



**Department of  
Electronic and  
Computer Engineering**

# Machine Learning Using Python Frameworks

Lorcan Williamson, 16160703

Supervisor: Dr. K. Murphy

Course: B.Eng Electronic and Computer Engineering

March 8, 2020

## **Abstract**

Machine learning and deep learning have advance rapidly in recent years due to the increase computing power and availability. Python is a popular language choice for machine learning tasks, with a number of frameworks available for the development of machine and deep learning models. This project aims to look at how code is developed using these frameworks, and compare their functionality to those in other languages.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Problem 1: Predicting Student test Scores</b>	<b>7</b>
2.1	Problem Background . . . . .	7
2.2	Dataset . . . . .	7
2.3	Implementations . . . . .	8
2.3.1	Neural Network . . . . .	9
2.3.2	Support Vector Machine . . . . .	13
2.3.3	Decision Tree . . . . .	18
2.3.4	Random Forest . . . . .	22
2.3.5	Comparison of Results . . . . .	23
<b>3</b>	<b>Problem 2: Detecting Pneumonia</b>	<b>26</b>
3.1	Problem Overview . . . . .	26
3.2	Convolutional Neural Networks . . . . .	27
3.2.1	Convolutional Layers . . . . .	28
3.2.2	Pooling Layers . . . . .	29
3.2.3	Normalization Layers . . . . .	29
3.3	Dataset . . . . .	29
3.4	Explored Solutions . . . . .	31
3.4.1	Simple CNN . . . . .	31
3.4.2	Image Augmentation . . . . .	33
3.4.3	Transfer learning . . . . .	35
3.5	Results . . . . .	38
<b>4</b>	<b>Results and Conclusions</b>	<b>40</b>
4.1	Results . . . . .	40

	3
4.2 Future work . . . . .	40
<b>APPENDICES</b>	<b>40</b>
<b>A Full Decision Trees</b>	<b>41</b>
<b>B Example Problem 3 outline</b>	<b>44</b>

# List of abbreviations

<b>ANN</b>	Artificial Neural Network
<b>BFGS</b>	Broyden-Fletcher-Goldfarb-Shanno algorithm
<b>CAD</b>	Computer aided diagnosis
<b>CNN</b>	Convolutional Neural Network
<b>DT</b>	Decision Tree
<b>MLP</b>	Multi-layered Perceptron
<b>MSE</b>	Mean Squared Error
<b>NN</b>	Neural Network
<b>PCA</b>	Principle Component Analysis
<b>RF</b>	Random Forest
<b>ReLu</b>	Rectified Linear Unit
<b>RNN</b>	Recurrent Neural Network
<b>SVM</b>	Support Vector Machine

# 1. Introduction

This project was undertaken to look at the machine and deep learning frameworks available in Python and the code development process used to build and train various machine learning models using them. To this end 3 of the main machine learning frameworks in Python were looked at: Scikit-learn[1] a machine learning framework; and TensorFlow[2], which focuses more on neural networks. Both the use of base TensorFlow was examined, as well as its use through Keras[3], a high level API designed for deep learning that can use TensorFlow or Theano, another machine learning framework, as the back-end. For this project TensorFlow was used as the back-end. Initially 3 example problems were to be looked at to demonstrate the various features of these frameworks, but this was changed to 2 afterwards due to time constraints. These two problems being:

- **Predicting student test performance:** The goal of this example was to predict the test scores of students based on a number of factors. This problem was chosen as a variety of common machine learning models such as Support Vector Machines (SVM), Decision Trees (DT), Random Forests (RF), and Neural Networks (NN) would be suitable for it. These models were chosen based on the work done in the paper this data was originally collected for, *Using Data Mining To Predict Secondary School Student Performance*[4]. The models for this were developed using R, so this also allows for a comparison between Python and R ML frameworks. However a number of addition techniques were used to try improve the previous results and to demonstrate some of the features of the Python frameworks.
- **Detecting pneumonia in chest x-rays:** The goal of the second example was to detect whether pneumonia was present in a chest x-ray or not. This would allow for a look at Convolutional Neural Networks, a deep learning model that has only really become common in the last decade due to the computational power and amount of data required to train them. These models excel at image recognition tasks, and thus would not be best suited to the student performance example problem. The data for this problem came from

*Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification*[5].

The third problem was to have been a forecasting problem to demonstrate how Long-Term-Short-Memory (LSTM) networks can be implemented and trained using Keras, and to introduce a regression problem along side the two classification problems. The goal of it would have been to forecast the trading prices of Bitcoin, using historic data.

Code for this project was developed using the following frameworks and technologies:

- Python (version 3.6)
- Scikit-learn (version 0.21.3)
- Keras (version 2.2.4 (TensorFlow version 1.13.1 backend))
- AWS EC2 instance (Hardware instance: p2.xlarge, OS Image: Deep Learning AMI (Ubuntu) Version 24.1)

## 2. Problem 1: Predicting Student test Scores

### 2.1 Problem Background

The goal of this problem was to look at how various machine learning models are implemented using Python frameworks. For this problem 4 types of ML models were looked at: SVM; DT; RF; and NN. These were implemented using the Scikit-learn and Tensorflow frameworks. These models were chosen as they were used in the original paper looking at this dataset[4], and they are some of the most commonly used types of machine learning models. The original paper used R to develop their models. By using similar models and methodology a comparison can be made between the functionality of the python frameworks and those in R using the RMiner framework.

### 2.2 Dataset

The dataset used consists of 2 csv files corresponding to the two subjects data was collected for, Mathematics and Portuguese, from Portuguese students. Both files have 33 columns, explained in Table 2.1, corresponding to the 32 features of the dataset, and 1 label. Each entry in the files corresponds to a student, some students appear in both files. There are 649 students represented in the Portuguese data, and 396 students in the Mathematics file. For this reason the Portuguese subject data was used for training and testing the models. Before the data was used for training it had to be preprocessed. All non-numeric features, such as *school* and *Mjob* were converted to numeric using a one-hot-label encoding. This preprocessing was done using Scikit-learn's `sklearn.preprocessing.LabelEncoder`. In Paulo Cortez and Alice Silva's original paper[4] they looked at three different prediction metrics: binary pass/fail classification; 5 grade classification using the Erasmus grading scale; and regression using the 20 grade levels in the Portuguese education system. For this project only the 5 level classification was looked at. This meant that the 20 grades in the original data needed to be



converted to the 5 Erasmus grades, using the following mapping:  $16 - 20 \rightarrow 5(A)$ ,  $14 - 15 \rightarrow 4(B)$ ,  $12 - 13 \rightarrow 3(C)$ ,  $10 - 11 \rightarrow 2(D)$ ,  $0 - 9 \rightarrow 1(F)$

Attribute	Description (Domain)
sex	student's sex (binary: female or male)
age	student's age (numeric: from 15 to 22)
school	student's school (binary: <i>Gabriel Pereira</i> or <i>Mousinho da Silveira</i> )
address	student's home address type (binary: urban or rural)
Pstatus	parent's cohabitation status (binary: living together or apart)
Medu	mother's education (numeric: from 0 to 4 <sup>a</sup> )
Mjob	mother's job (nominal <sup>b</sup> )
Fedu	father's education (numeric: from 0 to 4 <sup>a</sup> )
Fjob	father's job (nominal <sup>b</sup> )
guardian	student's guardian (nominal: mother, father or other)
famsize	family size (binary: $\leq 3$ or $> 3$ )
famrel	quality of family relationships (numeric: from 1 – very bad to 5 – excellent)
reason	reason to choose this school (nominal: close to home, school reputation, course preference or other)
traveltime	home to school travel time (numeric: 1 – $< 15$ min., 2 – 15 to 30 min., 3 – 30 min. to 1 hour or 4 – $> 1$ hour).
studytime	weekly study time (numeric: 1 – $< 2$ hours, 2 – 2 to 5 hours, 3 – 5 to 10 hours or 4 – $> 10$ hours)
failures	number of past class failures (numeric: $n$ if $1 \leq n < 3$ , else 4)
schoolsup	extra educational school support (binary: yes or no)
famsup	family educational support (binary: yes or no)
activities	extra-curricular activities (binary: yes or no)
paidclass	extra paid classes (binary: yes or no)
internet	Internet access at home (binary: yes or no)
nursery	attended nursery school (binary: yes or no)
higher	wants to take higher education (binary: yes or no)
romantic	with a romantic relationship (binary: yes or no)
freetime	free time after school (numeric: from 1 – very low to 5 – very high)
goout	going out with friends (numeric: from 1 – very low to 5 – very high)
Walc	weekend alcohol consumption (numeric: from 1 – very low to 5 – very high)
Dalc	workday alcohol consumption (numeric: from 1 – very low to 5 – very high)
health	current health status (numeric: from 1 – very bad to 5 – very good)
absences	number of school absences (numeric: from 0 to 93)
G1	first period grade (numeric: from 0 to 20)
G2	second period grade (numeric: from 0 to 20)
G3	final grade (numeric: from 0 to 20)

*a* 0 – none, 1 – primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education.

*b* teacher, health care related, civil services (e.g. administrative or police), at home or other.

Table 2.1: Student data available

## 2.3 Implementations

For this problem the methodology of Paulo Cortez and Alice Silva's original paper was followed, and then expanded upon to try improve the results. In the original paper 4 machine learning models were looked at; SVM, DT, RF, and NN. All these models were trained using

10-fold cross-validation, where the average validation accuracy was taken as the test accuracy. This was repeated 20 times for each model and the average accuracy and standard deviation was recorded. For this project the same 4 models were looked at, and trained with 10-fold cross-validation, but this was only done once for each model, and the average validation accuracy was compared to the expected. New models were then trained using data with extra processing, or with different hyper-parameters to try achieve better accuracy. The best models of each type were also tested on the maths dataset to see if they could perform well on unseen data from a different school subject.

