.

2.2 Structure of the brain

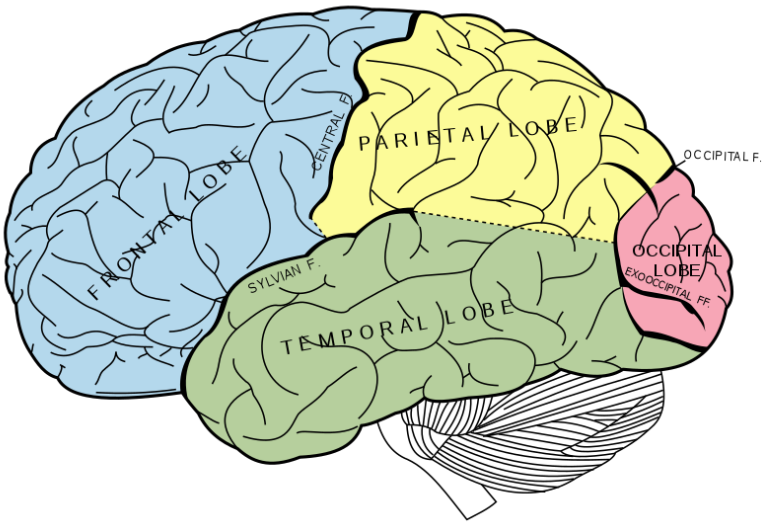

Figure 2.2: The four lobes of the cerebral cortex.
Image taken from Wikimedia Commons: `'Gray728.svg'`.

Information from throughout the body is communicated via nerves through the spinal cord to the brain. The brain is the most complex organ in the body, tasked with receiving, interpreting, and responding to nerve signals. It can be segmented into a number of regions, each responsible for different roles.

The largest region is the *cerebrum*, shown in Figure 2.2, which forms the outer surface of the brain. It is responsible for voluntary actions, senses, thought and memory, and is divided into the left and right hemispheres. Each hemisphere is divided into four lobes:

- The *frontal lobe*, located at the front of the cerebrum, is responsible for voluntary movement, skills and behaviours, mood, and memory.
- The *parietal lobe*, situated posterior to the frontal lobe, is responsible for the senses, including pain, and physical and spatial awareness.
- The *temporal lobe*, located exterior to the parietal lobe, is responsible for memory and auditory functions, including hearing and speech.
- The *occipital lobe*, located posterior to the parietal lobe, is responsible for visual information.

The outer surface of the brain consists of a layer of neurons referred to as *grey matter*. It is here that much of the brain processes occur (Dafny, 1997). Underneath the grey matter is a network of fibres that connects the grey matter neurons together. Collectively they form the *white matter*. The grey and white matter are shown in Figure 2.3.

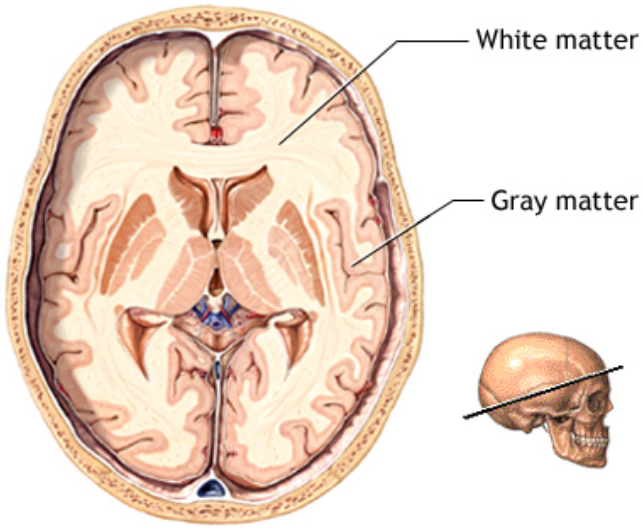

Figure 2.3: Grey matter occurs at the surface and within central structures such as the spinal cord. The white matter connects these structures together.
Image taken from '`https://medlineplus.gov/ency/imagepages/18117.htm`'.

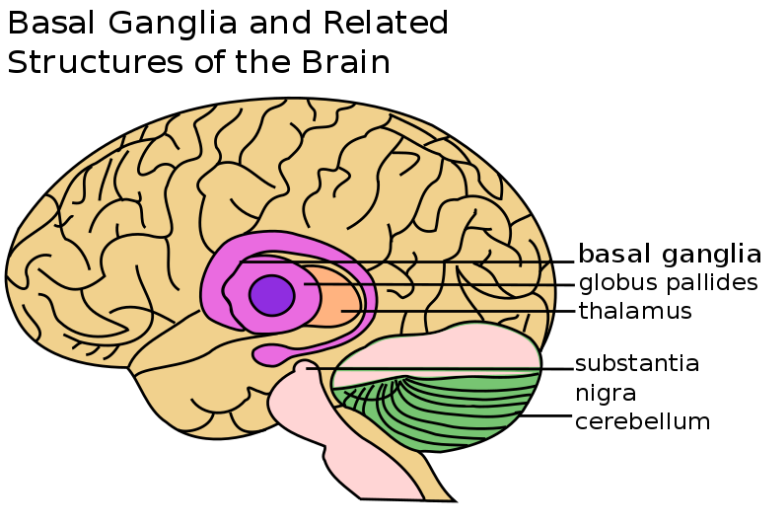

Figure 2.4: The basal ganglia and other related structures.
Image taken from Wikimedia Commons:
'Basal_Ganglia_and_Related_Structures.svg'.

At the centre of the brain are structures that form the *basal ganglia*, shown in Figure 2.4. This region of the brain is responsible for voluntary movement and learning. Connected to this structure is the *thalamus*, which is related to sensory and motor function.

At the base of the brain lies the *cerebellum*, responsible for coordination; and the *brain stem*, responsible for the transmission of nerve communications. Within the skull, the brain sits in the brain cavity filled with *cerebral spinal fluid* (CSF). This fluid can flow between the ridges at the brain's surface, filling gaps throughout the brain matter. CSF is produced in cavities at the centre of the brain known as the *cerebral ventricles*.

2.3 Magnetic Resonance Imaging

MRI is a radiological technique that uses magnetic fields and radio waves to generate greyscale images of organs inside the body (Rinck, 2013). Three imaging types produced are the *T1-weighted*, *T2-weighted* and *FLuid-Attenuated Inversion Recovery* (FLAIR) images, shown in Figure 2.5.

T1-weighted images are *hyperintense* (bright) in regions with high fat content and are *hypointense* (dark) in regions with high water content (Bitar et al., 2006).

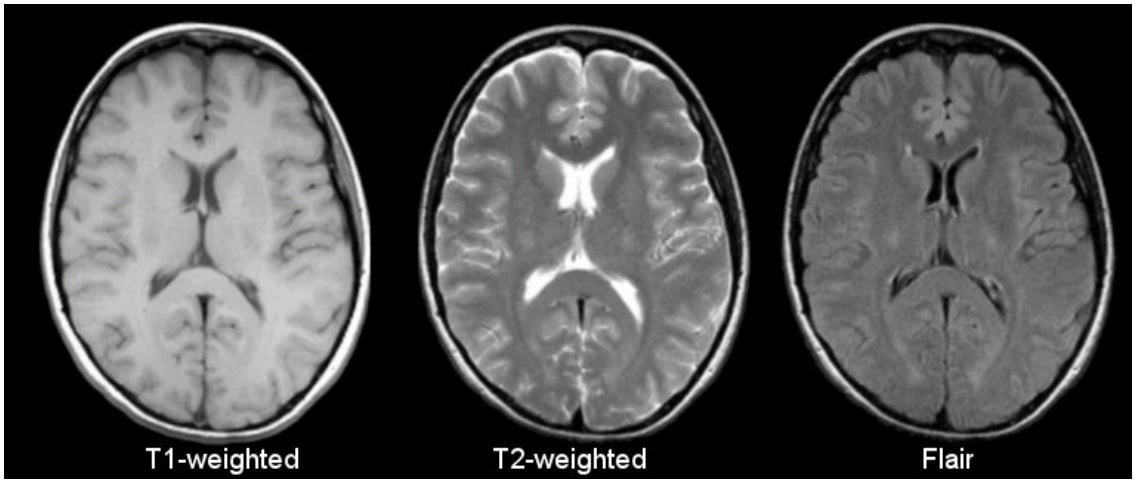

Figure 2.5: A comparison of T1, T2 and FLAIR images.
Image taken from Preston (2006).

They therefore return high intensities for brain matter and low intensities for CSF. T2-weighted images are hyperintense in regions that contain both high fat and water content, aiding in the detection of abnormalities (Bitar et al., 2006). FLAIR is an imaging sequence similar to T2-weighted imaging, excepting that CSF remains hypointense. Abnormalities will appear bright amongst the darker CSF, allowing for easier identification.

2.4 SVD biomarkers

During the analysis of MRI scans for small vessel disease (SVD), there are a number of biomarkers that clinicians observe. Each of these is defined in conjunction with the STRIVE criterion (Wardlaw et al., 2013b) shown in Figure 2.6. Schematics show a simplified representation of each biomarker for specified imaging types. We define a number of these biomarkers now. It should be noted that diffusion-weighted imaging (DWI), recent small subcortical infarcts, and brain atrophy will not be discussed in this thesis.

White matter hyperintensities (WMH) are regions of hyperintensity visible in FLAIR and T2-weighted imaging. They also appear in T1-weighted images as hypointense regions marginally brighter than CSF. Their cause is not well understood (Gouw et al., 2011).

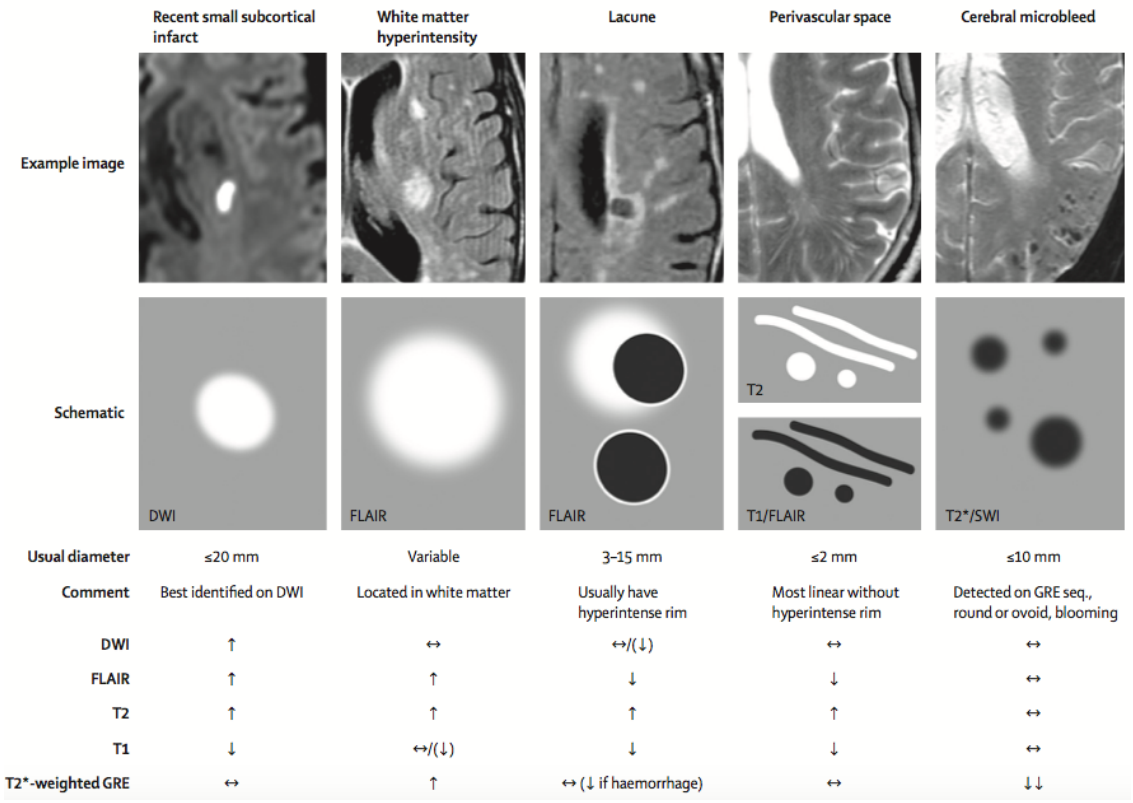

Figure 2.6: STRIVE criterion and MRI examples. Up and down arrows indicate expected hyperintensity and hypointensity respectively for each biomarker. Horizontal arrows indicate structures of similar intensity to the surrounding brain matter.
Image taken from Wardlaw et al. (2013b).

Lacunes are small brain cavities that usually appear without symptoms and are frequently found in the scans of elderly. Their presence indicates a heightened risk of stroke and dementia (Benjamin et al., 2018; van der Flier et al., 2005). In MRI, lacunes appear round, with a diameter of 3–15 mm. They tend to exhibit a darker signal intensity, similar to that of CSF, as they are filled with fluid. Lacunes have a tendency to occur in regions of white matter hyperintensity, so they will frequently have a hyperintense rim in FLAIR imaging.

Perivascular spaces are extensions of the fluid space surrounding blood vessels through the brain. They are generally microscopic but can become enlarged with age, and often appear alongside other SVD biomarkers such as lacunes and WMH. Perivascular spaces also exhibit a signal intensity similar to CSF as they are fluid-filled. They are found running parallel to vessels, and are generally found under 3 mm in diameter. They can be identified by appearing circular cross-sectionally

and rectangular when viewed in parallel to the vessels. In some instances, perivascular spaces can become enlarged, up to 10 mm in diameter. Under these circumstances, perivascular spaces can be difficult to distinguish from lacunes as their signal intensities are similar.

Cerebral microbleeds are blooming regions of microscopic bleeding in the brain, generally 2–5 mm in diameter. They are not visible on T1-weighted or FLAIR images, and are instead found in adjusted T2-weighted images.

2.5 Image rating methods

Without a biopsy for confirmation, the identification of SVD biomarkers relies on MRI analysis. Trained observers examine MRI volumes slice by slice and identify any lesions or points of interest. The rating of lacunes and enlarged perivascular spaces is particularly difficult as they exhibit similar intensities in T1-weighted and FLAIR scans, making them difficult to distinguish (Potter et al., 2015). Additionally, the manual rating process is time consuming. van den Heuvel et al. (2016) report that the rating of a single scan for microbleeds takes an average of one hour. Data sets with a large number of scans therefore take many hours to process, and the resulting identified biomarkers may not be reliable enough to warrant correct inference (Benjamin et al., 2018; Wardlaw et al., 2013b).

Machine learning algorithms have been developed to improve image rating quality, consistency, and speed. Existing algorithms have been successfully built to work alongside clinicians as computer-aided design (CAD) programs (van den Heuvel et al., 2016; Uchiyama et al., 2007a; Yokoyama et al., 2007), and as fully automated systems (Dou et al., 2016; Ghafoorian et al., 2017). This motivates our discussion on neural network structure and implementation.

Chapter 3

Neural networks

This chapter outlines the structure and workings of basic neural network models. Those who have a basic understanding of neural networks and wish to avoid the details surrounding model structure and minimisation algorithms can proceed to Chapter 4 for an overview of convolutional neural networks and Chapter 5 for existing lacune detection attempts.

Neural networks have become increasingly popular with advances in computing power and the availability of large data sets (Goodfellow et al., 2016). They have been proven successful with MRI discrimination tasks (Dou et al., 2016; Yokoyama et al., 2007) and image classification tasks (He et al., 2015; Krizhevsky et al., 2012; Szegedy et al., 2015). In some instances, neural networks have exhibited a higher image recognition accuracy than humans (He et al., 2015).

The construction of neural networks has to be conducted with care. The resulting models are difficult to interpret and prone to overfitting. We now discuss the underlying structure of neural networks, cost minimisation, and techniques to avoid overfitting the data.

3.1 Basic structure

The structure of neural networks is analogous to that of neurons in the brain. Each brain cell receives a signal, conducts a small amount of processing, and passes the resulting signal to the next cell. Decisions made by the brain are the result of many neurons processing information and communicating in parallel. Neural

networks adopt a similar structure. Individual nodes receive values and apply a transformation. The outputs of several nodes are passed as inputs into later nodes, feeding information forward until a final response is calculated. For this reason, the nodes are referred to as *neurons*.

The structure of a neural network neuron is shown in Figure 3.1, and its goal is to quantify a particular feature of the input variables. Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)^\intercal$ be a vector of n input variables. Let $\mathbf{w} = (w_1, \ldots, w_n)^\intercal$ be a vector of weights. Let b be an additional *bias* variable, not to be confused with statistical bias. The bias term is a constant used to leverage the value of the neuron independently of the input variables. Let $\sigma(\cdot)$ be some function, known as an *activation function*, which is generally nonlinear (see Section 3.2). Let $\cdot$ be the usual dot product. The output of a single neuron, known as an *activation value*, is given by

$$a = \sigma(\mathbf{w} \cdot \mathbf{x} + b).$$

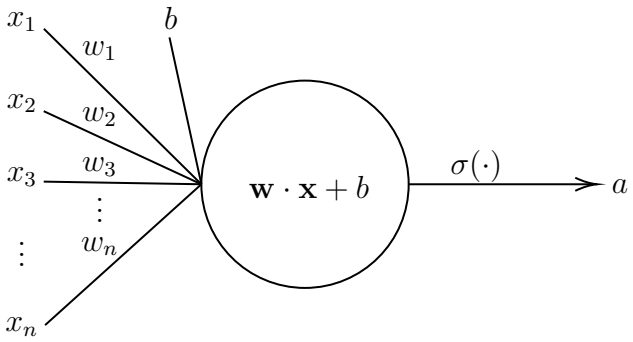

Figure 3.1: Structure of a basic neuron.

As an example, considering the following neuron structure.

Example 1. A neuron can be used to express a logical AND gate of two inputs. Let TRUE be encoded to 1 and FALSE be encoded to 0 so that $x_i \in \{0, 1\}$ for $i = 1, 2$. Let $\mathbf{w} = (1, 1)^\intercal$ and $b = -1.5$. Let the activation function be the step function

$\sigma(z) = 1$ for $z > 0$ and $\sigma(z) = 0$ otherwise. The resulting neuron has output

$$a(\mathbf{x}) = \begin{cases} 1, & x_1 + x_2 - 1.5 > 0 \\ 0, & \text{otherwise} \end{cases},$$

which will output 1 if and only if $\mathbf{x} = (1, 1)^\intercal$, satisfying the definition of an AND gate.

A large number of these neurons can be arranged in layers to form a *fully connected neural network*, as shown in Figure 3.2. The inputs $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ are used to predict the final response y. Let $\mathbf{a}^\ell$ be the vector of activation values of the ℓ-th layer, where $\mathbf{a}^0 := \mathbf{x}$. Let $\mathbf{w}^\ell$ be the matrix of weights used to calculate $\mathbf{a}^\ell$, such that each row entry $\mathbf{w}_j^{\ell\intercal}$ is used to compute the j-th neuron of the ℓ-th layer. Let $\mathbf{b}^\ell$ be the vector of biases of the ℓ-th layer. Hence, the vector of activation values of the ℓ-th layer can be expressed as

$$\mathbf{a}^\ell = \sigma(\mathbf{w}^\ell \mathbf{a}^{\ell-1} + \mathbf{b}^\ell), \tag{3.1.1}$$

where $\sigma(\cdot)$ applies the activation function to each element of the vector. Parameterising Equation (3.1.1) as $g_\ell(\mathbf{a}^{\ell-1}; \mathbf{w}^\ell, \mathbf{b}^\ell)$, the final output of the neural network

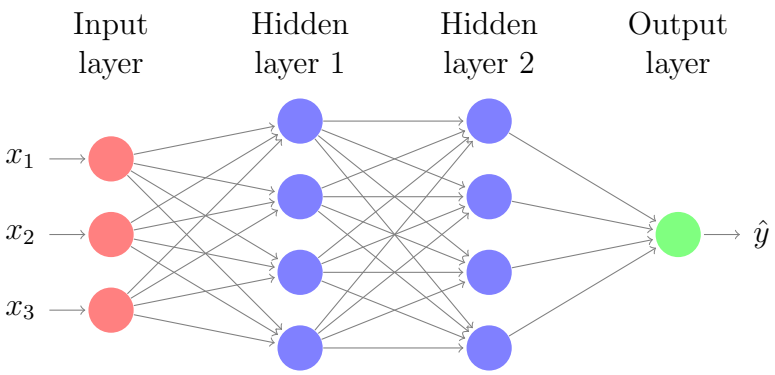

Figure 3.2: Basic neural network structure. Each hidden neuron (blue) and the output neuron (green) has the structure shown in Figure 3.1. For simplicity, the weights and biases are not shown.

can be written as a sequence of function compositions

$$\mathbf{a}^L = g_L \circ g_{L-1} \circ \ldots \circ g_1.$$

During model fitting, the quality of the estimated weights W and biases B is quantified using some *cost function* $C(W, B)$, where a cost of zero describes a perfect model fit. The weights and biases are trained to minimise this cost function with respect to (W, B). However, since the number of variables can be very large, it is not always feasible to analytically minimise $C(W, B)$. Instead, we can approximate this minimisation via the Gradient-Descent algorithm (see Section 3.4). This algorithm is repeated until the cost function falls within some tolerance τ, or reaches the maximum number of iterations n_T.

3.2 Activation functions

Activation functions, denoted by $\sigma(\cdot)$, are applied just before neuron output. Without these functions, the network can be reduced to a linear combination of inputs. Activation functions avoid this by introducing nonlinearity into the model.

The simplest neuron type is the *perceptron*, used in Example 1. It is characterised by,

$$x_i \in \{0, 1\} \text{ and } \sigma(z) = \begin{cases} 1, & z > 0 \\ 0, & z \leq 0 \end{cases}.$$

To allow for continuous outputs, other common activation functions include the sigmoids, which are monotonically increasing, smooth approximations to the step function. Examples of these include the sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

and hyperbolic tan function

$$\sigma(z) = \tanh(z).$$

Both the sigmoid and hyperbolic tan functions are common activation functions (Goodfellow et al., 2016), however they are computationally expensive and so are not preferred for image recognition tasks, which contain a large number of variables (LeCun et al., 2012; Nielson, 2015).

The Rectified Linear Unit (ReLU) activation (Maas et al., 2013) is given by,

$$\sigma(z) = \max(0, z).$$

The ReLU activation function is suitable for neural network optimisation as the function and its derivative are simplistic (Goodfellow et al., 2016). The neuron is said to be *active* for $z > 0$. In this region, the gradient is constant and large for positive input values, promoting training of the neuron's weights (see Section 3.4 on Gradient Descent). For $z \leq 0$, the neuron has a gradient of 0 and is deactivated, avoiding unnecessary parameter adjustments and computation.

The softmax activation function,

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_k e^{z_k}},$$

is used to structure a layer as a discrete probability distribution over k classes. Hence, it is frequently used in the output layer of classification tasks.

3.3 Cost functions

Weights and biases are chosen such that they approximately minimise some cost function. Nielson (2015) describes two restrictions on cost functions for them to be minimised using the Gradient Descent algorithm (see Section 3.4). The first

requirement is that the cost $C_X(\cdot)$ accrued from all samples X equals the mean of costs accrued from n distinct subsamples of X, denoted by X_i,

$$C_X(W, B) = \frac{1}{n} \sum_{i=1}^{n} C_{X_i}(W, B).$$

The second requirement is that the cost is independent of $\mathbf{a}^\ell$ for all $\ell < L$, noting that there remains a dependence on the final activation values. The most common cost function for classification tasks is *cross-entropy* (Nielson, 2015), given by

$$C(W, B) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n_L} \left[y_{ij} \log \left(a_j^L(\mathbf{x}_i, W, B) \right) + (1 - y_{ij}) \log \left((1 - a_j^L(\mathbf{x}_i, W, B)) \right) \right], \tag{3.3.1}$$

where n is the number of samples and n_L is the number of neurons in the output layer.

We now show the efficacy of cross-entropy as a measure of misclassification cost by first determining the Maximum Likelihood Estimator (MLE) of the weights (W, B), and secondly by considering distributional differences between the data and the proposed model.

Let $p_{data}(\mathbf{x})$ be the unknown probability distribution of the data points $X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$. Let $p_{model}(\mathbf{x}; \theta)$ be a family of probability distributions estimating the true $p_{data}(\mathbf{x})$. The MLE of θ is defined as

$$\hat{\theta}_{MLE} = \arg\max_{\theta} \prod_{i=1}^{n} p_{model}(\mathbf{x}_i; \theta).$$

This can be reformulated using the log-likelihood,

$$\hat{\theta}_{MLE} = \arg\max_{\theta} \sum_{i=1}^{n} \log p_{model}(\mathbf{x}_i; \theta).$$

Dividing by n, the expression can be written in terms of expectation,

$$\hat{\theta}_{MLE} = \arg\max_{\theta} \mathbb{E}_{\mathbf{x}\sim\hat{p}_{data}} \left[\log p_{model}(\mathbf{x};\theta)\right]. \tag{3.3.2}$$

This expression is equivalent to the minimisation of cross-entropy. The same minimisation occurs when considering the KL-divergence between p_{model} and p_{data}.

Definition 3.3.1. Kullback-Leibler (KL)-divergence is a measure of difference between two probability distributions $P(x)$ and $Q(x)$ given by

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)],$$

where $\cdot||\cdot$ denotes necessary ordering of the function inputs. Note that KL-divergence is not symmetric and is therefore not a metric.

Minimisation of the KL-divergence between $\hat{p}_{data}$ and p_{model} is given by

$$\hat{\theta}_{KL} = \arg\min_{\theta} \mathbb{E}_{\mathbf{x}\sim\hat{p}_{data}} \left[\log \hat{p}_{data}(\mathbf{x}) - \log p_{model}(\mathbf{x};\theta)\right]. \tag{3.3.3}$$

Since $\hat{p}_{data}$ is not a function of θ, Equation (3.3.3) is equivalent to

$$\hat{\theta}_{KL} = \arg\min_{\theta} -\mathbb{E}_{\mathbf{x}\sim\hat{p}_{data}} \left[\log p_{model}(\mathbf{x};\theta)\right]. \tag{3.3.4}$$

Both the MLE of θ (Equation (3.3.2)) and the minimiser of KL-divergence (Equation (3.3.4)) are equivalent to the minimisation of cross-entropy (Equation (3.3.1)). This minimisation is generally conducted through the Gradient Descent algorithm.

3.4 The Gradient Descent algorithm

Weights W and biases B are chosen such that they approximately minimise the cost function $C(W, B)$. Analytically minimising the cost function is possible, however the large number of variables makes this process slow, having time complexity $O(n^3)$ (Marquardt, 1963). The weights are instead estimated using the Gradient Descent algorithm, improving computation speed.

Let $\mathbf{v} = (W, B)$ of length N be the vector of all weights and biases in the network. Gradient Descent considers the gradient of the cost with respect to the current weights and biases given by

$$\nabla C(\mathbf{v}) = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}, \cdots, \frac{\partial C}{\partial v_N} \right).$$

The algorithm shifts the weights and biases by a small amount $\Delta\mathbf{v}$ such that the cost decreases. Define $\mathbf{v}' := \mathbf{v} + \Delta\mathbf{v}$ to be the updated value of $\mathbf{v}$. Let $\Delta C(\mathbf{v})$ be the change in the cost function as a result of the shifted $\mathbf{v}'$. An approximation of $\Delta C(\mathbf{v})$ is given by the *Total Differential Approximation*

$$\Delta C(\mathbf{v}) \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 + \cdots + \frac{\partial C}{\partial v_N}\Delta v_N \tag{3.4.1}$$

$$= \nabla C(\mathbf{v}) \cdot \Delta\mathbf{v}. \tag{3.4.2}$$

Note the distinction between the gradients denoted by ∇ and small changes denoted by Δ.

We update $\mathbf{v}'$ inductively such that the cost decreases by an amount proportional to a given *learning rate* η, such that

$$\Delta\mathbf{v} = -\eta\nabla C(\mathbf{v}), \quad \text{where } \eta > 0. \tag{3.4.3}$$

Substituting Equation (3.4.3) into Equation (3.4.2), we can write

$$\Delta C(\mathbf{v}) \approx -\eta\|\nabla C(\mathbf{v})\|^2 \leq 0.$$

Thus if $\mathbf{v}' := \mathbf{v} - \eta\nabla C(\mathbf{v})$, the cost function will decrease. In training a neural network, the size of the shift can be set to $\|\Delta\mathbf{v}\| = \varepsilon$, for some $\varepsilon > 0$. It can be shown that the $\Delta\mathbf{v}$ which gives the greatest decrease in $C(\mathbf{v})$ is a function of ε and ∇C (Nielson, 2015).

Lemma 3.4.1. *Let $\varepsilon > 0$ and suppose the size of the shift is constrained such that $\|\Delta\mathbf{v}\| = \varepsilon$. Then $\nabla C \cdot \Delta\mathbf{v}$ is minimised by $\Delta\mathbf{v} = -\eta\nabla C$, where $\eta = \dfrac{\varepsilon}{\|\nabla C\|}$.*

Proof. By the Cauchy-Schwarz Inequality, we have

$$|\nabla C \cdot \Delta\mathbf{v}| \leq \|\nabla C\| \times \|\Delta\mathbf{v}\|. \tag{3.4.4}$$

The minimum is given by,

$$\min(\nabla C \cdot \Delta\mathbf{v}) = -\|\nabla C\| \times \|\Delta\mathbf{v}\|. \tag{3.4.5}$$

Substituting $||\Delta\mathbf{v}|| = \varepsilon$ into Equation (3.4.5), we have

$$\min(\nabla C \cdot \Delta\mathbf{v}) = -\varepsilon\|\nabla C\|. \tag{3.4.6}$$

Multiplying the numerator and denominator of Equation (3.4.6) by $||\nabla C||$ gives

$$\min(\nabla C \cdot \Delta\mathbf{v}) = -\frac{\varepsilon\|\nabla C\|^2}{\|\nabla C\|} \tag{3.4.7}$$

$$= -\frac{\varepsilon\nabla C \cdot \nabla C}{\|\nabla C\|}. \tag{3.4.8}$$

By equating the coefficients of ∇C between Equations (3.4.5) and (3.4.8), and noting that $\eta = \varepsilon/||\nabla C||$,

$$\begin{aligned}\arg\min_{\Delta\mathbf{v}}(\nabla C \cdot \Delta\mathbf{v}) &= -\frac{\varepsilon\nabla C}{\|\nabla C\|} \\ &= -\eta\nabla C.\end{aligned}$$

□

The Gradient Descent algorithm moves the coefficients in the direction of the steepest negative gradient. It assumes that the starting values are close enough to the global minimum to converge. If this assumption is not satisfied, the algorithm

will instead converge to the local minimum. In this thesis, we are only concerned about estimating the local minimum as attaining the global minimum results in overfitting of the training data.

The Gradient Descent algorithm relies on the gradients of a large number of variables. We now discuss a number of techniques that improve algorithm efficiency, including the Stochastic Gradient Descent algorithm, the choice of learning rate and the Adam Optimiser.

Stochastic Gradient Descent

Gradient Descent is a highly computationally intensive algorithm as the neural network has a very large number of weights. To improve training time, Stochastic Gradient Descent is a popular alternative as it improves the time complexity from $O(n^3)$ (Marquardt, 1963) to $O(n)$ (Robbins and Monro, 1951). This algorithm improves learning speed by randomly selecting a small subset of the training set to learn from, referred to as a *batch*. When training has been completed for that batch, a new batch of training inputs is randomly selected and the process repeats. Once training has occurred over all batches, it is said that an *epoch* of training has been completed. In this manner, a relatively small number of samples is used for each weight adjustment. This drastically increases training speed whilst utilising all the information provided by the entire training set.

Learning rate

During Gradient Descent, the change in gradients $\Delta \mathbf{v} = -\eta \nabla C$ moves the weights in the direction of the steepest negative gradient (see Section 3.4). The learning rate η controls the magnitude of this movement. If η is too small, the adjustment of weights is also small and model training will take a long time. If η is too large, the weights may be shifted too far and the values of $\mathbf{v}$ will continuously move past the local minimum, as shown in Figure 3.3.

To aid in efficient training, it can be beneficial for learning rates to be adjusted during the training process. Higher initial learning rates allow for fast convergence

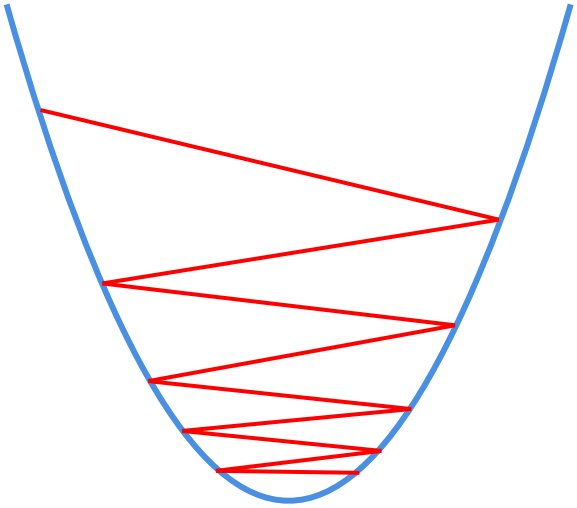

Figure 3.3: High learning rates adjust weights past the local minimum

toward the local minimum, and lower learning rates at later epochs give the algorithm greater precision.

Adam Optimiser

The Adam Optimiser is an algorithm that introduces additional learning rates to the Gradient Descent algorithm. Different learning rates are applied to each of the weights in the neural network, allowing for faster convergence of the cost function to the minimum (Kingma and Ba, 2014).

3.5 The Backpropagation algorithm

During Gradient Descent, it is necessary to calculate the gradient of the cost function with respect to the weights (W, B). Let $z_j^\ell = \mathbf{w}_j^\ell \cdot \mathbf{a}^{\ell-1} + b_j^\ell$ be the value of the j-th neuron of the ℓ-th layer before the activation function is applied. The Backpropagation algorithm is an efficient algorithm that determines the $\dfrac{\partial C}{\partial z}$ and relates them to the rates of interest, $\dfrac{\partial C}{\partial W}$ and $\dfrac{\partial C}{\partial B}$ (Nielson, 2015). The following lemmas are necessary to understand the Backpropagation algorithm.

Lemma 3.5.1. *The error of the j-th neuron of the final output layer is*

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L).$$

The vector of output layer errors can be expressed in the matrix form,

$$\delta^L = \Sigma'(\mathbf{z}^L)\nabla_a C, \tag{3.5.1}$$

where $\Sigma'(\cdot)$ is a matrix such that the j-th diagonal entry is $\sigma'(z_j^L)$ and all non-diagonal entries are 0.