### 2.3.1 Neural Network

#### Overview

A neural network is a machine made up of many artificial neurons. The basic model for an artificial neuron was described by McCulloch and Pitts in 1943[6], Figure 2.1. The McCulloch-Pitts neuron works by taking the weighted sum of the inputs, and passing this through some activation function which produces an activation level for the neuron. A number of functions can be used for the activation, though rectified linear has become a common standard in recent years[7].

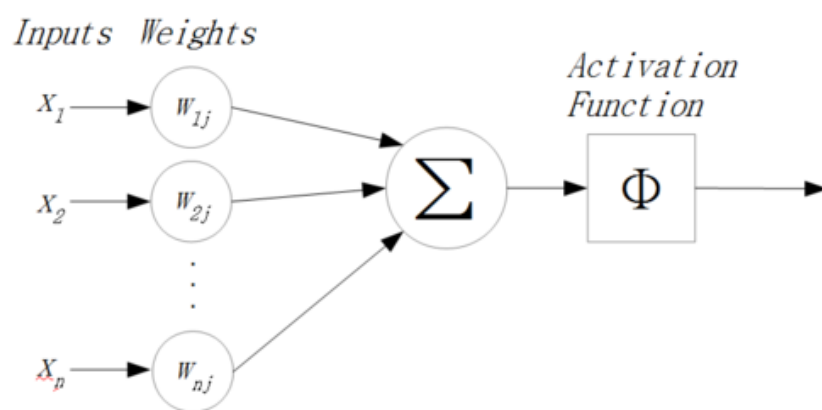


Figure 2.1: The McCulloch-Pitts neuron

The architecture of the model described in the paper was that of a multi-layer perceptron (MLP).

This is a neural network made up of only dense layers[8]. The MLP described was made up of: an input layer; a single hidden layer with  $H$  neurons; and an output layer (Fig. 2.2). The hyperparameter  $H$  was optimized using grid search where  $H \in \{0, 2, 4, 6, 8\}$ . The model was trained using the BFGS[9] optimizer algorithm for 100 epochs, the default for the `nnet` package, the neural network package used by RMiner.

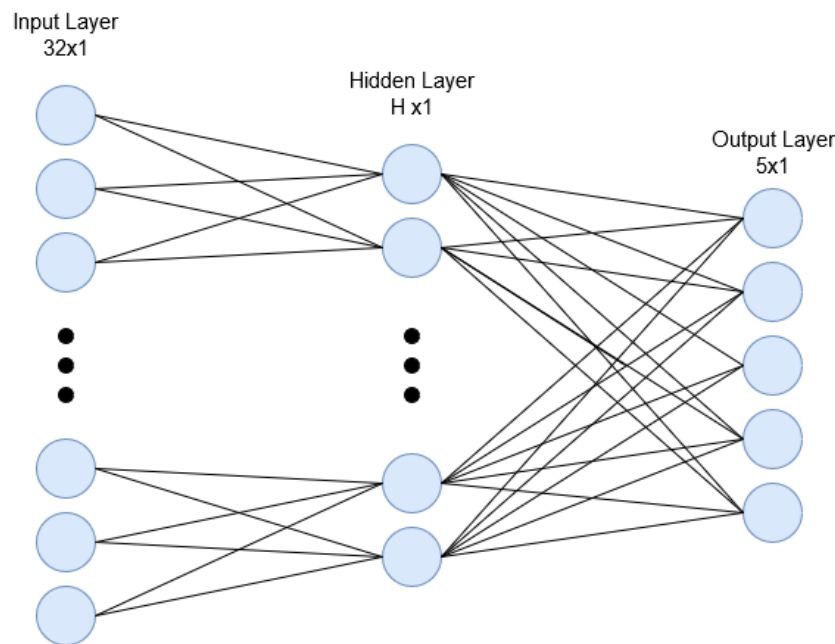


Figure 2.2: Multi-layer Perceptron architecture used in this problem

## Code Development

Code for this model was initially planned to be developed using just the Keras API with a TensorFlow backend. However Keras has no built-in BFGS optimizer. For this reason TensorFlow was used, and the Keras API was accessed through it as `tf.keras`. As the paper did not specify the loss function the mean squared error (MSE) was used<sup>1</sup> as it is the default for RMiner. The `tfp.optimizer.bfgs_minimize` function from the TensorFlow Probability library was used to minimize the loss function of the model. This function takes two positional arguments:

---

<sup>1</sup>Implemented using `tf.keras.losses.MeanSquaredError()`

- **value\_and\_gradients\_function**: A Python callable which takes a single 1D tensor of variables as input and returns the value of the function to be minimised and its gradient for that input tensor.
- **initial\_position**: The starting position of the variables.

Initially the BFGS optimizer was planned to be made by inheriting from the `Keras Optimizer` abstract<sup>2</sup> class and implementing the required methods. However this design decision was changed later as `bfgs_minimize` made it difficult to implement the required functions due to it not returning intermediate values. Another problem encountered was the fact that this function uses a single 1D tensor for the variable of the function, while the model uses multiple 2D tensors for the weights of the MLP. To solve this the 4 'weight' tensors<sup>3</sup> were mapped to and from a 1D tensor using TensorFlow's `dynamic_stitch` and `dynamic_partition`. A blog post by Py Chuang implementing a similar optimizer was used as reference[10] for creating the `value_and_gradient` function.

For my implementation I decided to extend `tf.keras.Sequential` to create a `BfgsMlp` class, which creates MLP with a single hidden layer. This overrides the `fit` method from `Sequential` to use the `bfgs_minimize` optimizer. A number of extra methods were also added to facilitate its use:

- **\_make\_val\_and\_grad\_func(self, X, y)**: A factory function to make the `value_and_gradient` function, which calculates the loss and gradient of the error surface for presentations `X` with labels `y`.
- **\_update\_weights(self, weights\_1d)**: Updates the model weights using a 1D tensor calculated by the optimizer
- **\_get\_weights\_1d(self)**: Gets the current weights for the model mapped to a 1D tensor

---

<sup>2</sup>While Python does not have abstract classes this is effectively an abstract class used to define all optimizer's behaviour

<sup>3</sup>2 weight matrices and 2 bias vectors

- `_make_1d_mapping(self)`: Called by `__init__` to create a mapping to and from a 1D tensor for the model weights.

Using this model the hyper-parameter  $H$  was then tuned using 10-fold cross-validation accuracy as the performance metric on which the best value of  $H$  was judged (Table 2.2). Once the best value of  $H$  was determined, it was desired to monitor the progression of the training and validation loss and accuracy over the 100 epochs, to see if over-fitting was occurring. There was a problem with this however, as the BFGS optimizer does not return intermediate values, and weights need to be updated in the `value_and_gradient` function. Therefore to monitor the intermediate loss and accuracy of the model, a new class, `BfgsTrainingMonitor`, was made. This takes the model and the datasets you want to test the model with each epoch, i.e. the training and validation datasets. This monitor can then be passed to the `BfgsMlp.fit` function, and uses a wrapper function `monitor(func)` to wrap the `value_and_gradient` function for the model. It then calls the models `evaluate(X, y)` function, and records the results for the metrics returned in a Python dictionary. This was used to create graphs to show how the accuracy and loss change by epoch, see Figure 2.3. There was one issue with this method however, which is that the `value_and_gradient` function appears to be called 3 times per iteration of the bfgs algorithm. No reason could be found for this in either the TensorFlow Probability documentation for this function, or its source code.

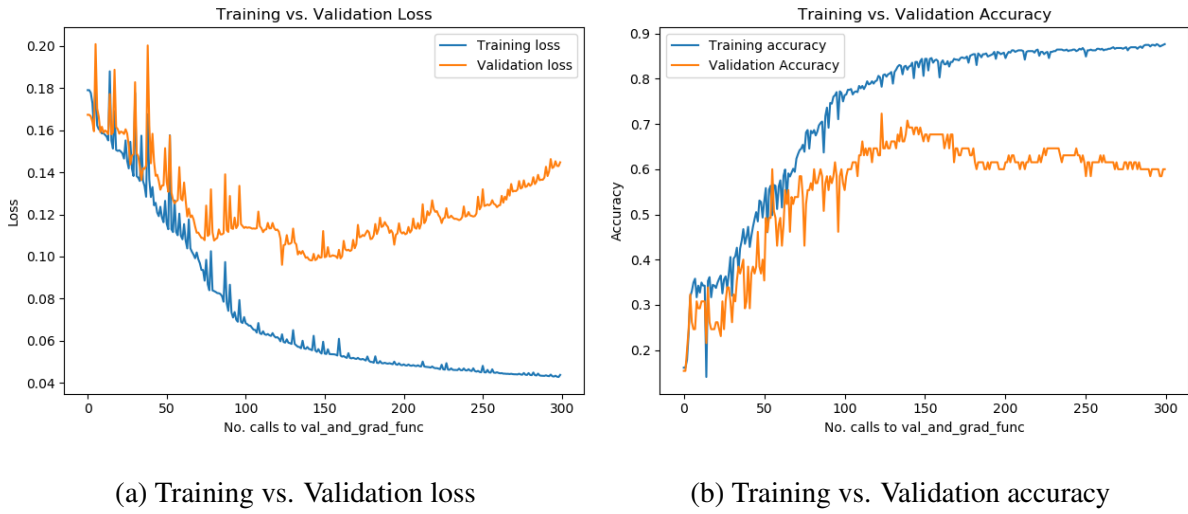
From these graphs it can be seen that after around 40 epochs the training loss continues to reduce while the validation loss begins increasing. This is a common sign that the model is over-fitting and learning the noise of the training data, rather than the underlying information[11]. Using 40 epochs and 4 hidden neurons, 5 final models were trained on 90% of the Portuguese data, and validation accuracy calculated using the other 10% of the data. The best model was then tested against the Maths data.