Proof. Noting that z_j^L is independent of a_k^L for all $k \neq j$, we can write

$$\begin{aligned}
\delta_j^L &= \frac{\partial C}{\partial z_j^L} \\
&= \sum_k \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_j^L} \\
&= \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} \\
&= \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L).
\end{aligned}$$

□

Lemma 3.5.2. *The error δ_j^ℓ of the ℓ-th layer can be written in terms of the errors of the $(\ell+1)$-th layer,*

$$\delta_j^\ell = \sum_k \delta_k^{\ell+1} w_{kj}^{\ell+1}\sigma'(z_j^\ell).$$

The vector of errors δ^ℓ for the ℓ-th layer can be expressed in the matrix form,

$$\delta^\ell = \Sigma'(\mathbf{z}^\ell)(w^{\ell+1})^\intercal \delta^{\ell+1}. \tag{3.5.2}$$

Proof. Using the chain rule,

$$\begin{aligned}\delta_j^\ell &= \frac{\partial C}{\partial z_j^\ell} \\ &= \sum_k \frac{\partial C}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial z_j^\ell} \\ &= \sum_k \delta_k^{\ell+1} \frac{\partial z_k^{\ell+1}}{\partial z_j^\ell}.\end{aligned}$$

Noting that $\dfrac{\partial z_k^{\ell+1}}{\partial z_j^\ell} = \dfrac{\partial}{\partial z_j^\ell}(\mathbf{w}_k^{\ell+1} \cdot \sigma(\mathbf{z}^\ell) + b_k^{\ell+1}) = w_{kj}^{\ell+1}\sigma'(z_j^\ell)$, we can write δ_j^ℓ as

$$\delta_j^\ell = \sum_k \delta_k^{\ell+1} w_{kj}^{\ell+1} \sigma'(z_j^\ell).$$

□

Lemma 3.5.3. *The error δ_j^ℓ is equivalent to the rate of change in the cost function with respect to the bias, such that*

$$\delta_j^\ell = \frac{\partial C}{\partial b_j^\ell}. \tag{3.5.3}$$

Proof. Noting that z_j^ℓ is independent of b_k^ℓ for all $j \neq k$,

$$\begin{aligned}\delta_j^\ell &= \frac{\partial C}{\partial z_j^\ell} \\ &= \sum_k \frac{\partial C}{\partial b_k^\ell} \frac{\partial b_k^\ell}{\partial z_j^\ell} \\ &= \frac{\partial C}{\partial b_j^\ell} \frac{\partial b_j^\ell}{\partial z_j^\ell} \\ &= \frac{\partial C}{\partial b_j^\ell}.\end{aligned}$$

□

Lemma 3.5.4. *The rate of change in cost with respect to any single weight value is given by*

$$\frac{\partial C}{\partial w_{jk}^{\ell}} = a_k^{\ell-1}\delta_j^{\ell}. \tag{3.5.4}$$

Proof. By definition,

$$z_k^{\ell} = \mathbf{w}_k^{\ell} \cdot \mathbf{a}^{\ell-1} + b_k^{\ell}.$$

Differentiating with respect to some weight w_{km}^{ℓ},

$$\frac{\partial z_k^{\ell}}{\partial w_{km}^{\ell}} = a_m^{\ell-1}.$$

Using the chain rule,

$$\frac{\partial C}{\partial w_{jk}^{\ell}} = \frac{\partial C}{\partial z_j^{\ell}} \frac{\partial z_j^{\ell}}{\partial w_{jk}^{\ell}} = \delta_j^{\ell} a_k^{\ell-1}.$$

□

The Backpropagation algorithm feeds an input $\mathbf{x}$ fowards through the network to determine the error in the final layer δ^L (Equation (3.5.1)). The algorithm moves backwards through the network from the final layer back to the first hidden layer to determine the remaining errors in the network δ^{ℓ}, $\ell < L$ (Equation (3.5.2)). From these errors, the cost gradients in terms of the weights and biases can be calculated (Equations (3.5.3) and (3.5.4)). The complete algorithm is detailed in Algorithm 1. Dependence on the gradient of the cost function makes the algorithm vulnerable to slow training if the gradients are close to zero. This motivates our discussion of the *vanishing gradient problem.*

Algorithm 1: Calculation of gradients during backpropagation

for $\mathbf{x}$ *in* $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ **do**

$\mathbf{a}^1 = \mathbf{w}^1 \cdot \mathbf{x} + \mathbf{b}^1$

FEEDFORWARD

for ℓ *in* $2, 3, \ldots, L$ **do**

$\mathbf{z}^\ell = \mathbf{w}^\ell \cdot \mathbf{a}^{\ell-1} + \mathbf{b}^\ell$

end

$\delta^L = \Sigma'(\mathbf{z}^L)\nabla_a C$

BACKPROPAGATION

for ℓ *in* $L-1, L-2, \ldots, 2$ **do**

$\delta^\ell = \Sigma'(\mathbf{z}^\ell)(w^{\ell+1})^\intercal \delta^{\ell+1}$

end

$\dfrac{\partial C}{\partial b_j^\ell} = \delta_j^\ell$

$\dfrac{\partial C}{\partial w_{jk}^\ell} = a_k^{\ell-1}\delta_j^\ell$

end

Vanishing gradient problem

The *vanishing gradient problem* refers to the stagnation of weights and biases during the Gradient Descent algorithm caused by very small activation function gradients. The errors δ^ℓ calculated during backpropagation are dependent on the gradient of the activation function (see Equation (3.5.1)). If this gradient becomes too small, then the errors δ and movement of weights $\Delta\mathbf{v}$ also become small (see Equation (3.4.3)), limiting the movement of weights and biases. For the sigmoid activation function,

$$\sigma'(z) = e^{-z}(1 + e^{-z})^{-2},$$

and for the hyperbolic tan activation function,

$$\sigma'(z) = 1 - \tanh^2(z).$$

For both the sigmoid and tanh activation functions, the limit as $z \to \pm\infty$ yields

$$\lim_{z\to\pm\infty} \sigma'(z) = 0,$$

causing stagnation of the movement of weights and biases. The ReLU activation function avoids this by having gradient

$$\sigma'(z) = \begin{cases} 1, & z > 0 \\ 0, & z < 0 \\ \text{not defined}, & z = 0 \end{cases}.$$

Neurons that return a positive z_j^ℓ have a large gradient and so remain actively trained. If the value of z_j^ℓ becomes negative, the activation will have a gradient of 0 and the neuron will cease training. It should be noted that, in practice, the gradient of ReLU activation will output 0 at $z = 0$ for simplicity (Goodfellow et al., 2016).

3.6 Weight initialisation

The initialisation of weights and biases affects the rate and quality of training (Mishkin and Matas, 2016). Intuitively, if the weights are initialised close to the final output values, it will be faster to train. Conversely, weights that are initialised poorly may take a longer time to train to the same accuracy, or the local minimum of the cost function may not converge close to the global minimum and produce a poor model.

If the weights in two neurons are similar, using the same activation function will result in similar outputs. Therefore, initialising weights and biases with the same values introduces a lot of redundant calculations. It is commonplace for weights and biases to be initialised randomly (Nielson, 2015). An example of an initialisation distribution is the truncated normal distribution. Let $X \sim \mathcal{N}(\mu, \sigma^2)$

and $-\infty \leq a < b \leq \infty$. For $x \in (a, b)$, the truncated normal X has probability density function (PDF)

$$f_X(x; \mu, \sigma^2, a, b) = \frac{\phi\left(\frac{x-\mu}{\sigma}\right)}{f\left(\Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)\right)},$$

where ϕ and Φ are the standard Normal PDF and cumulative density function (CDF) respectively. This method initialises the weights with a small amount of noise, with truncation of the Normal distribution helping to avoid vanishing gradients should the sigmoid or hyperbolic tan activation functions be used (see Section 3.5).

In the case that the ReLU activation function is used, the He Method (He et al., 2015) is a common initialisation algorithm. This method recognises the use of the ReLU activation function and scales its initialisations dependent on the size of the previous layer. According to He et al. (2015), the method helps to constrain the variance of the weights during training and thus improves training time and quality. The He Method samples from the Normal distribution $\mathcal{N}(0, 2/d_{\ell-1})$, where $d_{\ell-1}$ is the number of neurons in the $(\ell - 1)$-th layer.

3.7 Regularisation

Due to the large number of parameters to be estimated, parameters generally out-number observations. During training, observations must be reused and so neural network models have a tendency to overfit the data (Nielson, 2015). *Regularisation* is the process of mitigating the tendency to overfit data by applying penalties to the network as it trains. We now cover some regularisation techniques, including L2-regularisation, dropout, batch normalisation, data expansion, and early stopping.

L2-regularisation

Penalties can be applied to the cost function to artificially increase the error and promote model training. *L2-regularisation* adds a penalty term of scaled model

weights, similar to that of Ridge regression. The scaling of the weights is controlled by the regularisation parameter λ to give the updated cost function

$$C(W, B) + \frac{\lambda}{2n} \sum_{v \in (W,B)} v^2.$$

Dropout

During training, it is possible for individual neurons to become sensitive to patterns present in specific observations. Dropout layers (Srivastava et al., 2014) assign each connected neuron connected with a preset probability p of being deactivated, regardless of their input. This ensures that relevant data features are spread through several neurons, and that the impact of an individual neuron does not strongly affect the final result.

Batch normalisation

Deep neural network models have a large number of variables and so require a longer training time. Training through many epochs can cause some activation values to become very large. Batch normalisation involves normalising the activation values within each training batch. This stops the activations from becoming too large and also avoids overfitting by adding some noise to the weights and ensuring that critical features are spread across nodes (Ioffe and Szegedy, 2015).

Data expansion

Neural networks require a very large sample size to avoid overfitting as there are a large number of variables. If the number of samples is not large enough, additional data samples can be created by augmenting the existing data set (Perez and Wang, 2017). Common augmentations for image data include flipping horizontally and vertically, rotation and cropping. Note that not all augmentations will be valid for each context. For instance, images of text should not be flipped.

Early stopping

Lengthy model training can cause extensive overfitting of the data. Early stopping of finalises the model before severe overfitting takes place and testing performance drops (Prechelt, 2012). A common early stopping method is to save snapshots of the model during the training process and select the model that performs the best on the validation set.

Chapter 4

Convolutional neural networks

Convolutional neural networks (CNNs) are of particular interest when working with image problems as the input data is assumed to be two-dimensional. In regular neural networks (see Chapter 3), the neurons in each layer are connected to all the neurons of the previous layer. These layers are known as *fully connected layers*. In CNNs, convolutional layers examine only small subimages of the entire image sample. Each pixel of an image forms a neuron, which is connected only to nearby pixels, rather than all pixels in the image. The CNN is trained to identify a set of visual features which can be combined and interpreted for image classification tasks.

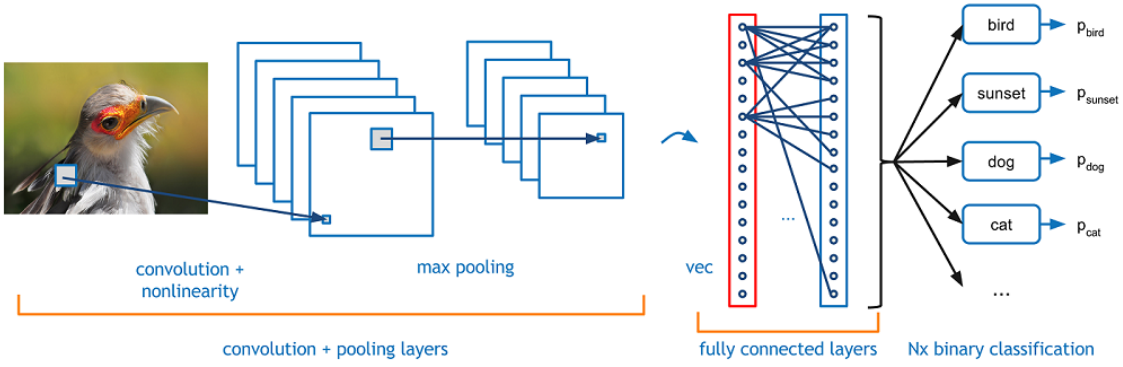

Figure 4.1: Typical CNN Structure to classify an image from a finite set of classes. Convolution layers with ReLU activation (nonlinearity) are followed by pooling layers. The final layers are fully connected, using softmax activation to generate a probability distribution.

Image taken from Deshpande (2016)

CNNs typically consist of alternating convolutional layers (see Section 4.1) and max pooling layers (see Section 4.2), as shown in Figure 4.1. Max pooling layers occur after convolutional layers to reduce the network dimensionality caused by image data. For classification tasks, the last layers of the network are fully connected to ensure the final outputs match the structure of the given responses. Softmax activation (see Section 3.2) is used for the final output layer to output the classifications as a probability distribution.

4.1 Convolutional layers

The neurons of convolutional layers have a similar overall structure to that of regular neurons (see Section 3.1). The input variables X are multiplied with weights W, summed, and added to a bias variable b. The resulting linear combination is passed through an activation function $\sigma(\cdot)$ to give the output of the neuron with activation value a.

The major difference between convolutional layers and regular neural network layers is the structure of the inputs and weights. Image data is two-dimensional and generally has multiple colour *channels*. For example, a coloured image contains three channels: red, green, and blue. The image can be represented as a three-dimensional array with its depth equal to the number of colour channels. The weights of convolutional layers are also formatted as three-dimensional arrays to accommodate for the depth of the input images. Unlike fully connected layers, convolutional layers do not multiply the weights to all of the input variables at once. Instead, the array of weights has a height and width designed to identify visual features in smaller subimages. A single array of weights that describes a feature is called a *filter*. Each convolutional layer can have multiple filters, analogous to having multiple neurons in a fully connected layer.

Each filter processes the whole image by multiplying element-wise with all existing subimages chosen sequentially. The process, known as *convolution*, begins at the top-left of the image and moves towards the bottom-right. Let the f-th filter of the convolutional layer be $F^{(f)}$, with dimension $M \times N \times C$. Given an input image

matrix X, number of channels C, and the bias of the f-th filter $b^{(f)}$, the value of the convolution at element $X_{j,k}$ is given by

$$a_{jk}^{(f)} = \sigma\left(\sum_{c=1}^{C}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} X_{j+m,k+n,c}F_{m,n,c}^{(f)} + b^{(f)}\right). \tag{4.1.1}$$

Let us consider an example.

Example 2. A convolution is to be applied to an input image with one channel, with identity activation function $\sigma(z) = z$.

Input $3 \times 3 \times 1$

1	0	1
2	0	2
1	1	-1

Filter $2 \times 2 \times 1$

1	0
-1	1

Bias

1

The filter in red is multiplied element-wise with the inputs given in blue. This gives:

1	0
-2	0

Taking the sum of the elements of the array and adding the bias in purple gives the activation value $(1 + 0 + -2 + 0) + 1 = 0$. This process is iterated for all 2×2 subimages in the original input, giving the following output:

Output $2 \times 2 \times 1$

0	3
3	-1

The output arrays of the convolutional filters are bound together to form a three-dimensional array, with depth dependent on the number of filters in the layer.

The convolution process involves a very large number of variables, many of which are redundant when identifying important features. We now discuss a number of techniques to reduce image dimensionality, speed up the convolution process, and provide flexiblity in network design.

Zero padding

During convolution, the filters multiply element-wise within the boundaries of the input image, resulting in loss of dimension at the borders. For an input image with dimension $J \times K$ and a filter with dimension $M \times N$, the filter output will be of dimension $(J - M + 1) \times (K - N + 1)$. This places a limit on the number of convolutional layers used before the image dimension becomes too small for further convolution.

Zero padding surrounds the edges of the input image with P layers of zeros, increasing the input dimension to $(J + 2P) \times (K + 2P)$. When convolution takes place, the resulting image size has dimension $(J+2P-M+1) \times (K+2P-N+1)$. The value of P can be set such that the convolution maintains the original image dimension, allowing the network to continue applying convolution to outputs, thus producing very deep networks (Szegedy et al., 2015).

Stride

Convolving filters at all possible locations of large input images results in lengthy training time and large outputs. Applying the convolution of filters at every S-th subimage reduces the number of variables whilst maintaining use of the whole input image. The magnitude of the filter movement is known as the *stride*.

An input image with dimension $J \times K$ convolved with a filter with dimension $M \times N$, with padding P and stride S, will have an output dimension of $(\lfloor (J + 2P - M)/S \rfloor + 1) \times (\lfloor (K + 2P - N)/S \rfloor + 1)$.

4.2 Pooling layers

Convolutional layers are followed by pooling layers to reduce dimensionality whilst retaining variables that identify significant features (Deshpande, 2016). The model

is only interested in identifying variables for classification and does not need the location to be highly specific. A sliding window of size $M \times M$ and stride S is applied to the top-left of the image input and moves across the image similarly to the convolutional layers. At each location, the pooling layer outputs a single value, which collectively form a matrix of reduced dimension to the input.

For *max pooling*, the values output are the maximum values of all inputs in each sliding window. Max pooling is a preferred pooling method as it emphasises features of interest located by the previous convolutional layer.

4.3 ReLU activation

The computation of neurons and convolutional layer outputs can be time consuming as there are a large number of variables in image data. The ReLU function, as defined in Section 3.2, can be computed very quickly and consequently is a common choice for convolutional neural networks (Deshpande, 2016). Additionally, the usage of the ReLU activation function helps avoid the vanishing gradient problem (see Section 3.5).

Chapter 5

Existing methodologies

We will now discuss the structure and performance of existing lacune detection models in greater detail. We discuss the thresholds developed by Yokoyama et al. (2007) (Section 5.1), the location-based CNN by Ghafoorian et al. (2017) (Section 5.2), and outline potential changes and simplifications to these models (Section 5.3).

5.1 Thresholding

This section describes an existing attempt to automate the identification of lacunes by developing and applying thresholds for certain variables. In 2007, Yokoyama et al. (2007) developed an algorithm that identifies lacune candidates by examining their *area*, *circularity*, and *gravitational centre*. Area A is the number of pixels that the candidate lacune covers. Circularity is given by $C = 4\pi \times \frac{A}{\ell^2}$, where ℓ is the perimeter of the candidate lacune. The gravitational centre is given by

$$(g_x, g_y) = \left(\frac{1}{A} \sum_{i=1}^{A} x_i, \frac{1}{A} \sum_{i=1}^{A} y_i \right),$$

where (x_i, y_i) denote the coordinates of the pixels that the lacune covers.

Yokoyama et al. (2007) claim that lacunes primarily occur in the *basal ganglia*, a structure at the centre of the brain. Thus, the search for lacunes was limited to a central circular region with the coordinates of the centre and the radius determined for each scan.

Candidate lacunes satisfy thresholds based on area, circularity, and gravitational centre. The thresholds are established dependent on the intensity of the surrounding

structures. For this reason, candidates are separated into two categories: isolated lacunes and those surrounded by white matter hyperintensities (WMH). Isolated lacune candidates are determined by extracting the cerebral ventricle and calculating the mean pixel intensity. Candidates are determined as regions with an intensity below 70% of the mean value. Isolated candidate lacunes satisfy the inequalities

$$\begin{aligned} 19 \leq A \leq 200, \\ 0.45 \leq C, \\ \text{and } (g_x - c_x)^2 + (g_y - c_y)^2 < 12000, \end{aligned}$$

where (c_x, c_y) is the centre of the circular region of interest. The region defining accepted candidates is shown in Figure 5.1.

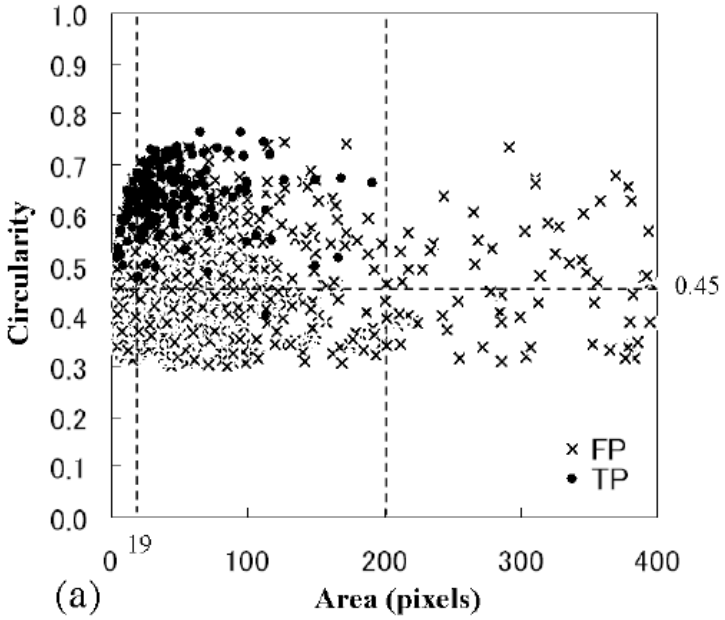

Figure 5.1: The threshold for candidate area and circularity is defined by the top-left region.

Image taken from Yokoyama et al. (2007)

Lacunes that appear next to WMH are more difficult to extract and are hence treated separately. These candidates are determined by taking differences between intensi-

ties of the region and cerebral ventricle CSF. Candidates satisfy the inequalities

$$25 \leq A \leq 100 \text{ and } 0.48 \leq C.$$

Once candidates are identified, the model eliminates false-positives by examining the T1-weighted scans. False-positive removal is based on a candidate's location, area, and gravitational centre.

The data set used to develop the model consisted of 100 scans totalling 832 axial slices. Model training and testing consisted of 20 and 80 scans respectively. The final algorithm exhibited a *sensitivity* rate (correct classification of positives) of 90.1% and an average of 1.7 false-positives per slice. Although the testing sensitivity was high, it was insufficient for the model to be usable on its own. Model improvement is required before it can be used to aid clinicians reliably. The low sensitivity rate may be the result of a restricted search area. Yokoyama et al. (2007) made the assumption that lacunes only occur deep within the brain, within the basal ganglia. This was the motivation behind defining a circular search region and extracting the cerebral ventricles. However, this assumption is not always satisfied. Though less common, lacunes have been found in other brain regions, including the cerebrum (see Section 6). Restricting the search area may inadvertently exclude lacunes from detection.

The model by Yokoyama et al. (2007) was dependent on a number of existing architectures. The first is the identification and extraction of particular brain regions. In this model, Yokoyama et al. (2007) were able to use existing binarisation techniques to extract the cerebral ventricle. Each sample also requires a calculated area, circularity, and gravitational centre. Each of these calculations requires sufficient time and resources, particularly if these calculations are to be conducted manually.

The comparatively low sensitivity may also be the result of too few variables. An indicator for hyperintensive rims and the area and circularity from associated

FLAIR images may improve the sensitivity rate. However this was not tested due to time and data constraints.

5.2 Convolutional neural networks

We will now discuss the convolutional neural network structure utilised by Ghafoorian et al. (2017). This model is a supervised machine learning model and forms the basis of our adjusted location-independent model. It consists of two phases; candidate generation using a CNN, and false-positive reduction using a three-dimensional CNN and location-based variables.

Candidate detection model

The data set consisted of 1,075 MRI scans split into training, validation, and testing sets of size 868, 96, and 111 respectively. Each scan contained two image weightings: T1-weighted and FLAIR. Data samples were generated by randomly sampling axial slices of dimension 51×51 pixels, each called a *patch*. The model was trained to detect lacunes at the centre of each patch, returning either a positive response $(1, 0)^\intercal$ or negative response $(0, 1)^\intercal$. Combining the T1-weighted image, FLAIR and response gave samples containing two 51×51 pixel images and a response vector in $\mathbb{R}^2$. For use in convolution, the T1-weighted image and the FLAIR image were treated as separate colour channels. Samples were generated such that there were twice as many negative samples as positive, totalling 3.2×10^5 training samples.

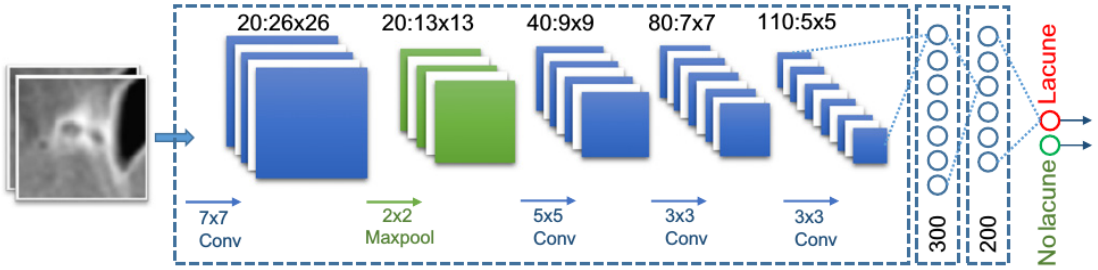

Figure 5.2: Candidate detection model structure.
Image taken from (Ghafoorian et al., 2017)

The CNN is made of seven layers, as shown in Figure 5.2. The first four are convolutional layers, followed by three fully connected layers. The four convolutional

layers respectively contain 20, 40, 80 and 110 filters, with filter sizes 7×7, 5×5, 3×3, and 3×3. It should be noted that Ghafoorian et al. (2017) did not discuss any dimension reduction prior to the first convolutional layer, however the included model diagram appears to apply stride or pooling techniques before the first convolutional layer, reducing the image dimension from 51×51 to 26×26. A max pooling layer is included between the first and second convolutional layer, with a filter size of 2×2 and stride of 2. No zero padding is applied to the convolutional or pooling layers. The convolutional layer is followed by three fully connected layers of 300, 200, and 2 neurons.

Batch normalisation and ReLU activation is applied to all neurons. Dropout is applied to the fully connected layers with a rate of 0.3. All weights are initialised using the He method (He et al., 2015). Training is conducted by stochastic gradient descent with the Adam optimiser. Each training batch has a sample size of 128. A decaying learning rate is used, from 5×10^{-4} to 10^{-6} by the last epoch. Cross-entropy loss is used to assess the model at the end of each batch. L2-regularisation is used with the penalty term set to 10^{-4}. The final layer applies softmax activation to ensure the output follows a probability distribution. An early stopping algorithm is used such that the model with the highest validation accuracy is saved.

Ghafoorian et al. (2017) comment that the model can be computationally intensive to run on many data points. To improve speed, the fully connected layers are converted into convolutional layers. Candidate lacunes are identified by feeding 51×51 samples of the tested scan through the trained CNN. For each given sample, the model outputs a lacune classification probability. Once this has been completed for the whole volume, a 10×10 sliding window identifies local maximums in the resulting probabilities. Sliding window samples with lacune probabilities below 0.1 are removed and the remaining samples are lacune candidates to be input into the second phase of the model. Note that there were no performance diagnostics provided for the first model stage.

False-positive reduction model

The second phase of the model serves to reduce the number of false-positives returned by the first stage of the model. This model uses a three-dimensional CNN, adding additional contextual information to the image. To capture varying levels of contextual information, each data point is captured at three different resolutions, 32×32×5, 64×64×5, and 128×128×5, These samples are reduced to dimension 32×32×5 before being input into the model to lower the number of variables. In total, there were 3.85×10^5 samples for training, and 3.5×10^4 for validation.

Each resolution is fed through its own 3D CNN, as shown in Figure 5.3. Each CNN branch contains six convolutional layers and a fully connected layer. The convolutional layers respectively have 64, 64, 128, 128, 256, and 256 filters of sizes 3×3×2, 3×3×2, 3×3×1, 3×3×1, 3×3×1, and 3×3×1.

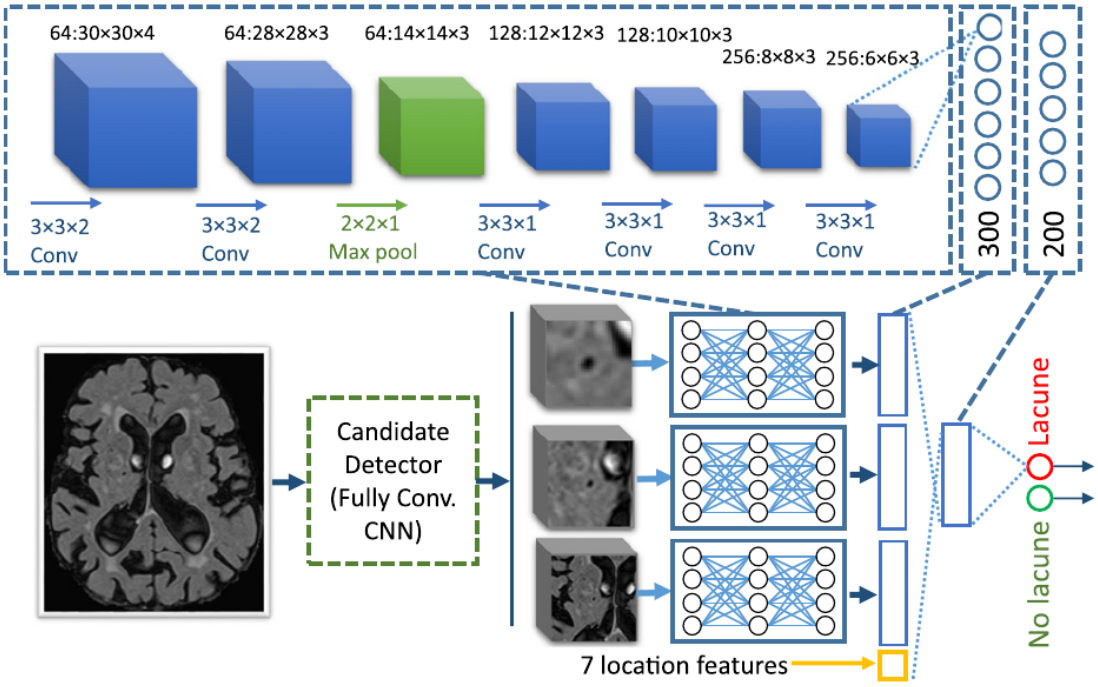

Figure 5.3: False-positive reduction model structure.
Image taken from Ghafoorian et al. (2017)

A single max pooling layer is placed after the second convolutional layer, of size 2×2×1. In each resolution branch, the final convolutional layer is followed by a fully connected layer of 300 neurons. The 3×300 fully connected neurons are

concatenated together with seven additional location variables. For each candidate, these location features include the x, y, and z coordinates, and distances to the left and right ventricles, cortex, and midsagittal brain surface.

The 907 fully connected neurons are passed to two final fully connected layers of size 200 and 2 neurons. The final layer consists of a softmax activation to output the lacune probabilities. All weights are He initialised. All neurons, except the output layer, have ReLU activation and are batch normalised. The fully connected layers have a dropout rate of 0.5. The cost function chosen was cross-entropy with L2-regularisation and penalty parameter 2×10^{-15}. This was minimised using stochastic gradient descent with an Adam optimiser. The learning rate is initialised at 5×10^{-4}. If training accuracy drops, the learning rate decays by a factor of 2. Each training batch contained 128 samples.

The model was trained for 40 epochs, with the final model and hyper-parameters chosen to maximise validation accuracy. The resulting model exhibited a sensitivity of 97.4% and an average of 0.13 false-positives per slice.

5.3 Proposed changes

In this study, we simplify the model by Ghafoorian et al. (2017) by assessing an adapted form of their candidate lacune detection model. In assessing the model with this structure, we remove the dependence on location-based variables. The models trained by Uchiyama et al. (2007c), Yokoyama et al. (2007), and Ghafoorian et al. (2017) each rely on location-based variables generated for each candidate lacune. Ghafoorian et al. (2017) claimed that the inclusion of location-based variables helps to differentiate lacunes from perivascular spaces, which occur primarily in the basal ganglia. However, lacunes can occur throughout the white and grey matter of the brain (Wardlaw et al., 2013a) and so adding these location variables could result in incorrect classification. In addition, the inclusion of location data requires extra data pre-processing. The location of brain structures is not inherently included with MRI data, and so efficient collection of these data must be coordinated using existing algorithms (Uchiyama et al., 2007c,b). In the case that estimation of the

location-based variables cannot be automated, the measurements have to be made manually for each candidate lacune.

Our model makes three major adjustments to the lacune detection model by Ghafoorian et al. (2017). The first is the exclusion of location-based variables. This ensures that no extra data collection is required in addition to the input MRI. The second adaptation is the extraction of brain tissue from the T1-weighted scans as a form of data de-identification (see Section 6.2). This process has the effect of enlarging regions of high CSF probability, including lacunes. The final adaptation is the delay of image dimension reduction until after the first convolutional layer. The model structure outlined by Ghafoorian et al. (2017) indicates early dimension reduction, with input data reduced from 51×51 to 26×26 after the first layer. This rapid loss in dimensionality may result in the removal of vital visual information, particularly if the lacunes being analysed are small. Instead, the image data dimensions are maintained and no stride or pooling is applied until the max pooling layer, indicated in Figure 5.2.

Chapter 6

Data sample collection

The retrieved data set contains MRI scans of two weightings: T1-weighted and FLAIR images; and an accompanying spreadsheet describing the anatomical locations of associated lacunes. In this chapter, we describe the source and format of the scans (Section 6.1), the effects of brain tissue extraction (Section 6.2), the generation of response values from the given spreadsheet (Section 6.3), and the final structure of each sample (Section 6.4).

6.1 MRI and preprocessing

The MRI and lacune location data sets were collected as a part of the Sydney Memory and Aging Study (Sydney MAS) conducted at the University of New South Wales' Centre for Healthy Brain Ageing, and were sourced from the second wave of MAS scans. A total of 411 scans were collected, of which 35 contain lacunes. They were acquired using a Philips 3T Achieva Quasar Dual scanner (Philips Medical Systems, The Netherlands). For radiologists' reference, the scanning parameters for the T1-weighted and FLAIR images are:

T1-weighted MRI - TR = 6.39 ms, TE = 2.9 ms, flip angle = 8°, matrix size = 256×256, field of view = 256×256×190, and slice thickness = 1 mm with no gap in between, yielding 1×1×1 mm^3 isotropic voxels.

FLAIR - TR = 10 000 ms, TE = 110 ms, TI = 2800 ms, matrix size = 512×512, slice thickness = 3.5 mm without gap, and in-plane resolution = 0.488×0.488 mm.

The FLAIR images were transformed using SPM12 (2017), such that their coordinates correspond to those from the T1 scans.

6.2 Extracting soft tissue

The resolution of T1-weighted images is high enough that it is possible to identify patients through their face structure and eyes. Brain matter (soft tissue) masks were generated to remove features that are not part of the brain tissue and de-identify the data.

Individual T1 images were segmented into grey matter, white matter, and CSF probability maps using the segmentation tool in SPM12 (2017). Grey matter and white matter probabilities were summed and voxels at a threshold of 0.5 or greater were included in the soft tissue mask. These masks were applied to each of the T1-weighted scans, as shown in Figure 6.1.