## Results

Table 2.2 shows the mean validation accuracy with 10-fold cross-validation for the MLP using BFGS optimization for 100 epochs with various numbers of hidden nodes,  $H$ . it can be

---

<sup>4</sup>Note No. calls to `val_and_grad_func` is  $3 \times$  the number of epochs

Figure 2.3: Training history for BfgsMlp with  $H = 4^4$ 

seen that the validation accuracy peaks when  $H = 4$ , at 67.85%. This value of  $H$  was then used by 5 models trained for 40 epochs to prevent over-fitting. The best performing model achieved a validation accuracy of 75.38%. This model was then tested against the student maths dataset, and achieved 65.32%, showing that it was able to generalise to unseen data.

<b>H</b>	0	2	4	6	8
<b>Accuracy</b>	28.92%	66.15%	67.85%	62.15%	64.62%

Table 2.2: Results of tuning H

### 2.3.2 Support Vector Machine

#### Overview

Support vector machines (SVM) were first described in 1995 by Corinna Cortes and Vladimir Vapnik in their paper *Support-Vector Networks*[12]. They are a popular machine learning algorithm as they reach a global optimum by maximising the distance between data points and the decision boundary. SVM allow for binary classification, though a multi-class SVM can be made from the combination of multiple binary SVMs, or using multi-class kernel-based vector

machines[13]. An SVM can only solve linearly-separable problems, which limits their usefulness, however through the use of the '*kernel trick*'[14], they can solve much more complex, non-linearly solvable problems. This is done by using a kernel function to map the features into some higher dimensional space, where they are linearly-separable.

SVM offer a number of hyper-parameters, such as:  $C$  a 'slack' variable; and  $\gamma$  a parameter used by the kernel functions. Only  $\gamma$  was tuned in the original paper, where  $\gamma \in \{2^{-9}, 2^{-7}, 2^{-5}, 2^{-3}, 2^{-1}\}$ . The slack variable  $C$  was left as the default value of 1, which is also the default in Scikit-Learn. No kernel was mentioned in the original paper, so several kernel functions were looked at: Radial-bias function (RBF); Sigmoid; Polynomial (degree 3); and linear, which does not map the data-points to a higher dimensionality space. No form of dimensionality reduction for the data such as principle component analysis (PCA) was looked at either, though it has been shown that removing features with little correlation to the labels can improve SVM classification performance[15]. For this reason it was looked at to demonstrate Scikit-Learn's various dimensionality reduction methods, and to see if it has any improvement on classification accuracy.

## Code Development

Code for this model was made using Scikit-Learn's `sklearn.svm.SVC` class. To tune the  $\gamma$  value, scikit-learn's `sklearn.model_selection.GridSearchCV` was used. This allows for a user to tune a model's hyper-parameter using a grid-search method, i.e. checking all combinations of hyper-parameters, and using k-fold cross-validation to measure performance. Hyper-parameters are given in a Python dictionary, along with an instance of the type of model to use and some training data. Once training of the models is done, information about the results can be accessed, along with the best performing model, and other various information about the training process. For this grid-search, both the  $\gamma$  value and the kernel functions were varied. The results for this can be seen in Tables 2.4 – 2.6 in the SVM results section. Next the affect of reducing dimensionality on the accuracy of the classification was looked at. For this a number of PCA algorithms were looked at. Principle component analysis, as the name suggests, looks at the variance captured in each feature, and continues to take the feature which represents the

most variance, until a certain threshold is reached[16]. The remaining features are then discarded, and the the selected features used as the new dataset. This also corresponds to taking the  $q$  largest eigenvectors of the data's covariance matrix, and this the method implemented by Scikit-Learn. There are numerous ways to select how many PCs should be used, such as Kaiser's stopping rule or the scree test **pca-choosing-2**. Kaiser's stopping rule states that only factors with an eigenvalue of greater than 1.00 should be used[17]. Table 2.3 shows the 11 largest eigenvalues, of which 10 would be used as they are greater than 1. The scree test takes a more graphical approach to this, by plotting the features vs. their eigenvalues in descending order, it can be seen when the eigenvalues level out, and this is taken as the cut-off[17], as seen in Figure 2.4, this happens after the first 3 features. An identical approach was then used to performing hyper-parameter tuning for the models using datasets of these reduced dimensions.

Factor	Eigenvalue
1	3.8335
2	2.5561
3	1.7764
4	1.5177
5	1.4267
6	1.3117
7	1.2629
8	1.1248
9	1.0694
10	1.0327
11	0.9958

Table 2.3: Factors and Eigenvalues

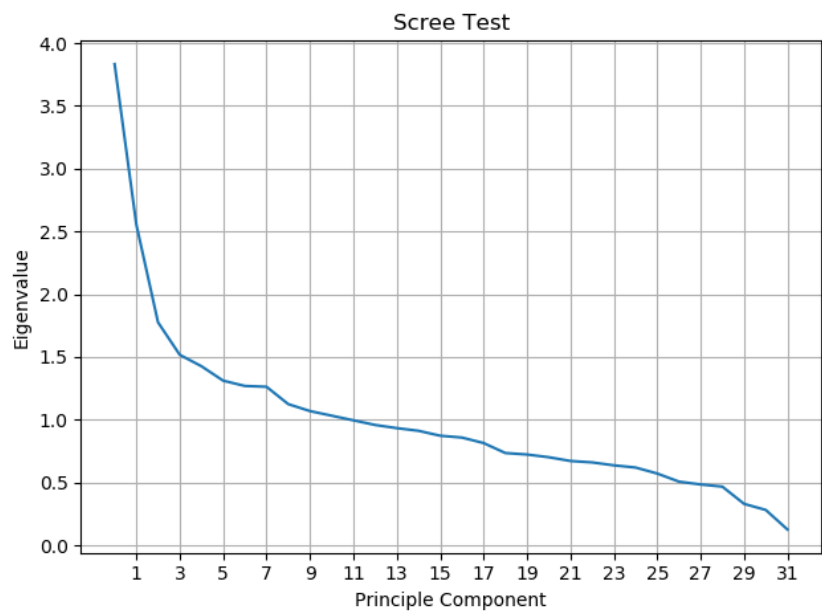


Figure 2.4: Scree Test showing factor vs. eigenvalue. Notice the drop off after 3 factors



## Results

For the first test ,not using PCA on the data, it was found that setting the kernel to 'linear', i.e. using no kernel function, got the best result and was closest to the result in the original paper, with a mean accuracy of  $67.62\% \pm 4.8$ . This is unexpected, as the best model with a kernel function only had a mean accuracy of  $51.73\% \pm 4.3$ , with a  $\gamma$  of 0.002. It also means that  $\gamma$  was unused by the best model, so tuning it was not needed. It can also be seen for all data inputs, Tables 2.4 – 2.6, that for  $\gamma < 2^{-5}$  no difference was seen in accuracy for any kernel. After this the best model from each grid-search<sup>5</sup> was tested against the maths dataset, and the following accuracies measured (Table 2.7).

		Gamma $\gamma$				
		$2^{-9}$	$2^{-7}$	$2^{-5}$	$2^{-3}$	$2^{-1}$
<b>Kernel</b>	<b>Linear</b>	<b>67.62%</b>	–	–	–	–
	<b>RBF</b>	31.84%	31.84%	31.84%	45.23%	32.37%
	<b>Poly</b>	31.84%	31.84%	31.84%	31.84%	50.02%
	<b>Sigmoid</b>	31.84%	31.84%	31.84%	38.00%	51.73%

Table 2.4: Results using no PCA (Best accuracy in bold)

		Gamma $\gamma$				
		$2^{-9}$	$2^{-7}$	$2^{-5}$	$2^{-3}$	$2^{-1}$
<b>Kernel</b>	<b>Linear</b>	<b>56.51%</b>	–	–	–	–
	<b>RBF</b>	31.84%	31.84%	31.84%	41.81%	47.93%
	<b>Poly</b>	31.84%	31.84%	31.84%	31.84%	46.75%
	<b>Sigmoid</b>	31.84%	31.84%	31.84%	37.66%	48.29%

Table 2.5: Results using Kaiser PC selection (Best accuracy in bold)

---

<sup>5</sup>Using the full data, the kaiser selected components and the scree selected components

		<b>Gamma <math>\gamma</math></b>				
		$2^{-9}$	$2^{-7}$	$2^{-5}$	$2^{-3}$	$2^{-1}$
<b>Kernel</b>	<b>Linear</b>	45.54%	–	–	–	–
	<b>RBF</b>	31.84%	31.84%	31.84%	40.77%	<b>46.39%</b>
	<b>Poly</b>	31.84%	31.84%	31.84%	31.84%	42.47%
	<b>Sigmoid</b>	31.84%	31.84%	31.84%	37.14%	33.08%

Table 2.6: Results using Scree PC selection (Best accuracy in bold)

<b>Component selection</b>	All	Kaiser	Scree
<b>Accuracy</b>	10.13%	10.13%	26.03%

Table 2.7: Accuracies of best models on maths data using different component selections

As mentioned, removing features from the data with little correlation can improve the accuracy of SVMs. For this data however this was not the case, both Kaiser and Scree principle component selections achieved worse accuracy on the Portuguese data. On the maths data none of the models performed well, with only the model using the 3 scree selected components performing better than random guessing with an accuracy of 26%. This most likely means that the SVMs were learning some underlying data that did not correlate well between maths and Portuguese. However there is another benefit of reducing dimensionality, which is the reduction to training times. Using the full 32 features of the training set, it took 0.04 seconds to train and model using a linear kernel on average, compared to 0.014 seconds when using 10 features and 0.009 seconds when using 3 features. While for a dataset of this size training time is not an issue, for datasets that could have over a 100,000 presentations this could represent reasonable trade-off in accuracy for training time, especially if features with low variance exist in the dataset.