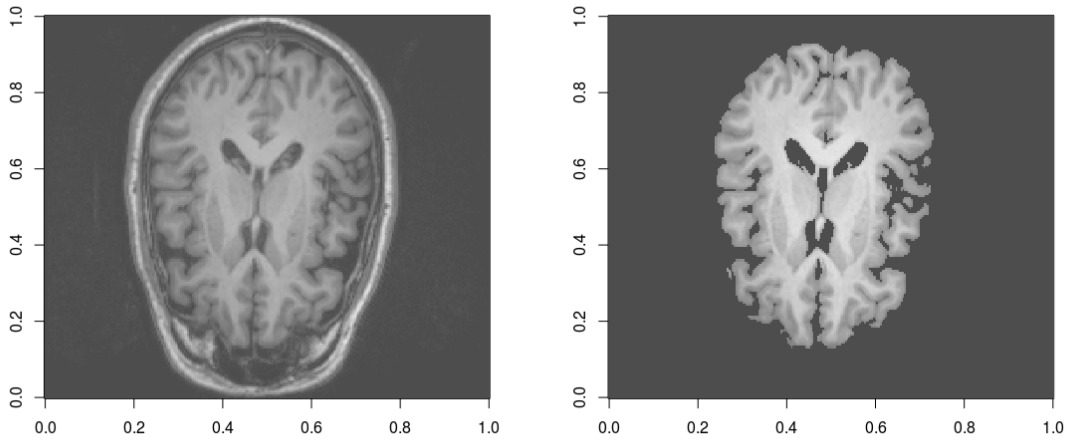

Figure 6.1: Original T1-weighted image and the extracted soft tissue.

The resulting images retain voxels that are likely to contain brain tissue. Other features, such as the skull and suspected CSF, are given an intensity of zero. Lacunes are filled with fluid and so have a signal intensity similar to that of CSF. Consequently, lacunes are removed by the soft tissue mask. It should be noted that lacunes are still visible in the FLAIR images.

6.3 Generating response arrays

The T1-weighted and FLAIR scans were rated visually by trained clinicians in accordance to the STRIVE criterion Wardlaw et al. (2013b). The clinicians visually analysed the scans slice by slice, identifying possible lacunes, perivascular spaces, and other lesions. Each candidate lesion was analysed by a team of clinicians to confirm the identification. The rating of lacunes was logged in Microsoft Excel, detailing the scan ID and the number of lacunes in each MRI scan. For each lacune detected, the spreadsheet indicates the axial slice, diameter in millimetres, hemispheric location (side), and the ID of the surrounding brain region. A sample of the spreadsheet data is shown in Table 6.1.

Scan ID	No. of lacunes	Axial Slice	Diameter	Side	Region
42	2	102	7	L	3 (Cortex)
102	1	112	10	R	4 (Thalamus)
...					

Table 6.1: Sample spreadsheet data. Each row describes one scan, identifying the size and location of lacunes. Columnn headings are repeated so that multiple lacunes can be identified per scan ID.

The provided data describes the approximate anatomical location of lacunes. This format is not immediately usable to the model as it lacks precise coordinates. To resolve this, the spreadsheet was used as a guide to visually identify lacunes. Once found, an overlay of lacunes for the T1-weighted scans was generated such that each pixel in the image corresponds to a positive binary response.

The responses were generated in FSLView (2007), a program used by neuroscientists to view and annotate MRI scans in the `.nifti` file format. For each brain scan described in the Excel spreadsheet, FSLView (2007) was used to generate a zero-initialised three-dimensional array of the same dimensions as the corresponding T1-weighted scan. Lacunes were visually identified by examining the indicated brain structure for lesions that appear dark in the T1-weighted images and with a hyperintense rim in the FLAIR images. The voxels that form the identified lacunes

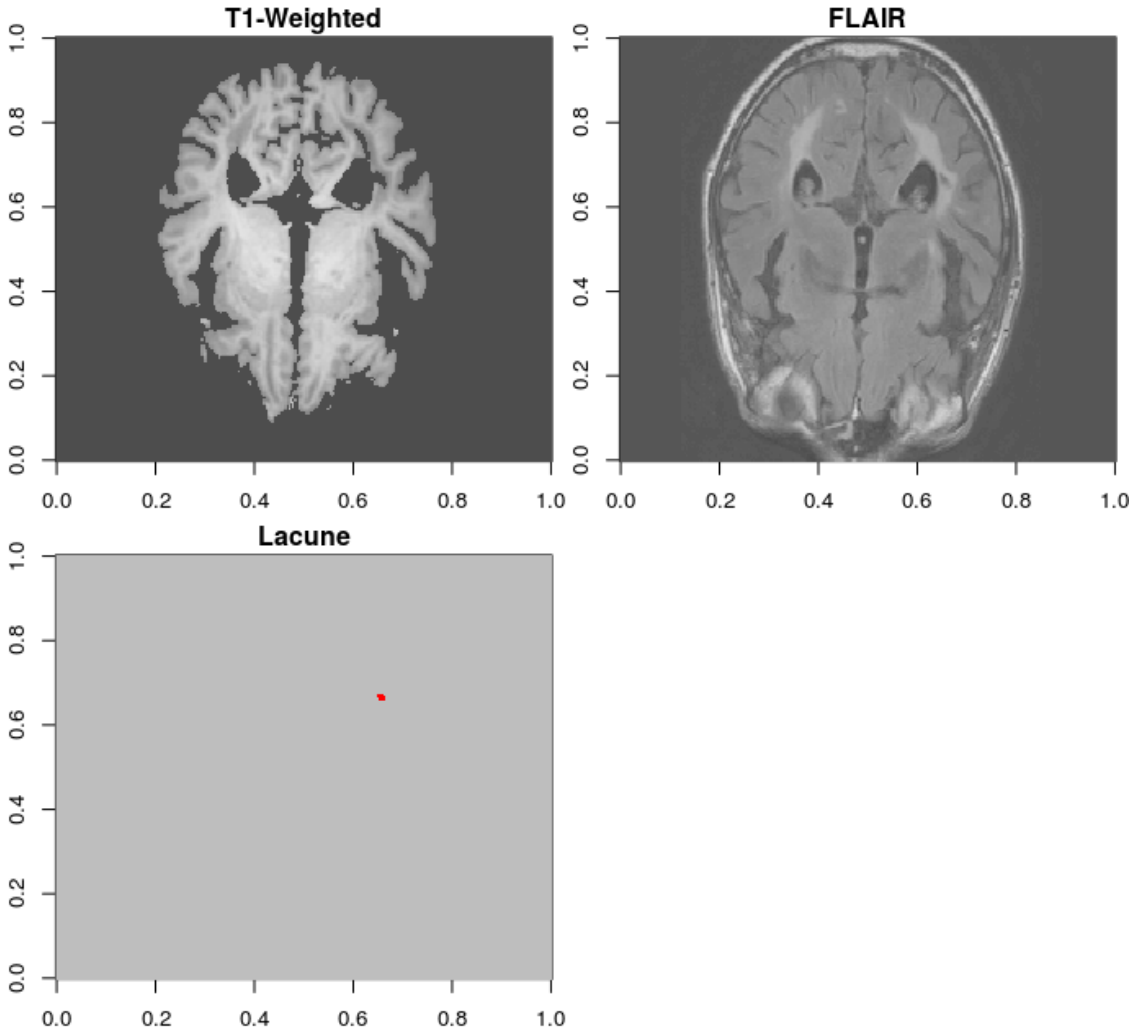

Figure 6.2: Comparison of T1-weighted images, corresponding FLAIR and lacune identification overlay (lacune in red).

were indicated with ones using the Brush tool. These overlays were saved as `.nifti` files so they could be imported alongside the soft tissue and FLAIR `.nifti` files. An example of a lacune overlay is shown in Figure 6.2.

6.4 Generating samples

The candidate generation model by Ghafoorian et al. (2017) specifies each sample to be 51×51 axial images of both T1-weighting and FLAIR. In their model, samples were chosen randomly such that positive lacune samples encompassed one-third of the data set. Data augmentation was used to increase the number of samples. In total, Ghafoorian et al. collected 3.2×10^5 training samples from 1,075 scans.

Our data set contains significantly fewer scans. In total the data set contains 411 MRI scans, of which 35 contain lacunes. The scans were imported into R and converted into three-dimensional arrays using the AnalyzeFMRI package (v1.1-17). Each value of an array is the MRI intensity at that voxel. Regions external to the scanned brain are given an intensity of zero.

The locations of the positive samples (lacunes) were extracted using the overlays generated in Section 6.3. For each nonzero value in the overlay, two 51×51-dimensional arrays were created of T1-weighted and FLAIR images. The pixel being classified occurs at the centre of each array. To increase sample size, additional augmented samples were formed by flipping the image horizontally. This method of sampling returned 3,846 lacune samples in total. Examples are shown in Figure 6.3.

Negative (non-lacune) samples were generated by considering voxels that return a negative response in the lacune overlay. The number of lacunes appearing in MRI is fairly low, and each lacune has a diameter up to only 15 mm. Therefore, the number of potential negative samples vastly outnumbers positive samples. Given too many negative samples, the model will have a tendency to classify each given sample as negative without the cost function reporting large errors. For example, a data set containing 99% negatives will return a 99% classification accuracy for a model that outputs all points as negative.

Negative samples were chosen in intervals of 25 pixels with starting locations chosen randomly. Samples were discarded if the central voxel was surrounded by a $4 \times 4 \times 4$ empty volume, reducing the number of sparse samples sourced externally to the brain. Examples are shown in Figure 6.4. This was used to generate a total of 39,983 negative samples. Positives samples make up 8.78% of the dataset.

Model training was conducted on three separate data sets: training, validation, and testing. The set of scan IDs containing lacunes and the set without lacunes were each split in the ratio 50:25:25. Samples were partitioned into the three separate

data sets by scan ID to ensure zero correlation between the samples during the validation and testing phases. The sampling method assumes independence of

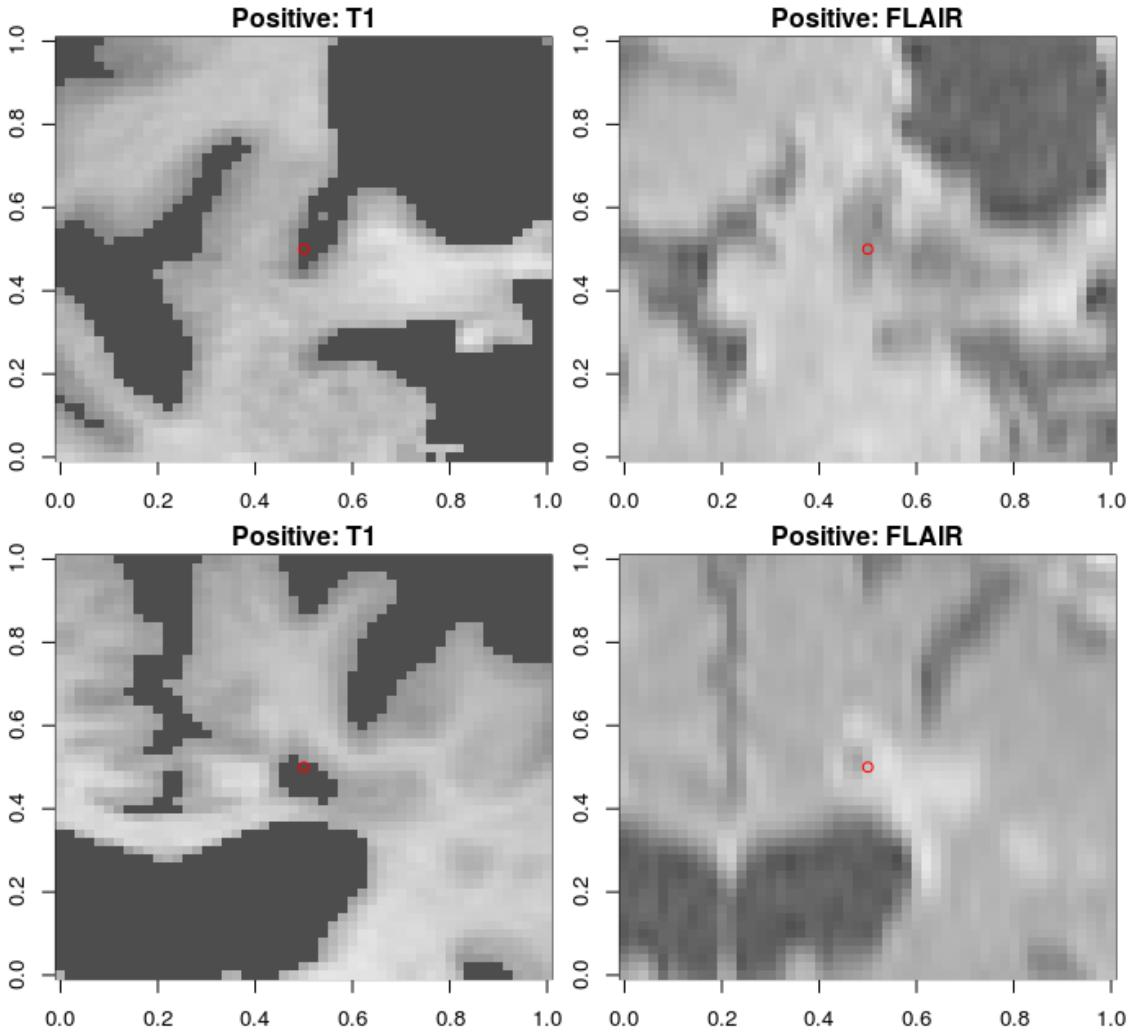

Figure 6.3: Examples of positive samples. T1-weighted images and corresponding FLAIR.

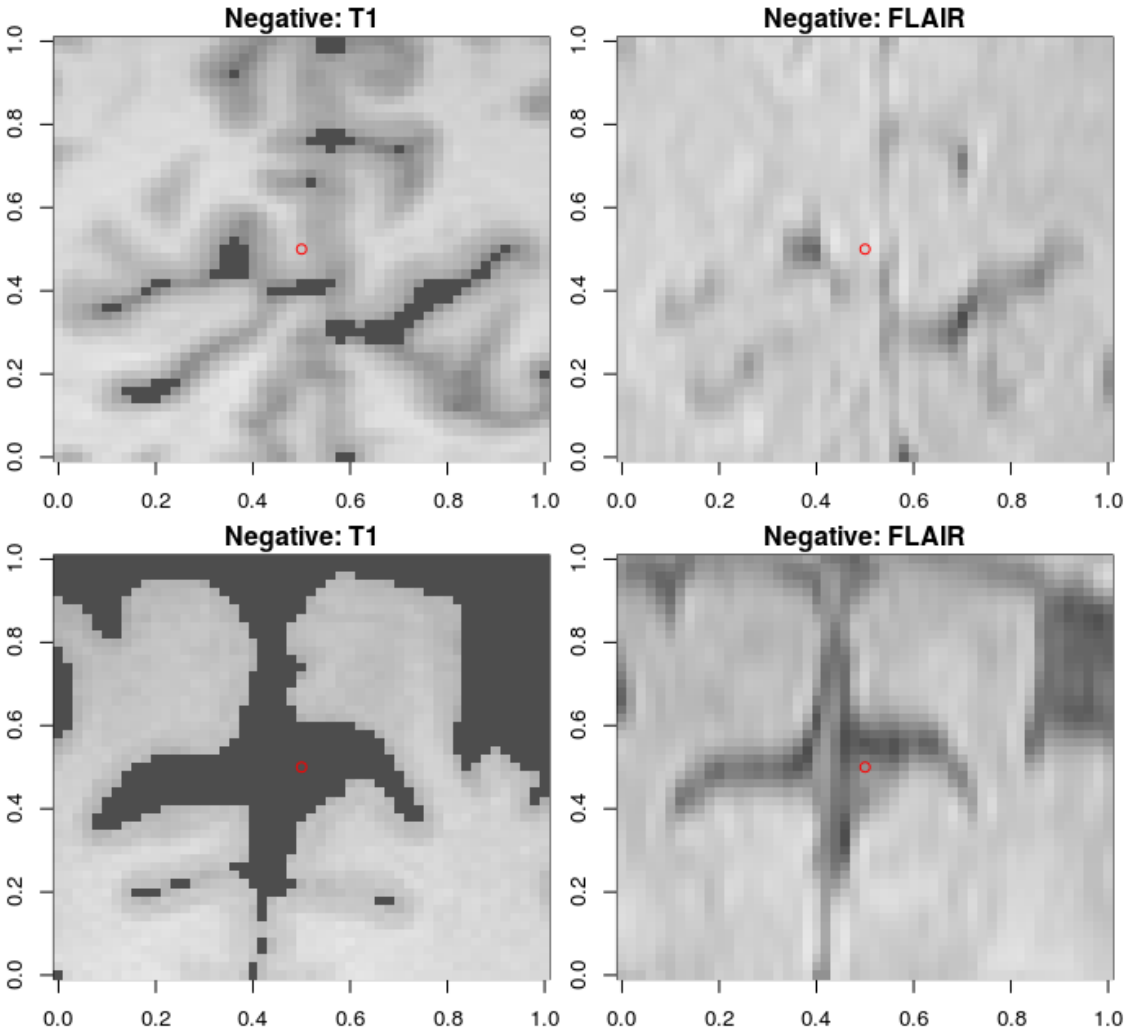

Figure 6.4: Examples of negative samples. T1-weighted images and corresponding FLAIR.

Chapter 7

Results

7.1 Final model structure

The final simplified model has a similar structure to that of the candidate detection model by Ghafoorian et al. (2017). The overall structure is shown in Figure 7.1. The input data contains two axial images: soft tissue extracted T1 and FLAIR images. Each of the images has resolution 51×51 pixels and is centred at the same coordinate. The model classifies the central pixel as either a positive (lacune) or negative (non-lacune) by returning the classification with the highest probability.