### 2.3.3 Decision Tree

#### Overview

Binary trees are a very common and well studied data-structure in computer science. Searching through a balanced binary tree is very efficient,  $O(\log N)$ , where  $N$  is the number of nodes. Decision trees are binary trees constructed to classify data based on features[14]. This makes them very fast for querying once they have been constructed. There are a number of algorithms for constructing decision trees, such as ID3[18] or CART[19]. These are usually greedy algorithms that focus on trying to separate the features to split the classes the most evenly. To do this they use some measure of informational entropy, which measures how much the data can be split by a single feature, which is a maximum when it splits the data in half[20].

The default entropy function used by the `rpart` models, the decision tree package used by `rminer` is Gini impurity, Eqn. 2.1, which is the entropy measurement used by CART. This function computes the 'purity' of a node, where a pure node is one that corresponds to a single class  $i$  in the dataset, if that node where to split the data based on some feature  $k$ . If we count the number of data points  $N(i)$  then we can calculate the Gini impurity as:

$$G_k = 1 - \sum_{i=1}^c N(i)^2 \quad (2.1)$$

Along with the criterion function on which nodes are split, there are a number of other hyper-parameters that can be used to improve decision tree performance. Of particular importance to preventing over-fitting are the maximum depth of the decision tree, and the *complexity parameter* used in minimal cost-complexity pruning. The maximum depth, as the name suggests, simply limits the maximum number of layers a decision tree can have. This stops the tree from splitting features too many times to try and correctly classify every point. Minimal cost-complexity pruning is an algorithm for pruning nodes from trees to minimise  $R_\alpha(T)$ , where  $\alpha \geq 0$  is the complexity parameter (Eqn. 2.2). Here  $R(T)$  is the total misclassification rate of tree  $T$ , and  $|T|$  is the number of terminal nodes in  $T$ .

$$R_\alpha(T) = R(T) + \alpha|T| \quad (2.2)$$

## Code Development

For tuning the max depth and the complexity parameter of the decision tree `GridSearchCV` was again used. While `rpart` supports both of these hyper-parameters, it does not have default values for them, and none was mentioned in the paper. For this reason the grid-search was set to tune the `max_depth` and the `ccp_alpha` hyper-parameters of the `DecisionTreeClassifier` from Scikit-learn. The max depth was tried with 3 different values, 3, 5, and None. For the complexity parameter two values were tested, 0, and the optimal  $\alpha$  value. The optimal  $\alpha$  value was calculated as follows:

1. Using `DecisionTreeClassifier.cost_complexity_pruning_path(X, y)` the effective  $\alpha$  of each node in a tree can be calculated. This is the  $\alpha$  at which the node would be pruned from the tree.
2. From these effective alphas we can create DTs using these values and plot the number of nodes and depth of the tree's against  $\alpha$ , as seen in Figure 2.5, the depth and number of nodes quickly drop off as  $\alpha$  increases.
3. Using these models we can also plot the training and validation accuracy against  $\alpha$ , as seen in Figure 2.6. From this the best  $\alpha$  to prevent over-fitting can be seen to be around 0.005. This value maximises validation accuracy, and as seen in Figure 2.5 also reduces the number of nodes and the depth drastically, improving generalisation of the trees.

The two best trees were then graphed using `sklearn.tree.export_graphviz` and `graphviz`. These can be seen in Figures 2.7 and 2.8. These offer an easy way to see how the data is being split at each node, and the effect of cost-complexity pruning on a decision tree<sup>6</sup>. The colour each node corresponds to the dominant class at that node, with shade corresponding to the proportion of data points of that class at that node. From these it can be seen how cost-complexity pruning reduces the number of terminal nodes.

---

<sup>6</sup>A full page version of these trees can be seen in the appendix, as they are quite large

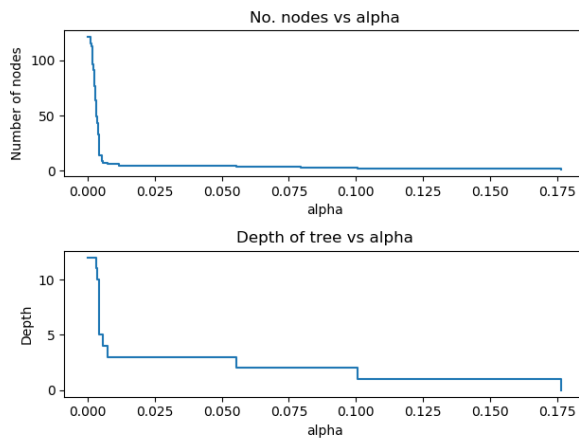
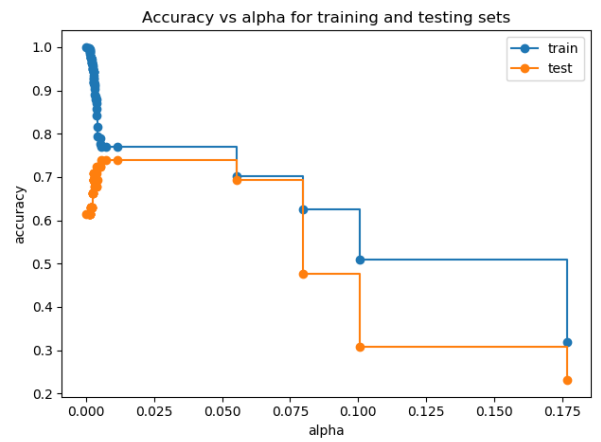
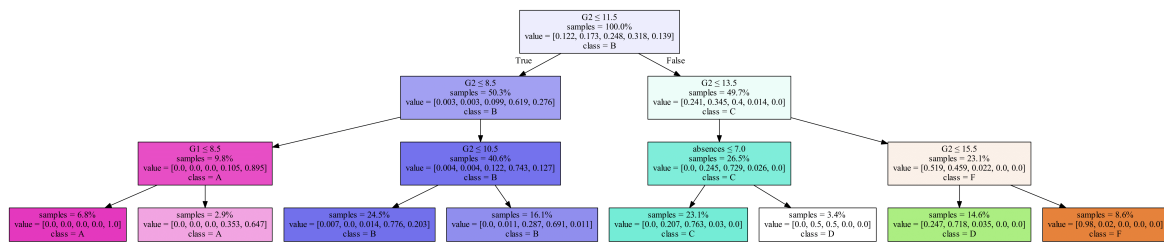
Figure 2.5:  $\alpha$  vs. Size of treeFigure 2.6:  $\alpha$  vs. Training and Validation accuracy

Figure 2.7: Best decision tree without pruning

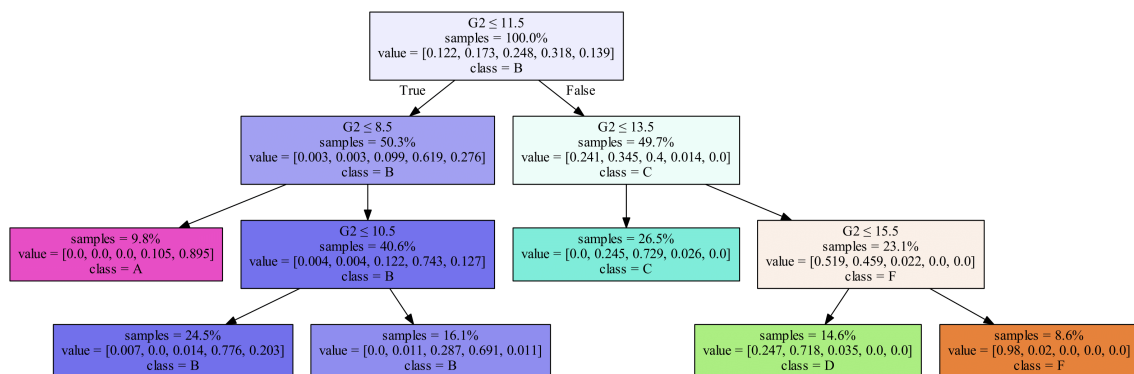


Figure 2.8: Best decision tree with pruning

## Results

The results from the hyper-parameter grid-search using 10-fold cross-validation can be seen below, Table 2.8. From this a number of observations can be made about the effect of both the max depth and  $\alpha$  value have on a decision tree's accuracy and training time. The first observation is that both limiting the max depth and using cost-complexity pruning improve validation results greatly. Without pruning limiting the max depth improves validation accuracy by around 14%, and without limiting the depth pruning improves the validation accuracy by around 11%. However it is also seen that the combination of limiting depth and pruning does not improve the validation much compared to the use of just on of these techniques, only around 0.5%.

Max Depth	$\alpha$	Mean fit time (s)	Mean accuracy	Standard deviation
3	0	0.0000	75.83%	5.6%
5	0	0.0020	68.28%	6.5%
—	0	0.0038	61.25%	9.2%
<b>3</b>	<b>0.005</b>	<b>0.0025</b>	<b>76.34%</b>	<b>5.8%</b>
5	0.005	0.0032	73.93%	7.2%
—	0.005	0.0050	72.38%	9.1%

Table 2.8: GridSearchCV results

The best model with and without pruning were then evaluated using the unseen maths dataset. With pruning an accuracy of 74.94% was measured on the unseen data, compared to 71.39% with pruning.

### 2.3.4 Random Forest

#### Overview

Random forests are a type of ensemble learner made up of decision trees. Ensemble learners work by taking a large number, of weak learners<sup>7</sup>, each classifying the data in a different way, and combining there predictions using some algorithm[14]. To create a random forest the method of training the decision trees is modified slightly. Now when a node is expanded only some subset of features  $m$  is considered to split the data on. The trees are then trained and combined using bootstrap-aggregation, also known as bagging. Bootstrapping involves taking some random subset of the dataset with replacement. A number of these bootstrap samples are combined to produce a new dataset of the same size as the original dataset, but that may now contain duplicates.

Again the original paper used the defaults for training the RF. The RF used by `rminer` comes from the `randomForest` package in R. This defaults to 500 trees, and a minimum node size of 1 for classification. Node size refers to the number of data points associated with a node. This parameter is equivalent to the `min_samples_leaf` in Scikit-learn's `RandomForestClassifier`. However the size of the trees can also be limited by the `max_depth` parameter. Scikit-learn uses 100 trees with no max depth if the default options are used. It was decided to see how 100 trees perform in this task compared to 500. As neither library limits the size of the trees by default, and producing 100's of large trees can take significant time, it was decided to also seem how limiting the size of the trees using both `max_depth` affected the accuracy. This was checked using a grid-search method, as with the previous models.

#### Code Development

As hyper-parameters were to be tuned using a grid-search method, `GridSearchCV` was again used. The `min_samples_leaf` argument can take either an integer value, or a float, depending on if it should be interpreted as the absolute number or a proportion of the total

---

<sup>7</sup>A weak learner is a model whose accuracy is slightly better than chance

dataset. Here the hyper-parameters to be tuned were `n_estimators`  $\in \{100, 500\}$ , and `max_depth`  $\in \{3, \text{None}\}$ . Graphing some of the trees produced, Figure ??, it can be seen that now many more features are considered for splitting compared to when just using a single decision tree, which mainly used just *G1* and *G2*.

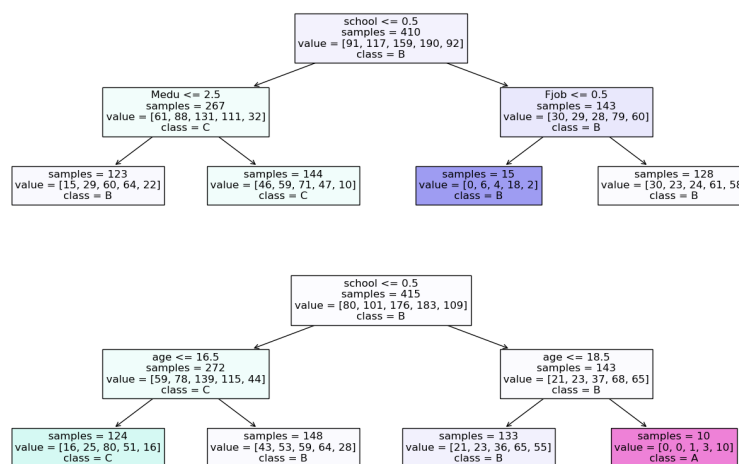


Figure 2.9: Two of the decision trees making up a random forest

## Results

The results from the grid-search cross-validation can be seen below. Of note is the fact that while limiting the size of the trees generated reduced the fitting time by around 20%, not limiting the size, and reducing the number of trees not only reduced the training time by around 75%, but also achieved a slightly better validation accuracy. The best tree was then tested on the unseen maths data and an accuracy of 64.81%.

### 2.3.5 Comparison of Results

Comparing the results from the NN, SVM, DT, and RF implementations, using Python frameworks Scikit-learn and Tensorflow following the original paper using R, and implementing



Max Depth	No. Estimators	Mean fit time (s)	Mean accuracy	Standard deviation
3	100	0.165	67.03%	5.4%
3	500	0.845	68.42%	5.5%
None	100	0.214	69.48%	5.3%
None	500	1.076	<b>71.32%</b>	6.6%

Table 2.9: GridSearchCV results

extra techniques (Table 2.10) it can see that both using R frameworks and Python frameworks achieves comparable results. Table 2.10 also shows that of the three models in which extra techniques were explored, only the limiting of the Decision tree's size had a positive impact on accuracy.

Model	NN	SVM	DT	RF
<b>R</b>	65.1%	64.5%	76.1%	<b>73.5%</b>
<b>Python</b>	67.9%	<b>67.6%</b>	75.8%	71.4%
<b>Python + modifications</b>	<b>75.38%</b>	51.7%	<u><b>76.3%</b></u>	68.4%

Table 2.10: Overall results for all implementations, best per model type in **bold**, best overall underlined

All these models, trained on the Portuguese dataset, were then tested using the maths dataset. This was not done by the original paper, in which each dataset was used to train its own models. A comparison of the models trained on the Portuguese data and ones trained on the maths data can be seen below in Table 2.11. From this it can be seen that having more data from a different school subject did not help to improve performance at predicting maths results for SVM, DT, or RF models. It did however improve the accuracy for the NN. The SVM did not generalise well between the two subjects, which may indicate it was over-fitted to the Portuguese data.

Overall this example problem shows that the Python frameworks Scikit-learn and Tensor-Flow have comparable performance to the R RMiner framework. It also shows that a number of machine learning techniques, such as dimensionality reduction, cross-validation, grid-search

Model	NN	SVM	DT	RF
<b>Trained on Maths</b>	60.3%	<b>59.6%</b>	<b>76.7%</b>	<b>72.4%</b>
<b>Trained on Portuguese</b>	<b>65.3%</b>	26%	74.9%	64.81%

Table 2.11: Comparison of model accuracy with maths dataset when trained on maths vs. Portuguese

parameter tuning, and decision tree pruning are easily performed using Scikit-learn. It was also seen that while TensorFlow already has many standard optimizers implemented already, it is also possible use custom optimizers from different Python libraries.

## 3. Problem 2: Detecting Pneumonia

### 3.1 Problem Overview

Pneumonia is an inflammation of the lungs which can be caused by either a fungal, bacterial or viral infection. Of these bacterial or viral infection are the more common cause [source](#). According to the *The European Lung white book; Respiratory Health and Disease in Europe* it had a mortality rate in Ireland of 32.96 per 100,000 in 2011[21], which, when compared to the total death rate in Ireland for 2011 of 6.2 per 1,000, means that pneumonia was responsible for 5.3% of deaths in Ireland in 2011.

There are a number of methods to reliably rule in or out pneumonia when diagnosing a patient, such as the lack of certain symptoms or the presence of rates and bronchial breathing sounds, but chest radiography is generally considered the best method of confirming a pneumonia diagnosis[22]. Computer aided diagnosis (CAD) is already used for a number of diseases, including pneumonia[23]. Most of these CAD models focus on the use of machine vision techniques rather than machine learning approaches. However, in recent years there has been a great amount of research into the use of deep learning in medical diagnosis, in particular for use image-based diagnosis, such as MRI, CAT or X-Rays[24]. These models are often able to achieve comparable, or sometimes higher, detection rates of these diseases compared to doctors, making them very useful tools.

This example problem aims to demonstrate how a machine learning model could be developed for detecting pneumonia from chest x-rays. A number of deep learning methods will be explored, but only convolutional neural network (CNNs) models will be used, as almost all real world models for image-based medical diagnosis use CNNs[24].

## 3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a class of neural networks which use convolutional layers to extract translation invariant *features* from an *feature map*, which can be any matrix  $m \in \mathbb{R}^{w \times h \times d}$ [25], such as an image. These features can then be passed as a vector to a neural network and the neural network can then be trained as usual. CNN classification models can generally be broken up into two distinct blocks of layers; a set of layers for feature extraction, and a set of layers for classification.

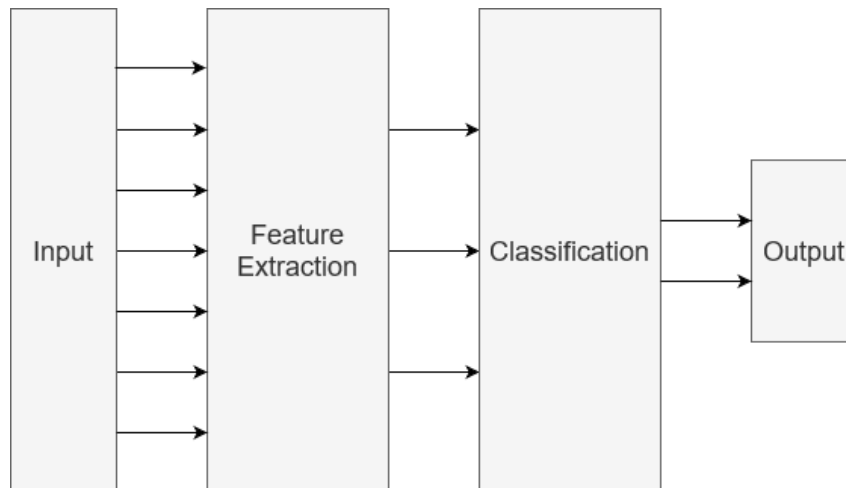


Figure 3.1: Block diagram of typical CNN

Despite being shown to be effective at solving machine vision tasks and fully trainable by back-propagation since 1989[26], it was not until as recently as 2012, when a CNN achieved a top-5 test error rate of 15.3%, compared to next highest of 26.3% in the ImageNet challenge[27] that they were considered state-of-the-art.

There are three types of layer typically found in the feature extraction block of a CNN model; Convolutional layers, pooling layers, and normalization layers[25]. Keras[3] has a number of built in implementations of these layers to allow their use in models.

### 3.2.1 Convolutional Layers

Similar to how perceptrons were designed to approximate the neurons in the brain, units in convolutional layers were designed to replicate the cells found in the visual cortex, which have a receptive field, and respond regions of the visual field[28]. To do this, convolutional layers use a number,  $n$ , of filters as their base units, which when convolved with an input feature map produce  $n$  output feature maps. The filters, commonly called kernels, are matrices of weights  $w \in \mathbb{R}$  of size  $k_w \times k_h$  where  $k_w, k_h \in \mathbb{N}_{\geq 1}$ [25].

This creates a number of hyperparameters needed to define a convolutional layer;

- $n$ , the number of filters,
- $(k_w, k_h)$  the shape of the filters, typically square,
- the activation function for the layer, which should be non-linear (similar to a normal perceptron layer),
- the stride  $s \in \mathbb{N}_{\geq 1}$ , which defines how much to move the kernel by when doing the convolutional,
- and the padding, which is used to determine the values convolved with the filter when it overlaps the edges of the image.