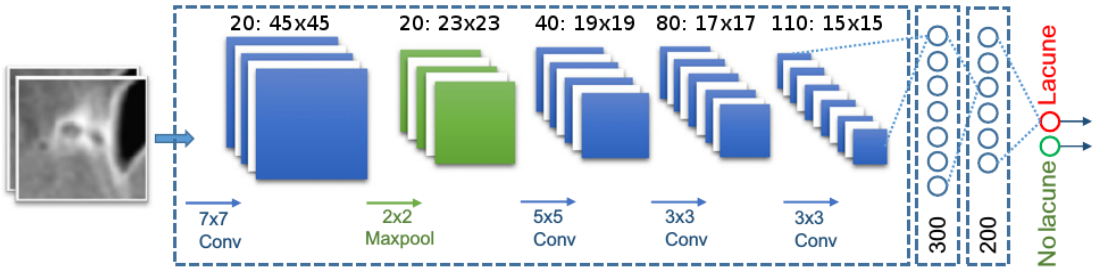

Figure 7.1: Simplified model structure.
Image adapted from Ghafoorian et al. (2017)

The model consists of four convolutional layers of size 20, 40, 80, and 110, with filter sizes respectively 7×7, 5×5, 3×3, and 3×3. A single max pooling layer is placed after the first convolutional layer, with size 2×2 and stride 2. The convolutional layers are followed by three fully connected layers of size 300, 200, and 2. Softmax activation is applied to the final layer to output a probability distribution.

All other layers have ReLU activation in order to avoid the vanishing gradient problem (see Section 3.5). All neurons undergo batch normalisation and are initialised using the He method (He et al., 2015). A dropout rate of 0.3 was applied to the fully connected layers.

Training was conducted using stochastic gradient descent with the Adam optimiser and a batch size of 128. A decaying learning rate was set from 5×10^{-4} to 1×10^{-6} by the 40th epoch. Cross-entropy cost was used with L2-regularisation of penalty rate 1×10^{-4}. An early stopping mechanism was implemented such that the model is tested against the validation set at the end of each epoch. If the validation accuracy improves, the model is saved. Once 40 epochs have run, we save the model with the highest validation accuracy.

Two models were trained using different positive-negative response ratios. The first model was developed using the positive-negative response ratio described by Ghafoorian et al. (2017): one-third positive and two-thirds negative. The second model was trained on the entire developed set of positive and negative lacune samples. Positive samples encompass 8.78% of data samples in this data set. All data sets were saved as R data files (`.Rda`).

7.2 Training environment

Model code was developed in R (v3.5.0) using RStudio (v1.1.453). The neural network was built and trained using Tensorflow (v1.10.0) through the R interface `tensorflow` (v1.8). The model was trained on a Linux machine running Ubuntu (release 16.04). The Tensorflow model was trained on the machine's CPU, an Intel® Core™ i7-4790 CPU 3.60GHz, with 16GB of RAM.

7.3 Results

First model

The first model generated has the same positive-negative sample ratio as outlined by Ghafoorian et al. (2017). The data consists of one-third positives and two-thirds

negatives, allowing for a total training sample size of 5,800. The model was trained for 40 epochs over 30 minutes.

Reusing the same data samples a large number of times introduces overfitting into the network (Goodfellow et al., 2016), where the accuracy of the training data set is very high compared to the accuracy of the validation and testing data sets. Hence, whilst assessing model training accuracy can efficiently convey early model improvement, it is not indicative of the model's performance when given new data.

Training accuracy was calculated every 5 batches of 128 samples. The resulting training accuracies are shown in Figure 7.2. Training accuracy increased rapidly within the first 200 batches. Model improvement occurred in steps, visible at batches 20, 40, and 100, as new features were found and more precise weight changes were made with the decreasing learning rate (Folly and Venayagamoorthy, 2009; Nielson, 2015). The model reported a 100% training accuracy consistently by

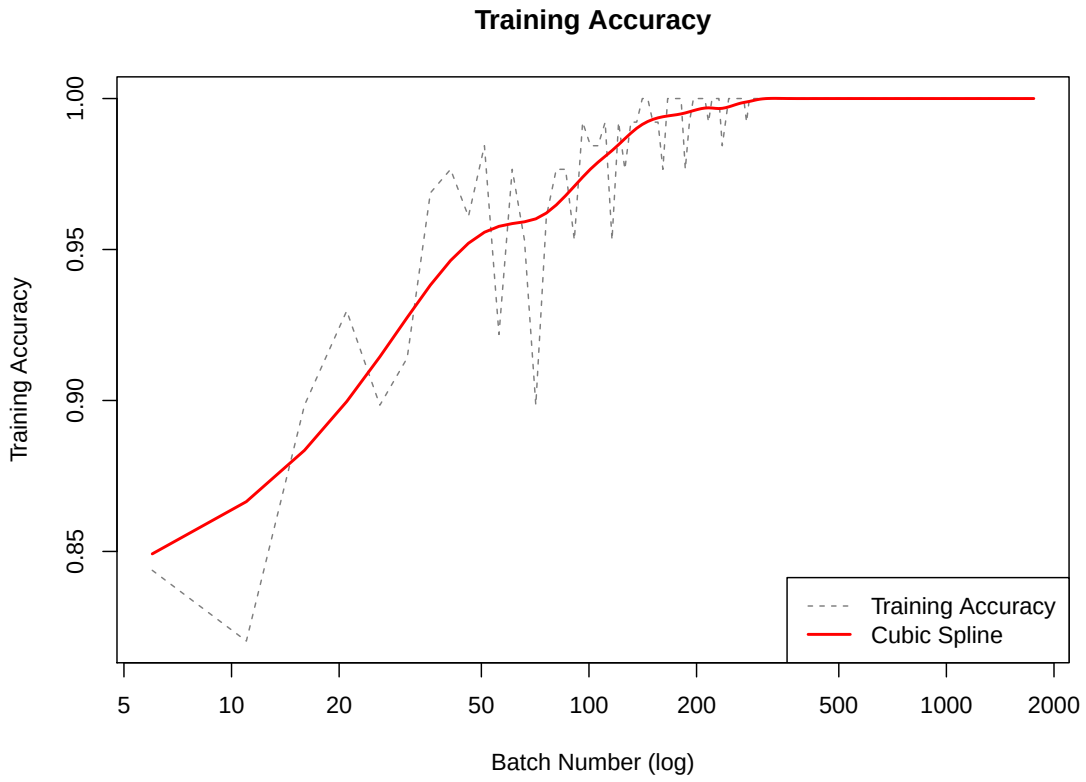

Figure 7.2: Training data accuracy logged every 5 batches.

batch 300. This is typical behaviour for a neural network model as it overfits the training data. In later training epochs, the training cost and learning rates are low, thus changes in weights are also low (see Section 3.5 on Backpropagation).

Validation accuracy was tested after each epoch. Under the assumption that the training, validation, and testing sets are uncorrelated, validation accuracy assesses the model's performance on new data. The resulting validation accuracies are shown in Figure 7.3. Validation accuracy increases rapidly within the first 10 epochs. The maximum validation accuracy occurred at epochs 18, 22, and 25, achieving an accuracy of 99.2%. The model was saved at epoch 18 to help minimise overfitting that may occur at later epochs. Applying this *best validation accuracy* model to the testing data set resulted in an accuracy of 99.4%.

Though overall accuracy is a strong indicator of model performance, some types of inaccuracy are more detrimental than others. In the identification of lacunes, it

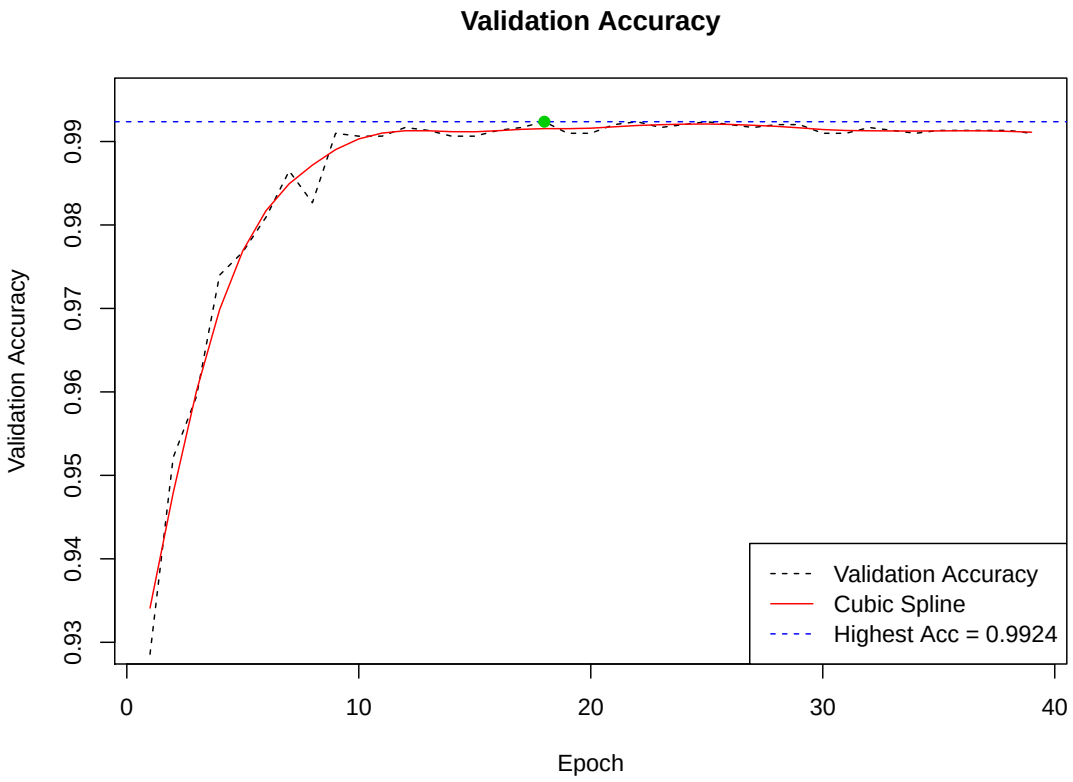

Figure 7.3: Validation data accuracy logged after each epoch. The highest validation accuracy occurs at epoch 18 (green).

is preferable to have a larger number of false-positive classifications than misclassify a true lacune. Hence, models will be compared by considering actual and predicted classification differences. Using this scheme, there are four outcomes: true-positive, true-negative, false-positive, and false-negative. These outcomes can be tabulated as a *confusion matrix*, shown in Table 7.1.

	Positive	Negative
True	1004	1863
False	16	2

Table 7.1: Confusion matrix of the Model 1 testing data set.

Model performance was evaluated using sensitivity, the proportion of correctly classified positive samples; and specificity, the proportion of correctly classified negative samples. The exhibited sensitivity and specificity of the model is 99.6% and 98.4% respectively.

Second model

The second model was built using all of the generated positive and negative samples described in Section 6.4. The resulting training accuracy is shown in Figure 7.4. The additional noise present in comparison to the previous model is a result of the larger sample size, increasing from 5,800 to 22,000 samples. Note that the batch size and number of training epochs remained at 128 and 40 respectively, resulting in a larger number of batches processed during training. Model training duration was 85 minutes. The behaviour of this model was similar to that of the previous model, with a steep increase in training accuracy in early epochs followed by consistent correct classifications. Training accuracy was consistently at 100% by batch 1,000.

The performance of the model on the validation set is given in Figure 7.5. Validation accuracy was maximised at epochs 28 and 34, achieving an accuracy of 99.8%. The model was saved at the earlier maximisation, epoch 28, to reduce overfitting.

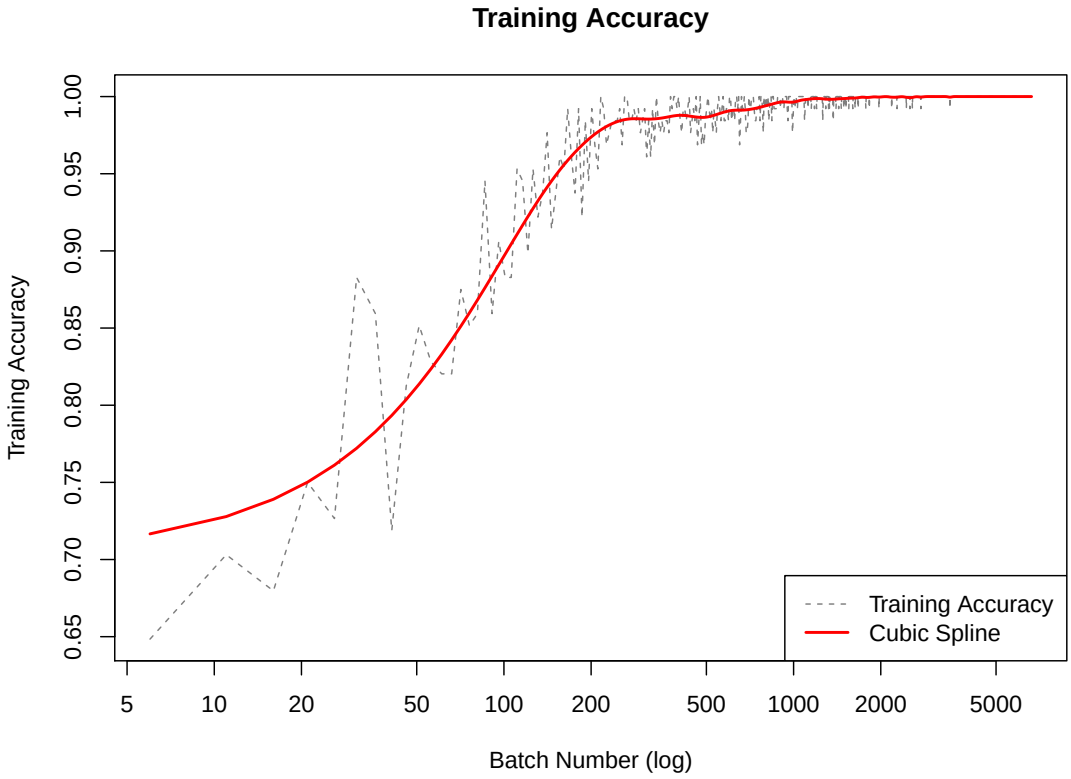

Figure 7.4: Training data accuracy logged every 5 batches.

Applying this best validation accuracy model to the test set achieved an accuracy of 99.8%. The confusion matrix, given by Table 7.2, yields a sensitivity of 99.9% and a specificity of 99.7%.

	Positive	Negative
True	961	9977
False	19	1

Table 7.2: Confusion matrix of the Model 2 testing data set.

Model comparisons

We now discuss the effect of the positive-negative sample ratio and sample size on model sensitivities and specificities. An outline of the models is given in Table 7.3. We observe that the second trained model has a higher sensitivity and specificity

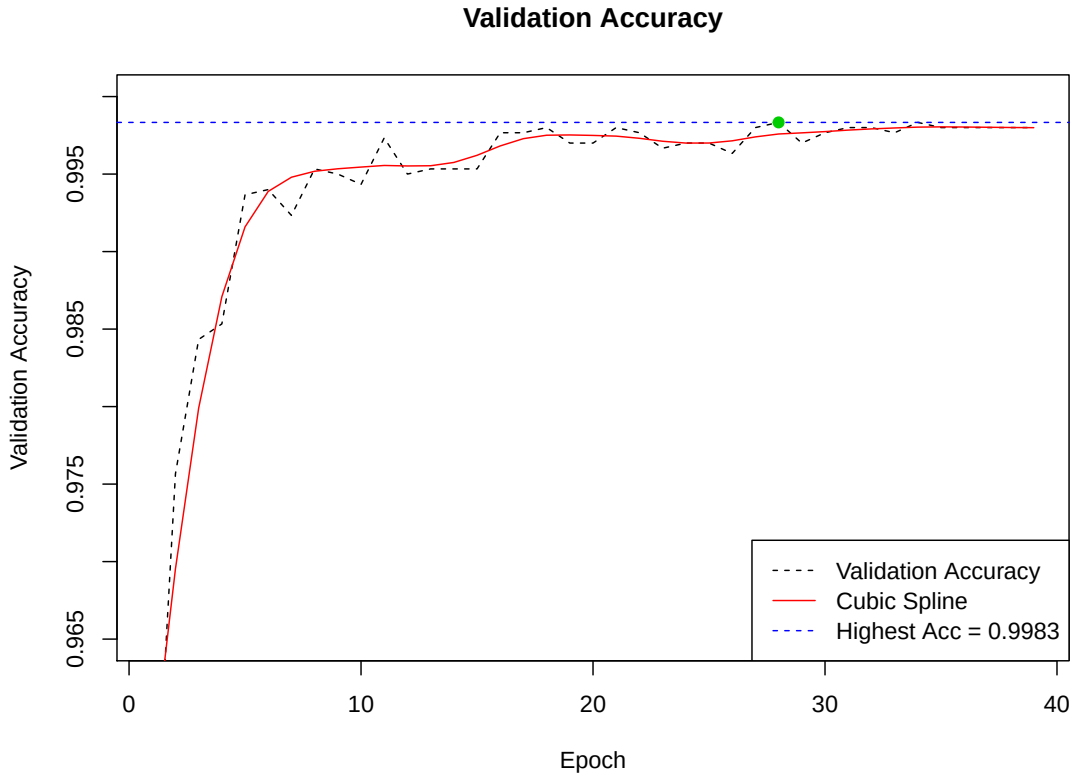

Figure 7.5: Validation data accuracy logged after each epoch

than that of the first model. To compare the models statistically, we conduct a hypothesis test on the differences in performance between the two models.

	Sensitivity	Specificity
Model 1	0.9980 (1004/1006)	0.9915 (1863/1879)
Model 2	0.9990 (961/962)	0.9981 (9977/9996)

Table 7.3: Sensitivities and specificities of both models.