Keras offers a number of implementations of convolutional layers in the `keras.layers` module, covering several subtypes of convolutional layer, and with a number of arguments to set the hyperparameters mentioned above;

- Basic convolution; basic convolutional layers with various shapes of filter, such as 1D, 2D and 3D are offered by `Conv1D`, `Conv2D` and `Conv3D` respectively.
- Depthwise separable convolution; when dealing with images with multiple channels, such as an RGB image, the number of multiplications done during convolution can get very large. To reduce the number of multiplications done, depthwise convolution can be done instead, convolving each channel of the image separately, and then the channels can be

combined using a  $1 \times 1 \times d$  kernel. This produces the same result as a regular convolution, but requires far fewer multiplications, and also fewer parameters[29]. This is implemented in the keras layers `SeparableConv1D` and `SeparableConv2D`.

- Depthwise convolution; if only a depthwise convolution is desired, i.e. a convolution on each input channel individually without combining afterwards, then a `DepthwiseConv2D` layer can be used.

### 3.2.2 Pooling Layers

Pooling is used to reduce the size of a feature map, while trying to retain the features. This is done by creating a summary of  $p \times p$  areas of the image. The summary is created by applying a function to the pooled area, commonly used functions being; max pooling, taking the maximum value in the area; average pooling, taking the mean of an area;  $l_2$  pooling, which takes the  $l_2$  norm of the pooled area; and stochastic pooling, which selects a value for each area using its activation value to compute a probability.[25].

### 3.2.3 Normalization Layers

Not sure if worth talking about

## 3.3 Dataset

The chest x-rays used in this problem come from the *Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification* dataset[5]<sup>1</sup> produced by Daniel Kermany, Kang Zhang and Michael Goldbaum using and is used under the Creative Commons *CC BY 4.0* license. This dataset contains 5856 images split into 3 directories; a training directory of 5216 images; a testing directory containing 624 images; and a validation directory containing 16 images. Pneumonia is prevalent in around 75% of the training presentations, and 60% of the test presentations.

---

<sup>1</sup>Version 2

The images were loaded using Keras's `ImageDataGenerator`. This acts as a Python generator, loading in images in batches as they are needed, rather than all at once. The advantage of this is that only the images in the current batch are loaded into memory, meaning large datasets don't fill up memory unnecessarily. Using an `ImageDataGenerator` also allows for various preprocessing tasks to be carried out on the images as they are loaded.

To create a dataset usable by Keras, three `ImageDataGenerator` were made, one for training data, one for validation data, and one for test data. Due to the small number of images in the validation folder, both the training and validation presentations were drawn from the training folder. The proportion of the size of the training dataset to validation dataset was controlled using the `validation_split` argument of the `ImageDataGenerator`. The training/validation generator also used shuffling to randomise the images presented in each batch. Both generators rescaled the pixel values of the images to between 0 and 1, and resized the images to  $256 \times 256$ .

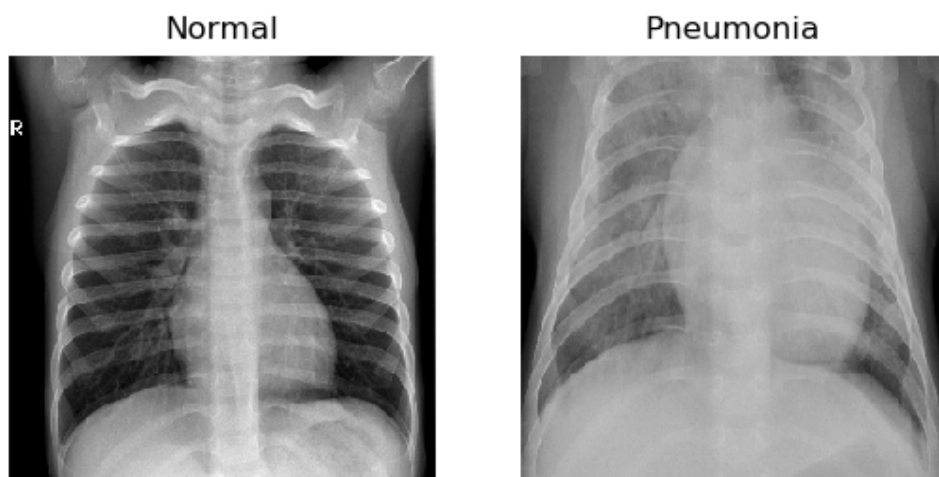


Figure 3.2: Example of x-ray with and without pneumonia

## 3.4 Explored Solutions

An iterative design approach was used to tackle this problem, with each iteration of the design using more advanced models and techniques.

### 3.4.1 Simple CNN

The first solution looked at was a simple convolutional neural network, based on a similar model on Kaggle[30]. This was composed of 3 convolutional layers, with a  $3 \times 3$  kernel size, and rectified linear activation function, each followed by a max-pooling layer, with a  $2 \times 2$  pool size. After these convolutional layers a dense layer of 64 neurons, again using the rectified linear activation function, before a single neuron output layer, using a sigmoid activation function to create a binary classifier. A dropout layer was used to remove the effect of 50% of neurons in the hidden layer impacting the output during training. This is done to help prevent over-fitting[31].

Due to the unbalance nature of the classes, a dictionary of class weights was passed to the `fit` function of the model. This is used to weigh the updates to the models weights more heavily towards presentations of class 0, i.e. x-rays without pneumonia. This has the effect of effectively balancing the data. Without doing this the model simply guesses class 1 for any presentation as it has a 75% chance of being correct. Instead of looking at accuracy during training, specificity and sensitivity were instead used as metrics. Sensitivity is the measure of the proportion of true positives identified as positives, while specificity is the measure of the proportion of true negatives identified as such. These metrics are often looked at when dealing with problems that require a low false positive or false negative rate, such as in medical diagnosis.

Keras allows for callback functions to be passed to the model to allow interaction with the model during fitting. There are several built-in callbacks offering functionality such as early-stopping, check-pointing, and logging. This model was trained for 24 epochs with a batch size of 16 images. The `TensorBoard` callback was used to monitor and record the training cycle. This callback allows for the training metrics to be monitored and re-opened in an interactive locally hosted web-page, as well as a number of other visualisations of the model and its training



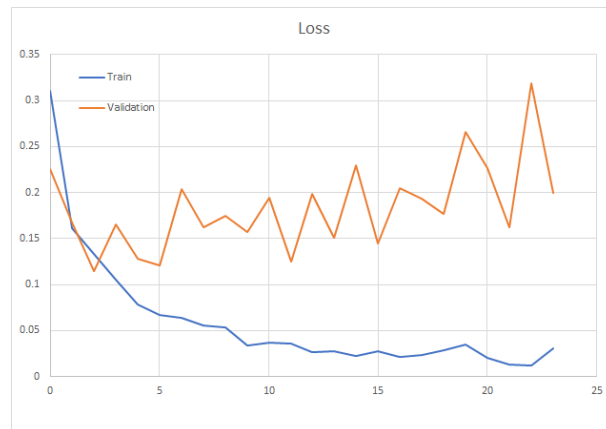


Figure 3.3: Training and validation loss

performance. Figures 3.3 – 3.4 show the training and validation loss, sensitivity, and specificity during training.

These graphs provide some insight into the model performance. From the loss (Fig. 3.3) it can be seen that the training loss continues to drop, while the validation loss increases, this is a common sign of the model over-fitting to the noise of the training data, rather than learning the underlying data. Comparing the sensitivity and specificity plots (Figs. 3.4 & 3.5), they both perform well, however the validation specificity remains much closer to the training value and is more stable than the sensitivity. This is an issue, as it implies that it has a high rate of false negatives, compared to false positives, which are relatively low on unseen data.

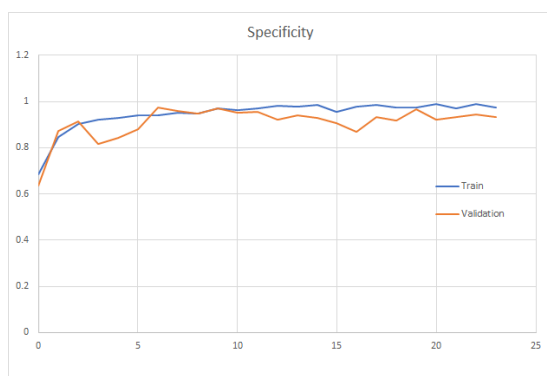


Figure 3.4: Training and validation specificity

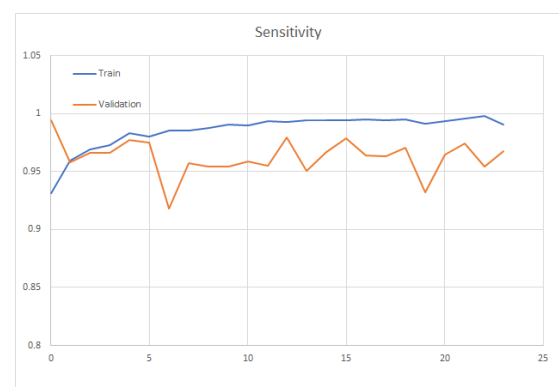


Figure 3.5: Training and validation sensitivity

### 3.4.2 Image Augmentation

The issue of over-training is a common problem for machine and deep-learning models, and often results in poor performance from the models when exposed to real-world, unseen data. A common reason for over-fitting is a lack of training data, making it easier for the model to learn the intricacies of each data-point, rather than having to learn the underlying information. Increasing the size of a dataset can often be difficult, as data, especially medical data, can be uncommon, as in the case of rare diseases, or hard to access. Instead data augmentation or warping is used to effectively create new data-points from the existing dataset[32]. This involves transforming presentations in such a way as to make a new, but still realistic presentation.

Image augmentation is a type of data augmentation that involves performing transformations on the training images, to try stop the model from learning the noise in the dataset, and instead learn the desired features[33]. The hope being that by applying semi-random augmentations to the images before they are shown to the system, undesired information will become more random and the system will learn less about it. There are a number of transformations, both elastic and rigid, that may be applied to an image. The transformations that are suitable depend largely on the subject of the image. While elastic transformations have been shown to be effective for certain medical imaging tasks, such as detecting breast cancer[34], they are not suitable for images involving rigid objects, such as chest x-rays[35].