First, we test the difference in sensitivity proportions. The Central Limit Theorem (CLT) is not applicable as the number of false-negatives is too small in both models. Instead, we use Fisher's Exact test. Let X_1 and X_2 be the number of lacunes correctly identified under the first and second models respectively. Let n_1

and n_2 be their respective sample sizes. Let $n = n_1 + n_2$ and $X = X_1 + X_2$. Under the null hypothesis, X_1 has a hypergeometric distribution with probability

$$P(X_1 = x_1 | X = x) = \frac{\binom{n_1}{x_1}\binom{n - n_1}{x - x_1}}{\binom{n}{x}}$$

A two-sided Fisher's Exact test was conducted on the number of positive samples correctly classified by each model, under the assumption that the two models have the same performance. The resulting p-value is 1, and we conclude that the two sensitivity proportions are not significantly different.

We now test the difference in specificity rates. We hypothesise that both sample proportions come from the same sampling distribution. We can apply a z-test since we have sufficiently large number of observations and the data are independent. Let X_1 and X_2 be the number of correctly classified negative samples from the first and second models respectively. Let n_1 and n_2 be the total number of negative samples, so that $\hat{p}_1 = X_1/n_1$ and $\hat{p}_2 = X_2/n_2$ are the observed specificities. The observed pooled specificity proportion is

$$\hat{p}_{pooled} = \frac{1863 + 9977}{1879 + 9996} = 0.9970526.$$

From the CLT,

$$Z = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{p_{pooled}(1 - p_{pooled})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}} \approx \mathcal{N}(0, 1).$$

The resulting test statistic is $z = -4.853$. The probability of the observing the given proportion difference under the null hypothesis is $P(Z < z) = 6.09 \times 10^{-7}$. This is well below the 5% significance level and we conclude that the specificity of the second model is higher than that of the first.

It has been shown that the second model did not have a significantly different sensitivity rate, but had an increased specificity. We hypothesise that this increase

was due to the larger sample size rather than the effect of the positive-negative ratio. The model was able to correctly classify a greater number of negative samples since it was exposed to a larger number of negatives during training.

The resulting sensitivity and specificity rates of the final model are very high. Hence the simplicity of the final model can be maintained by consolidating the model after the two-dimensional CNN. Ghafoorian et al. (2017) reduced the number of false-positives by training three parallel three-dimensional CNNs on candidate lacune samples. These outputs were concatenated with seven additional location-based variables into a fully-connected neural network. This process introduces a large number of extra weights, and training and usage time. Additionally, extra data processing is required to generate the three-dimensional samples and to retrieve the location-based variables.

Our model therefore removes the false-positive reduction component of the model by Ghafoorian et al. (2017) and achieves a final sensitivity rate of 99.9% and specificity rate of 99.8%.

The performance of a neural network model is difficult to justify as the final model has a complex interpretation. We now examine examples of correct and incorrect classifications to identify visual features that may have influenced model behaviour. Figure 7.6 demonstrates an example of a sample that was correctly classified as a lacune. True-positive samples generally consisted of a low or zero intensity in the T1-weighted image, and a dark region surrounded by a hyperintense rim in the FLAIR image. False-positive classifications, as shown in Figures 7.7 and 7.8, occurred with samples that were near low or zero intensity regions in the T1-weighted image, and also appeared close to a dark structure with a hyperintense rim in the FLAIR image. However, the hyperintense rims visible in these samples were not the result of lacunes, often occurring near the outer edge of the brain matter or adjacent to the cerebral ventricles. False-negative samples, shown in Figure 7.9, were rare and generally occurred at the edges of small lacunes that had little or no presence in the FLAIR imaging.

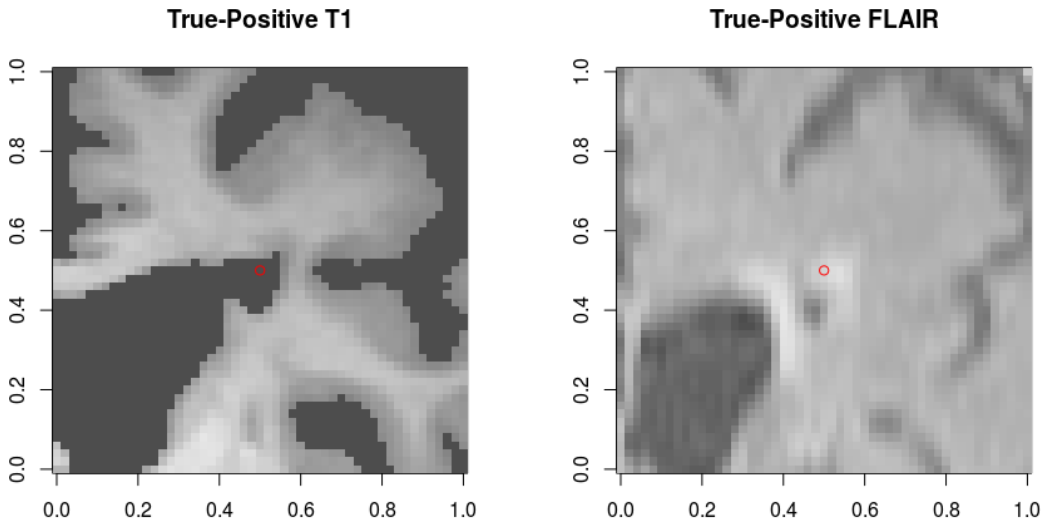

Figure 7.6: True-positive sample. The sample has a zero intensity in the T1-weighted image. A distinct dark region with hyperintense rim is visible in the FLAIR image.

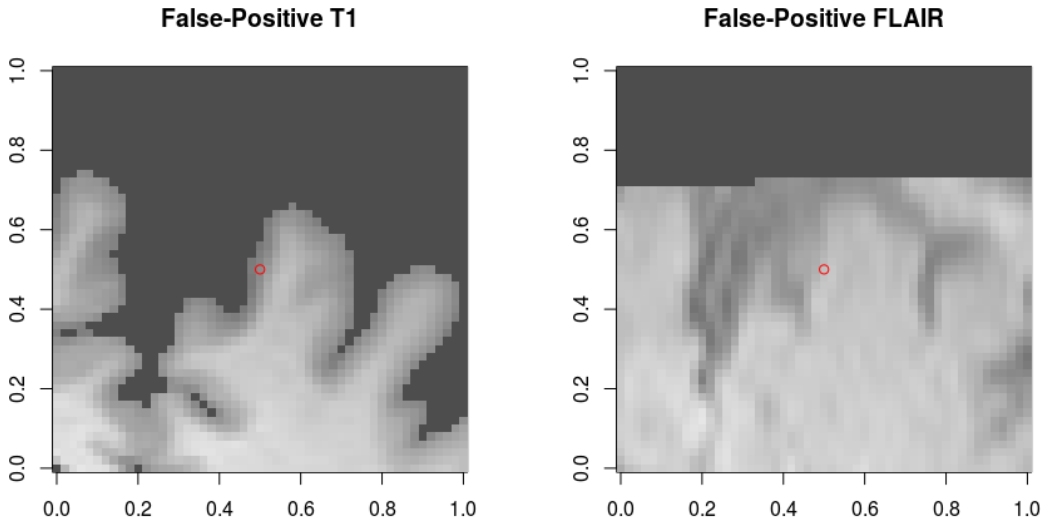

Figure 7.7: False-positive sample. The sample occurs near the edge of the brain matter and so has a low intensity in the T1-weighted scans. The false-positive classification may arise from the small amount of hyperintensity visible in the FLAIR image.

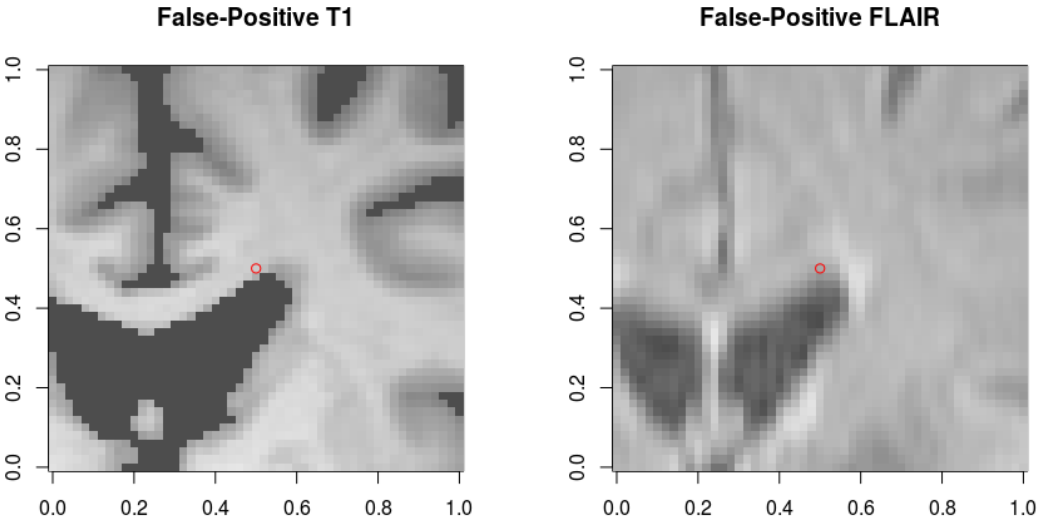

Figure 7.8: False-positive sample. The sample occurs next to the cerebral ventricles, which have a zero intensity in the T1-weighted image. A dark region with a hyperintense edge is visible in the FLAIR image.

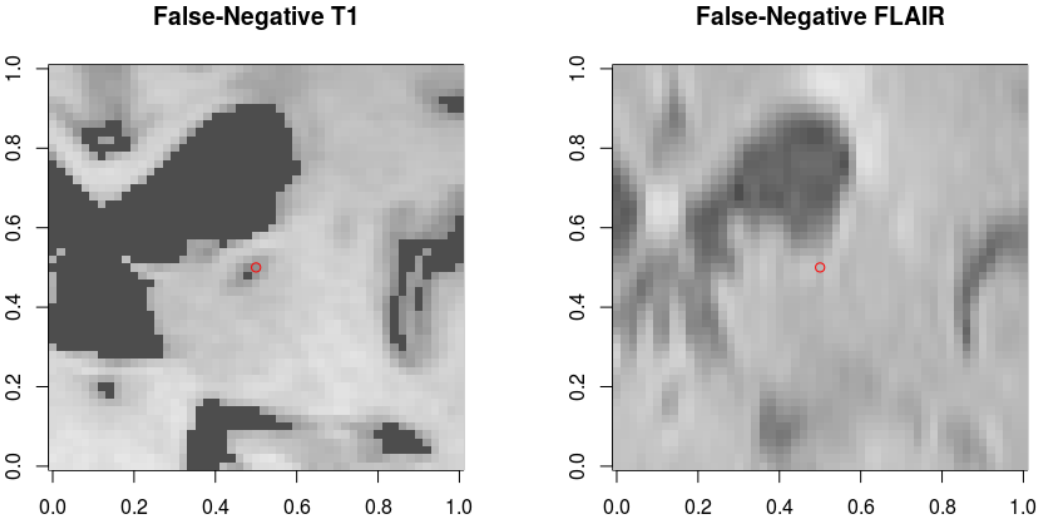

Figure 7.9: False-negative sample. The lacune has a diameter of only 3 mm, and is visible in the T1-weighted image. The FLAIR image has no visible dark region around the lacune or hyperintense rim.

Chapter 8

Effect of sample correlations

Observed performance of the model was augmented by the correlation of samples between training, validation, and testing data sets. Specifically, this dependence was constructed by data augmentation of the positive samples, which were distributed randomly amongst the data sets.

8.1 Updated model performance

Upon sampling correction, we trained the model on the updated data set. The first model was trained on all soft tissue extracted T1-weighted and FLAIR images, as described in Section 6.4. The resulting testing performance is shown in Table 8.1.

	Positive	Negative
True	206	6740
False	1196	164

Table 8.1: Confusion matrix of the first corrected model testing set.

The resulting sensitivity and specificity rates were respectively 55.7% and 84.9%. The model was able to detect some lacunes, however its efficacy was much lower than that of the existing models by Uchiyama et al. (2015) and Ghafoorian et al. (2017).

A second model was trained to examine the impact of information loss due to soft tissue extraction. Samples were generated similarly to the process described

in Section 6.4, replacing the soft tissue extracted T1-weighted images with original T1-weighted images. The resulting testing performance is shown in Table 8.2.

	Positive	Negative
True	319	7303
False	633	55

Table 8.2: Confusion matrix of the second corrected model testing set.

The resulting sensitivity and specificity rates were respectively 85.3% and 92.0%. This model exhibited a significant improvement in lacune detection, indicating the benefit of information retention surrounding lacunes.

Further improvements in sensitivity can be made by increasing the sample size. The corrected data set consisted of 40,000 samples, whereas the data set used by Ghafoorian et al. (2017) consisted of 3.2×10^5 samples. Increasing the number of lacunes observed during model training may improve sensitivity rates.

The model exhibits a large number of false-positives. These can be reduced through the inclusion of three-dimensional CNNs, increasing the amount of contextual information.

The realisation of sample correlations motivates a discussion on the false-positive rate reported by Ghafoorian et al. (2017). The T1-weighted scans have approximately 190 axial slices, thus an average of 0.13 false-positives per slice indicates approximately 25 false-positives per volume. The final model described in Section 7.3 exhibited a sensitivity of 99.8%, which would output an average of 100 false-positives across 50,000 brain tissue sample inputs. The specificity reported by Ghafoorian et al. (2017) is hence lower than that of the advantaged model. This performance comes into question as the quality of the responses being estimated is bounded by the quality of lacune detection by clinicians, and also the subjectivity surrounding pixels at the border of lacunes. Each voxel of the T1-weighted scans is a $1 \times 1 \times 1$ mm cube, hence the lacunes do not fill each voxel perfectly. We hypothesise that initial rating inconsistencies and subjectivity at the borders of lacunes should

affect response quality and increase the number of false-positives. This motivates us to question the false-positive reduction statistic reported by Ghafoorian et al. (2017) and conclude that replication of the model on new data is required.

Chapter 9

Conclusion

The detection of lacunes in MRI is necessary for neurological study surrounding cerebral small vessel disease and the incidence of stroke. Current identification methods are time consuming and largely inconsistent, resulting in potentially invalid research conclusions and misdiagnosis for those with a heightened risk of stroke. Lacune detection models exist and are successful in classifying lacunes, however are reliant on location-based variables. These variables must be estimated using existing tools to segment the brain structure, however these tools are not necessarily available to clinicians.

We proposed a location-independent two-dimensional convolutional neural network model for the detection of lacunes in MRI. Our proposed model adapts the lacune detection model by Ghafoorian et al. (2017) and makes three major changes. The first is the removal of dependence on location-based variables. This reduces the quantity and complexity of data pre-processing, and ensures that samples are classified correctly even if they occur in less frequently observed regions.

The second adaptation is use of soft tissue extraction on the T1-weighted scans prior to model input. This process occurs to remove identification features visible in MRI, and has the additional effect of enlarging spaces that appear similar to CSF, and hence enlarges lacunes.

The third change is the removal of dimension reduction prior to the first convolutional layer. Early reduction of image dimension results in loss of pixel information that may contain lacunes. Maintaining the dimension of the input image ensures

that vital pixel information is retained early in the model, which aids in the identification of smaller lacunes.

The final simplified lacune detection model on correlated data exhibited a testing sensitivity and specificity of 99.9% and 99.8% respectively. The model was consolidated after candidate generation to maintain model simplicity, and reduce computation time and data pre-processing.

With uncorrelated data, an identical model structure exhibited a testing sensitivity and specificity of respectively 55.7% and 84.9%. Training the model on T1-weighted data without soft tissue extraction increased the sensitivity and specificity rates respectively to 85.3% and 92.0%.

9.1 Improvements and further research

Under the assumption of uncorrelated data sets, the original model exhibits a very high detection performance, however the incidence of remaining false-positive errors requires clinicians to supervise the model outputs. Each positive classification must be examined by clinicians to reduce the number of false-positives. The high specificity rate helps to restrict this number, however the dimensions of each MRI scan are high, involving approximately 50,000 voxels within the brain tissue. Hence, further model improvement can be made to reduce the number of false-positives.

Information inherent in two-dimensional sample subimages may not contain enough context to make an accurate classification for some samples. This is particularly the case with samples that contain blood vessels or perivascular spaces that appear circular from the axial perspective. The amount of context included in the sample can be increased by considering three-dimensional samples and employing a three-dimensional CNN. The inclusion of volume based samples ensures that a distinction can be made between structures that only appear circular in the two-dimensional patches, and samples that are truly spherical. It should be noted that three-dimensional CNNs require a large number of weights and hence have lengthy computation time if applied to all points in a volume. Implementation time can be

improved by applying the three-dimensional CNN to the candidate lacunes indicated by the initial two-dimensional CNN as a form of false-positive reduction.

Soft tissue extraction applied to the T1-weighted scans resulted in very distinct edges between the brain matter and surrounding structures. This resulted in the masking of voxels that exhibited a high probability of CSF content, which includes lacunes. Additional research could be conducted on the impact of soft tissue extraction on lacune detection. If found to be detrimental to model performance, it may be beneficial to implement lacune detection on T1-weighted scans prior to soft tissue extraction.

Once the sensitivity and specificity rates of the correlation corrected model is improved, the algorithm is to be included as part of the Sydney MAS MRI data processing pipeline. Inclusion of an improved model will assist neuroscientists in the efficient and consistent detection of lacunes, and thus ensure the validity of medical studies conducted. Additionally, the inclusion of a model during medical practice would provide accurate detection of lacunes in the MRI of individual patients, indicating any heightened risk of stroke. The awareness of any increased risk could then result in appropriate treatment and possible avoidance of stroke.