The Keras *ImageDataGenerator* has a number of arguments that can be used to apply these transformations to images as they are presented. Of relevance are:

- Rotations; An integer can be passed to set the limit in degrees for random rotations to apply to the image using the `rotation_range` keyword argument. As not all the x-rays are perfectly vertical, a small rotation is allowable without making an unrealistic presentation.
- Shifts; The image can be shifted vertically or horizontally by a random number of pixels using the `width_shift_range` and `height_shift_range` keyword arguments. Similar to rotations small shifts in both horizontally and vertically are acceptable.
- Mirroring; The image may 50% of the time be mirrored around either the vertical or hori-

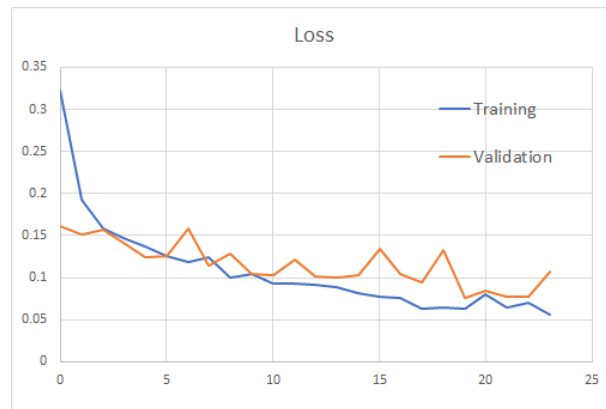


Figure 3.6: Training and validation loss

zontal axis using the `horizontal_flip` and `vertical_flip` keyword arguments. As pneumonia is possible in both the right and left lung, flipping the image along the vertical axis is reasonable.

By using these transformations, the over-fitting issue of the simple CNN described previously can be improved, as seen in Figure 3.6. However while the validation specificity has remained close to the training specificity (Fig. 3.7), the validation sensitivity has become a lot more inconsistent (Fig. 3.8), and on average dropped compared to when no image augmentation was done. This means the model is now predicting false negatives more often before, which is undesirable.

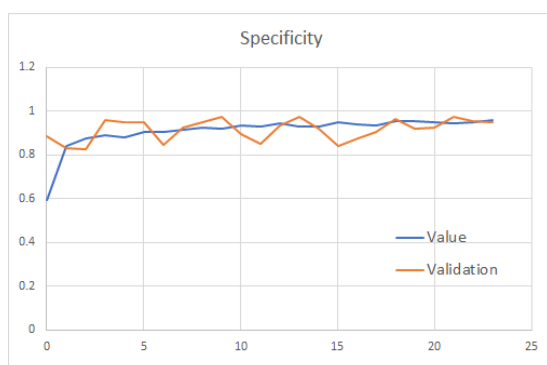


Figure 3.7: Training and validation specificity

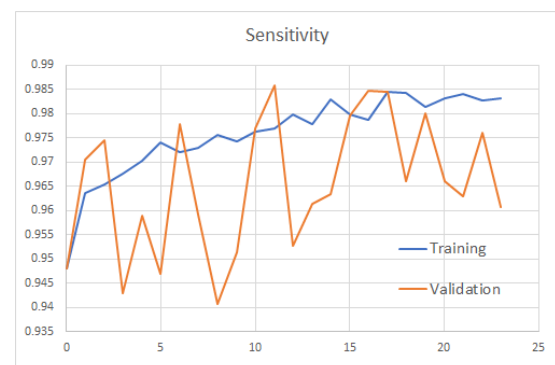


Figure 3.8: Training and validation sensitivity

### 3.4.3 Transfer learning

Now that the over-fitting issue was resolved, the erratic sensitivity and specificity issue could be looked at. The most probable reason for the inconsistent sensitivity and specificity is that there are some presentations in the validation that it doesn't classify well, so ends up guessing. This would also explain why the two metrics got worse after applying augmentation, there are now effectively more presentations it must guess on. The easiest ways to resolve this would be to allow it to train for more epochs, or to use a more powerful model, by adding more convolutional layers.

Training a more powerful model however will require more computing power, and drastically increase the time taken to train a model. Transfer learning is a common method used to bypass this problem. This is a method whereby powerful models already trained at some task are re-purposed to solve some other task. This method is common in image recognition problems and has been used for a variety of medical diagnosis models. The idea is that by taking the convolutional layers used to extract features by some powerful model trained on a vary large, highly varied dataset, such as CIFAR-100, which is a dataset with 60,000 images spread over 100 classes. A model trained on such a dataset should therefore be good at extracting a wide variety of features from images, and then feeding these features into a neural network it will be able to solve some new problem.

For this problem the VGG-16 model[36] was used as the basis for the model. The VGG-16 model was trained by Oxford's *Visual Geometry Group* and submitted into the 2014 ImageNet challenge where it won first place in localisation and second place in classification. This is one of the reasons it was chosen as a basis for transfer learning, the other two being: it has been shown to work for detecting pneumonia before[37]; and it is available through Keras. The VGG-16 model uses 18 layers for feature extraction, 13 convolutional layers and 5 max-pooling layers, as seen in Figure 3.9.

Using Keras these layers can be loaded in with a number of options, such as the input shape, and whether to initialise the weights randomly or to the weights when the model was trained on ImageNet. For this model the ImageNet weights were used and the input shape was set to  $256 \times 256 \times 3$ . This resulted in 14.7 million parameters. These weights were then fixed to

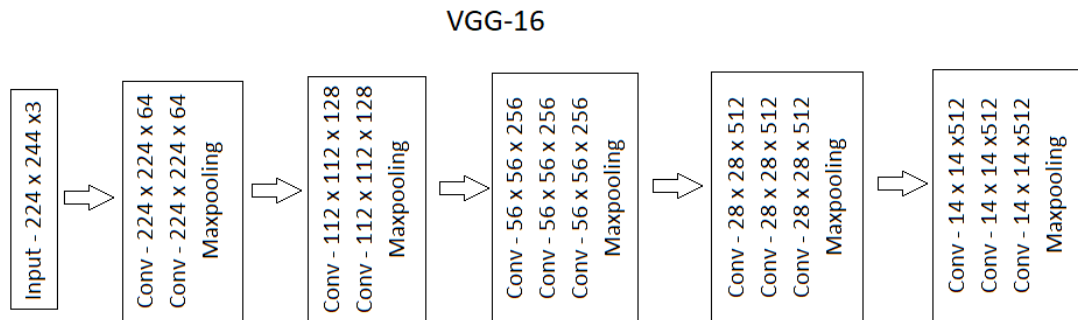


Figure 3.9: Feature extraction section of VGG-16

speed up training by looping over the layers and setting `layer.trainable = False`. To then classify the extracted features a dense layer with 256 neurons using `relu` activation were used. A `dropout` layer was then used to apply a dropout of 0.5 before a single neuron using a sigmoid activation function was used for binary classification (Fig. 3.10).

When fitting this time two new callbacks were also passed to the model, `ReduceLROnPlateau` and `EarlyStopping`. `ReduceLROnPlateau` is used to monitor some metric of the model, in this case validation loss, and if no improvement is seen in a certain number of epochs to reduce the learning rate. This is done as models often benefit from a reduction in learning rate once training stagnates, which is likely to happen sooner with a more powerful model. `EarlyStopping` is used to stop the model training if no improvement to validation loss is seen in a certain number of epochs, in this case 5 epochs. This helps to prevent over-fitting, which can happen quickly with very deep models. The model was then trained using the Adam optimizer for 32 epochs, but training was stopped by `EarlyStopping` after 10. Figures 3.11 – 3.14 show how the loss, sensitivity, specificity, and learning rate change during training.