Recent attention has been drawn to other forms of automated medical diagnosis. Algorithms have been found successful in the detection of breast cancer (Liu et al., 2018) and bone fractures (Lindsey et al., 2018). The structure of data in these contexts is similar to the three-dimensional array structure of MRI, hence the CNN techniques applied can be extended to other medical classification and diagnostic tasks.

Bibliography

Bean, B. (2014). The abcs of mri. Retrieved from http://www.24x7mag.com. Accessed 22 September 2018.

Benjamin, P., Trippier, S., Lawrence, A., Lambert, C., Zeestraten, E., Williams, O., Patel, B., Morris, R., Barrick, T., Mackinnon, A., and Markus, H. (2018). Lacunar infarcts, but not perivascular spaces, are predictors of cognitive decline in cerebral small-vessel disease. *Stroke*.

Bitar, R., Moody, A. R., Sarrazin, J., McGregor, C., Christakis, M., Symons, S., Roberts, T. P., Leung, G., Perng, R., Tadros, S., and Nelson, A. (2006). Mr pulse sequences: What every radiologist wants to know but is afraid to ask. *Radiographics*, 26(2):513–537.

Dafny, M. (1997). Overview of the nervous system. Retrieved from https://nba.uth.tmc.edu/neuroscience. Accessed 27 June 2018.

de Leeuw, F.-E., de Groot, J. C., Achten, E., Oudkerk, M., Ramos, L. M. P., Heijboer, R., Hofman, A., Jolles, J., van Gijn, J., and Breteler, M. M. B. (2001). Prevalence of cerebral white matter lesions in elderly people: a population based magnetic resonance imaging study. the rotterdam scan study. *Journal of Neurology, Neurosurgery & Psychiatry*, 70(1).

Deshpande, A. (2016). A beginner's guide to understanding convolutional neural networks. Retrieved from https://adeshpande3.github.io. Accessed 25 August 2018.

Dou, Q., Chen, H., Yu, L., Zhao, L., Qin, J., Wang, D., Mok, V., Shi, L., and Heng, P.-A. (2016). Automatic detection of cerebral microbleeds from mr images

via 3d convolutional neural networks. *IEEE Transactions on Medical Imaging*, 35(5):1182–1195.

Folly, K. A. and Venayagamoorthy, G. K. (2009). Effects of learning rate on the performance of the population based incremental learning algorithm. *2009 International Joint Conference on Neural Networks*, pages 861–868.

for Human Neuroimaging, W. C. (2017). Statistical parametric mapping. Release 7219.

Ghafoorian, M., Karssemeijer, N., Heskes, T., Bergkamp, M., Wissink, J., Obels, J., Keizer, K., Leeuw, F.-E., Ginneken, B., Marchiori, E., and Platel, B. (2017). Deep multi-scale location-aware 3d convolutional neural networks for automated detection of lacunes of presumed vascular origin. *NeuroImage: Clinical*, 14:391–399.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. Retrieved from http://www.deeplearningbook.org. Accessed 17 August 2018.

Gouw, A. A., Seewann, A., van Der Flier, W. M., Barkhof, F., Rozemuller, A. M., Scheltens, P., and Geurts, J. J. G. (2011). Heterogeneity of small vessel disease: a systematic review of mri and histopathology correlations. *Journal of Neurology, Neurosurgery and Psychiatry*, 82(2).

Group, F. A. (2007). Fslview. 3.2.0.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv*.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *ArXiv*.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 2:1097–1105.

LeCun, Y., Bottou, L., Orr, G., and Müller, K.-R. (2012). Efficient backprop. *Neural Networks: Tricks of the Trade*, pages 9–48.

Lindsey, R., Daluiski, A., Chopra, S., Lachapelle, A., Mozer, M., Sicular, S., Hanel, D., Gardner, M., Gupta, A., Hotchkiss, R., and Potter, H. (2018). Deep neural network improves fracture detection by clinicians. *Proceedings of the National Academy of Sciences*.

Liu, Y., Kohlberger, T., Norouzi, M., Dahl, G., Smith, J., Mohtashamian, A., Olson, N., Peng, L., Hipp, J., and Stumpe, M. (2018). Artificial intelligence-based breast cancer nodal metastasis detection. *Archives of pathology & laboratory medicine*.

Maas, A., Hannun, A., and Ng, A. (2013). Rectifier nonlinearities improve neural network acoustic models. *International conference on machine learning*.

Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2):431–441.

Mishkin, D. and Matas, J. (2016). All you need is a good init. *ArXiv*.

Nielson, M. A. (2015). Neural networks and deep learning. Retrieved from http://neuralnetworksanddeeplearning.com. Accessed 20 June 2018.

Norrving, B. (2008). Lacunar infarcts: no black holes in the brain are benign. *Practical neurology*, 8(4).

Perez, L. and Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *ArXiv*.

Potter, G., Chappell, F., Morris, Z., and Wardlaw, J. (2015). Cerebral perivascular spaces visible on magnetic resonance imaging: Development of a qualitative rating scale and its observer reliability. *Cerebrovascular Diseases*, 39(4):224–231.

Prechelt, L. (2012). Early stopping - but when? *Neural Networks: Tricks of the Trade*, pages 53–67.

Preston, D. C. (2006). Magnetic Resonance Imaging (MRI) of the Brain and Spine: Basics. Retrieved from http://casemed.case.edu/clerkships/neurology. Accessed 16 July 2018.

Rinck, P. (2013). *Magnetic Resonance in Medicine. The Basic Textbook of the European Magnetic Resonance Forum.* 7th ed. edition.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Szegedy, C., Wei Liu, P., Yangqing Jia, S., Sermanet, D., Reed, D., Anguelov, V., Erhan, A., Vanhoucke, A., and Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

Uchiyama, Y., Abe, A., Muramatsu, C., Hara, T., Shiraishi, J., and Fujita, H. (2015). Eigenspace template matching for detection of lacunar infarcts on mr images. *Journal of Digital Imaging*, 28(1):116–122.

Uchiyama, Y., Yokoyama, R., Ando, H., Asano, T., Kato, H., Yamakawa, H., Yamakawa, H., Hara, T., Iwama, T., Hoshi, H., and Fujita, H. (2007a). Computer-aided diagnosis scheme for detection of lacunar infarcts on mr images. *Academic Radiology*, 14(12):1554–1561.

Uchiyama, Y., Yokoyama, R., Ando, H., Asano, T., Kato, H., Yamakawa, H., Yamakawa, H., Hara, T., Iwama, T., Hoshi, H., and Fujita, H. (2007b). Improvement of automated detection method of lacunar infarcts in brain mr images. *Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 1599–1602.

Uchiyama, Y., Yokoyama, R., Ando, H., Asano, T., Kato, H., Yamakawa, H., Yamakawa, H., Hara, T., Iwama, T., Hoshi, H., and Fujita, H. (2007c). Improvement of automated detection method of lacunar infarcts in brain mr images. *Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society.*

van den Heuvel, T., van der Eerden, A., Manniesing, R., Ghafoorian, M., Tan, T., Andriessen, T., Vyvere, T., van den Hauwe, L., Romeny, B., Goraj, B., and Platel, B. (2016). Automated detection of cerebral microbleeds in patients with traumatic brain injury. *Neuroimage-Clinical*, 12:241–251.

van der Flier, W., van Straaten, E., Barkhof, F., Verdelho, A., Madureira, S., Pantoni, L., Inzitari, D., Erkinjuntti, T., Crisby, M., Waldemar, G., Schmidt, R., Fazekas, F., and Scheltens, P. (2005). Small vessel disease and general cognitive function in nondisabled elderly: The ladis study. *Stroke*, 36(10):2116–2120.

Wardlaw, J., Smith, C., and Dichgans, M. (2013a). Mechanisms of sporadic cerebral small vessel disease: insights from neuroimaging. *Lancet Neurology*, 12(5):483–497.

Wardlaw, J., Smith, E., Biessels, G., Cordonnier, C., Fazekas, F., Frayne, R., Lindley, R., O'Brien, J., Barkhof, F., Benavente, O., Black, S., Brayne, C., Breteler, M., Chabriat, H., DeCarli, C., de Leeuw, F.-E., Doubal, F., Duering, M., Fox, N., Greenberg, S., Hachinski, V., Kilimann, I., Mok, V., Oostenbrugge, R., Pantoni, L., Speck, O., Stephan, B., Teipel, S., Viswanathan, A., Werring, D., Chen, C., Smith, C., van Buchem, M., Norrving, B., Gorelick, P., and Dichgans, M. (2013b). Neuroimaging standards for research into small vessel disease and its contribution to ageing and neurodegeneration. *The Lancet Neurology*, 12(8):822–838.

Yokoyama, R., Zhang, X., Uchiyama, Y., Fujita, H., Hara, T., Zhou, X., Kanematsu, M., Asano, T., Kondo, H., Goshima, S., Hoshi, H., and Iwama, T. (2007). Development of an automated method for the detection of chronic lacunar infarct regions in brain mr images. *IEICE Transactions*, pages 943–954.

Appendices

Appendix A

R code for candidate lacune detection model

```
library(tensorflow)
library(crayon)

# Data placeholders
# Num of samples. 51x51 = 2601. x2 channels = 5202
x <- tf$placeholder(tf$float32, shape(NULL, 5202L))
# Num of samples, 2 outcomes
y_ <- tf$placeholder(tf$float32, shape(NULL, 2L))

# Weights initialised by He Method
he.init <- tf$contrib$layers$
  variance_scaling_initializer(factor = 2.0,
                               mode = "FAN_AVG",
                               uniform = FALSE)
# Variable initialiser functions
weight.variable <- function(shape) {
  initial <- he.init(shape)
  tf$Variable(initial)
}

beta.variable <- function(shape) {
```

```
  tf$Variable(tf$zeros(shape))
}

scale.variable <- function(shape) {
  tf$Variable(tf$ones(shape))
}

conv2d <- function(x, W) {
  tf$nn$conv2d(x, W, strides = c(1, 1, 1, 1), padding = "VALID")
}

# Batch Normlisation
batch.norm <- function(z, beta, scale) {
  moments <- tf$nn$moments(z, 0L)
  tf$nn$batch_normalization(z, moments[[1]], moments[[2]]^2,
                            beta,scale, 1e-3)
}


# Dropout of 0.3 on fully connected layers (avoids overfitting)
keep.prob <- tf$placeholder(tf$float32)


# Reshape x into sample size of 51x51
# 2 colour channels (FLAIR & T1)
x.image <- tf$reshape(x, shape(-1L, 51L, 51L, 2L))


# Conv Layers -------------------------------------------------
```

```
# conv 1: 20 filters, 7x7 size
W.conv1 <- weight.variable(shape(7L, 7L, 2L, 20L))
z.conv1 <- conv2d(x.image, W.conv1)
# Batch normalisation
beta.conv1 <- beta.variable(shape(20L))
scale.conv1 <- scale.variable(shape(20L))
bn.conv1 <- batch.norm(z.conv1, beta.conv1, scale.conv1)
a.conv1 <- tf$nn$relu(bn.conv1)

# pool: 2x2 size, 2 stride
a.pool1 <- tf$nn$max_pool(a.conv1, ksize = c(1L,2L,2L,1L),
                          strides = c(1L,2L,2L,1L),
                          padding = "VALID")

# conv 2: 40 filters, 5x5 size
W.conv2 <- weight.variable(shape(5L, 5L, 20L, 40L))
z.conv2 <- conv2d(a.pool1, W.conv2)
beta.conv2 <- beta.variable(shape(40L))
scale.conv2 <- scale.variable(shape(40L))
bn.conv2 <- batch.norm(z.conv2, beta.conv2, scale.conv2)
a.conv2 <- tf$nn$relu(bn.conv2)

# conv 3: 80 filters, 3x3 size
W.conv3 <- weight.variable(shape(3L, 3L, 40L, 80L))
z.conv3 <- conv2d(a.conv2, W.conv3)
beta.conv3 <- beta.variable(shape(80L))
scale.conv3 <- scale.variable(shape(80L))
bn.conv3 <- batch.norm(z.conv3, beta.conv3, scale.conv3)
```

```
a.conv3 <- tf$nn$relu(bn.conv3)

# conv 4: 110 filters, 3x3 size
W.conv4 <- weight.variable(shape(3L, 3L, 80L, 110L))
z.conv4 <- conv2d(a.conv3, W.conv4)
beta.conv4 <- beta.variable(shape(110L))
scale.conv4 <- scale.variable(shape(110L))
bn.conv4 <- batch.norm(z.conv4, beta.conv4, scale.conv4)
a.conv4 <- tf$nn$relu(bn.conv4)



# Fully Connected Layers ---------------------------------------

a.conv4.flat <- tf$reshape(a.conv4, shape(-1L, 14L*14L*110L))
# full 1: 300 size
W.fcl1 <- weight.variable(shape(14L*14L*110L, 300L))
z.fcl1 <- tf$matmul(a.conv4.flat, W.fcl1)
beta.fcl1 <- beta.variable(shape(300L))
scale.fcl1 <- scale.variable(shape(300L))
bn.fcl1 <- batch.norm(z.fcl1, beta.fcl1, scale.fcl1)
a.fcl1 <- tf$nn$relu(bn.fcl1)
a.fcl1.drop <- tf$nn$dropout(a.fcl1, keep.prob)

# full 2: 200 size
W.fcl2 <- weight.variable(shape(300L, 200L))
z.fcl2 <- tf$matmul(a.fcl1.drop, W.fcl2)
beta.fcl2 <- beta.variable(shape(200L))
scale.fcl2 <- scale.variable(shape(200L))
bn.fcl2 <- batch.norm(z.fcl2, beta.fcl2, scale.fcl2)
```

```
a.fcl2 <- tf$nn$relu(bn.fcl2)
a.fcl2.drop <- tf$nn$dropout(a.fcl2, keep.prob)

# full 3: 2 size
W.fcl3 <- weight.variable(shape(200L, 2L))
z.fcl3 <- tf$matmul(a.fcl2.drop, W.fcl3)
beta.fcl3 <- beta.variable(shape(2L))
scale.fcl3 <- scale.variable(shape(2L))
bn.fcl3 <- batch.norm(z.fcl3, beta.fcl3, scale.fcl3)

# Softmax classifier
y <- tf$nn$softmax(bn.fcl3)



# Training ----------------------------------------------------

# Cross entropy loss
# L2 regularisation with lambda_2 = 0.0001
cross.entropy <- tf$reduce_mean(
  -tf$reduce_sum(y_ * tf$log(y + 1e-10),reduction_indices = 1L))
l2.reg <- cross.entropy + 0.0001 * tf$reduce_sum(W.fcl3^2)

# Stochastic gradient descent (Adam optimiser)
learn.rate <- tf$placeholder(tf$float32)
train.step <- tf$train$AdamOptimizer(learn.rate)$minimize(l2.reg)

# Calculate prediction accuracy
correct.prediction <- tf$equal(tf$argmax(y, 1L),
                               tf$argmax(y_, 1L))
```

```
accuracy <- tf$reduce_mean(tf$cast(correct.prediction,
                                   tf$float32))

# Model saver
saver <- tf$train$Saver()

# Tensorflow session
sess <- tf$InteractiveSession()
sess$run(tf$global_variables_initializer())

# Stochastic gradient descent: batches of 128
num.samples <- nrow(training)
max.epochs <- 40
# Decaying learning rate. 5e-4 reduced to 1e-6
learning.rates <- seq(5e-4, 1e-6, length.out = max.epochs)

# Store training and validation accuracies
train.accuracy <- numeric(max.epochs*num.samples)
i.train.acc <- 1
valid.accuracy <- numeric(max.epochs)
i.valid.acc <- 1

best.accuracy <- 0
e <- 1
# Model training loop
while (e < max.epochs) {
  stoch.training <- training[sample(nrow(training)),]
  for (i in seq(1, num.samples-128, by = 128)) {
    # Run training step
```

```
  train.step$run(feed_dict = dict(
    x = stoch.training[i:(i+127), 5:5206],
    y_ = stoch.training[i:(i+127), 5207:5208],
    keep.prob = 0.7, learn.rate = learning.rates[e]))

  # Report training accuracy every 5 batches
  if (i %% 5 == 0) {
    train.accuracy[i.train.acc] <- accuracy$eval(
      feed_dict = dict(
      x = training[i:(i+127), 5:5206],
      y_ = training[i:(i+127), 5207:5208],
      keep.prob = 1.0, learn.rate = learning.rates[e]))
    cat(sprintf(" Acc: %g", train.accuracy[i.train.acc]))
    i.train.acc <- i.train.acc + 1
    plot(train.accuracy[max(i.train.acc-500,1):i.train.acc])
  }
  cat("\n")
}
# Report validation accuracy
n.valid <- 3000
valid.accuracy[i.valid.acc] <- accuracy$eval(feed_dict = dict(
  x = validation[1:n.valid,5:5206],
  y_ = validation[1:n.valid,5207:5208],
  keep.prob = 1.0, learn.rate = learning.rates[e]))
cat(sprintf("epoch %d, validation accuracy %g\n",
            e, valid.accuracy[i.valid.acc]))

# Early stopping - highest accuracy on validation set
if(valid.accuracy[i.valid.acc] > best.accuracy) {
```

```
    cat("Saving Model..\n")
    best.accuracy <- valid.accuracy[i.valid.acc]
    saver$save(sess, "/srv/scratch/z5016924/model.ckpt")
  }
  i.valid.acc <- i.valid.acc + 1

  e <- e + 1
}
```