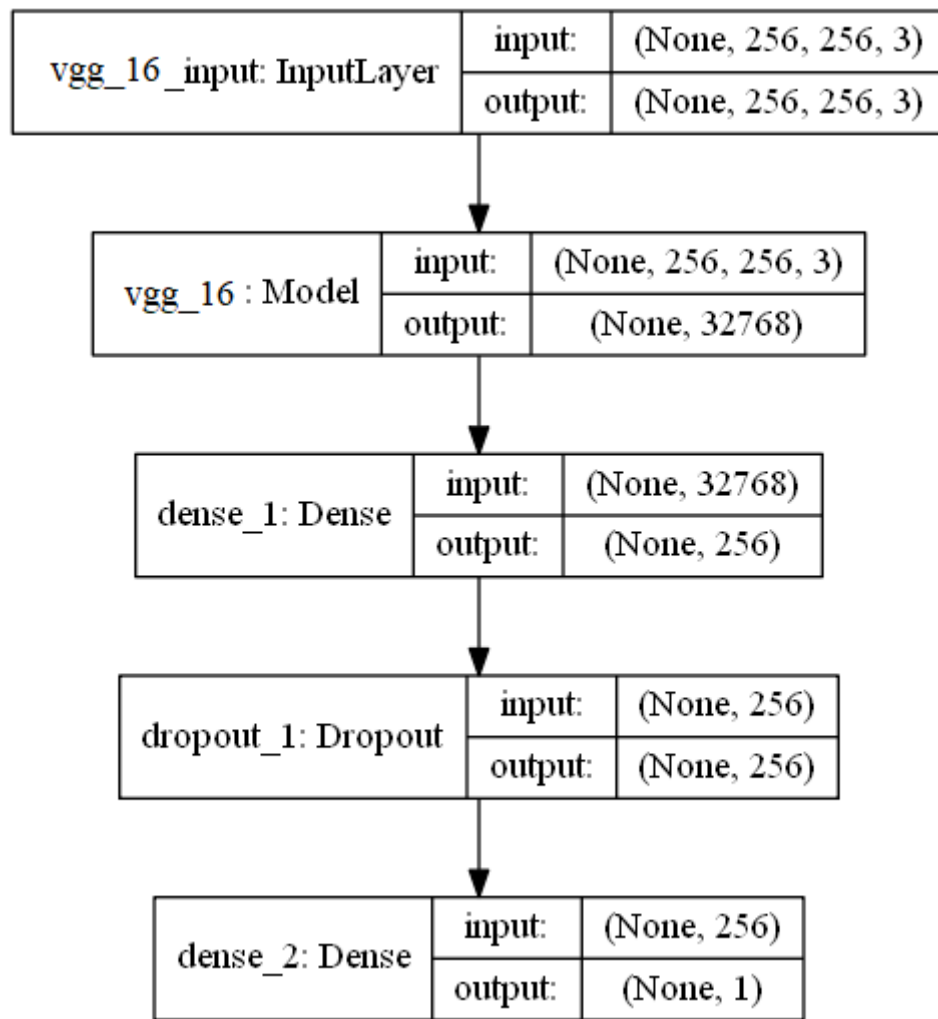


Figure 3.10: Transfer learning model architecture

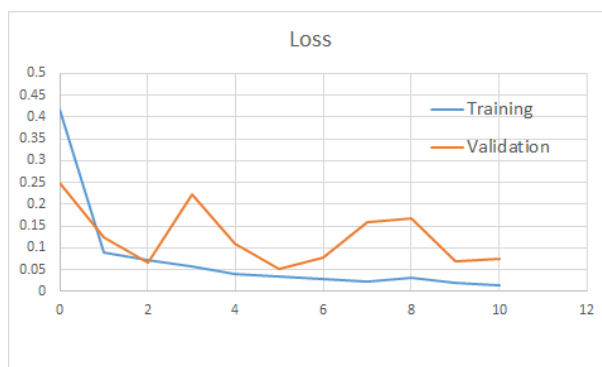


Figure 3.11: Training and validation loss

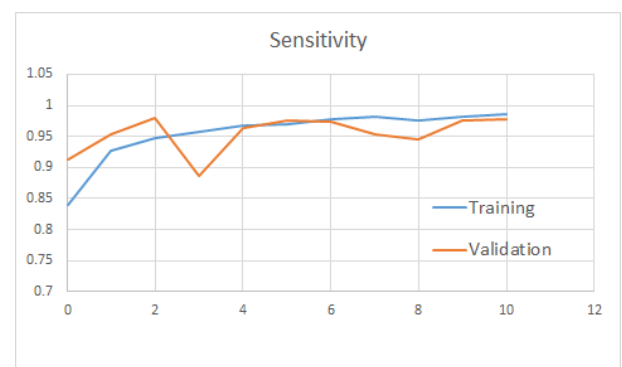
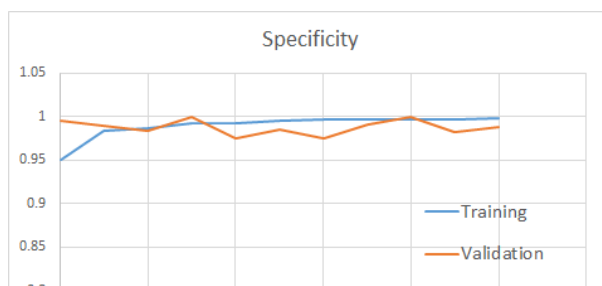


Figure 3.12: Training and validation sensitivity



From these plots it can be seen that both the validation sensitivity and specificity quite closely follows the training sensitivity and specificity. It can also be seen that the learning rate was reduced by a factor of 10 for the last 2 epochs.

### 3.5 Results

This model was then used to predict the unseen test data, and a confusion matrix was made from the results, seen in Figure 3.15. From this the sensitivity, and specificity can be calculated to be 94.46% and 67.94% respectively. The overall accuracy can also be calculated to be 87.02%. The higher sensitivity compared to specificity is good, as it is more important to have few false negative, rather than few false positives. A set of sample images and the percentage prediction for each class can be seen in Figure 3.16. This shows that it predicts correctly very confidently x-rays with pneumonia, and quite confidently for x-rays without.

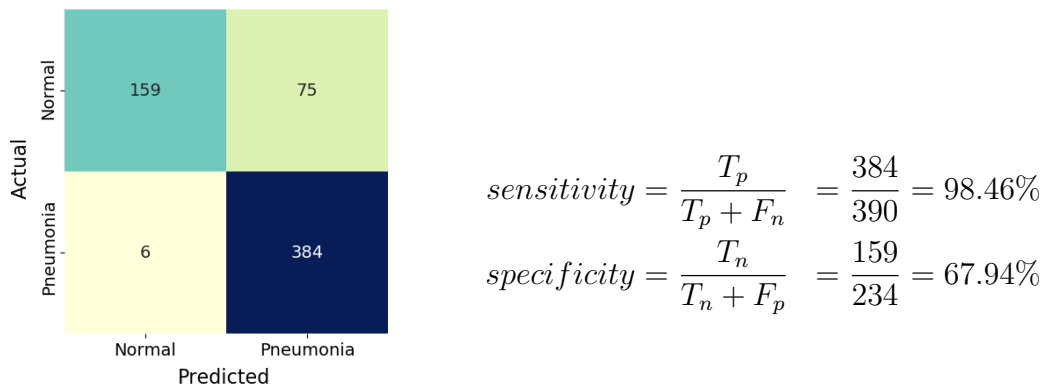


Figure 3.15: Confusion matrix of  
final model predictions

One problem encountered was when using the `Sequential.evaluate_generator` function in Keras, was the sensitivity and specificity results did not align with those calculated directly from the predictions, which it calculates as 61.54% and 25.48% respectively. It was initially assumed that this was some problem with the sensitivity and specificity function definitions, but the fact that the results from training using these functions were reasonable and that other implementations of these functions produced the same scores indicated otherwise.

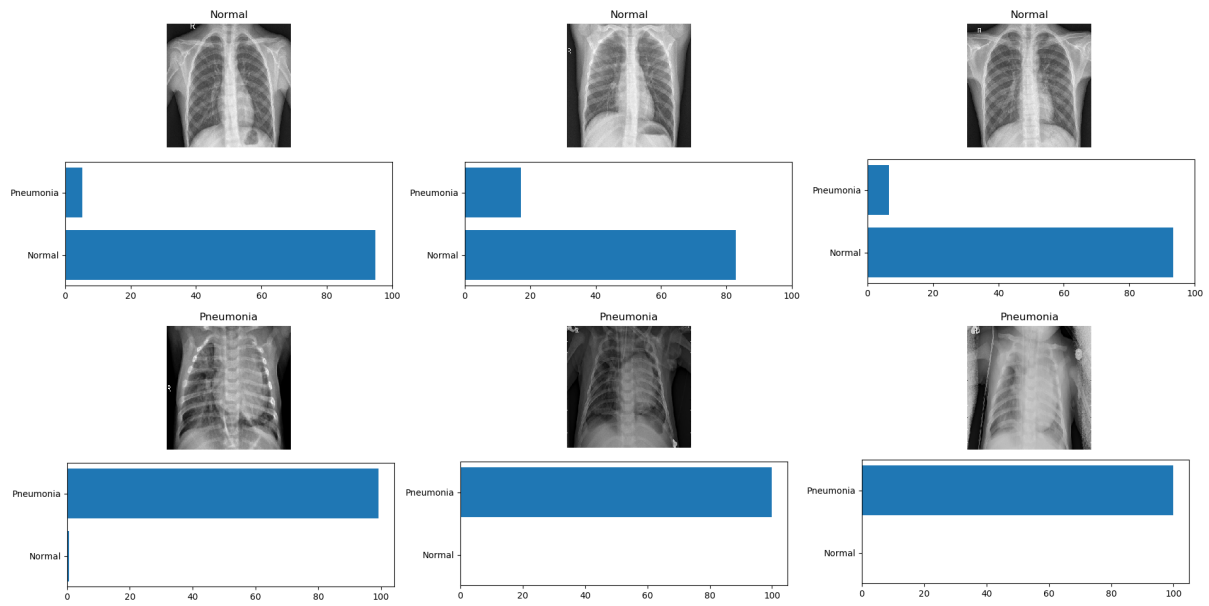


Figure 3.16: Sample of images and the prediction confidence percentage for each class

It was found that there are a number of issues reported on GitHub relating to incorrect metrics when comparing results from the `evaluate_generator` and `predict_generator` functions[38],[39],[40].



## 4. Results and Conclusions

### 4.1 Results

These two examples demonstrate some of the machine and deep learning algorithms and techniques available through the Scikit-Learn and TensorFlow frameworks, as well as the functional API's that exist to help use them, such as Keras. Both Scikit-Learn and TensorFlow work well together, allowing for tools to be shared between them. They perform as expected, matching results with models made using libraries in other languages commonly used for machine learning, such as R. There are some features missing from TensorFlow available in RMiner, such as a number of relatively common optimizers, for example BFGS. While these functions can be found and implemented from other libraries, it would be nice if there was more compatibility with them and the `Optimizer` objects in `tf.keras`, particularly the TensorFlow probability optimizer functions.

There were also a few issues when using pure Keras, instead of `tf.keras`, such as with the `evaluate_generator` and `predict_generator` functions producing conflicting sensitivity and specificity results.

### 4.2 Future work

As mentioned previously a third example problem looking at forecasting using time-series data was planned to be explored, to look at LSTM networks and regression using these frameworks. An outline of the intended plan for this problem is included in the appendix. An examination of why `evaluate_generator` and `predict_generator` produce conflicting results could also be done. Due to time constraints an example to recreate the problem was not made, and no issue submitted to the Keras Github repository, however this will be done.

## A. Full Decision Trees

The full decision trees made with and without using cost-complexity pruning for example problem 2 can be seen below, Figures A.1 & A.2.

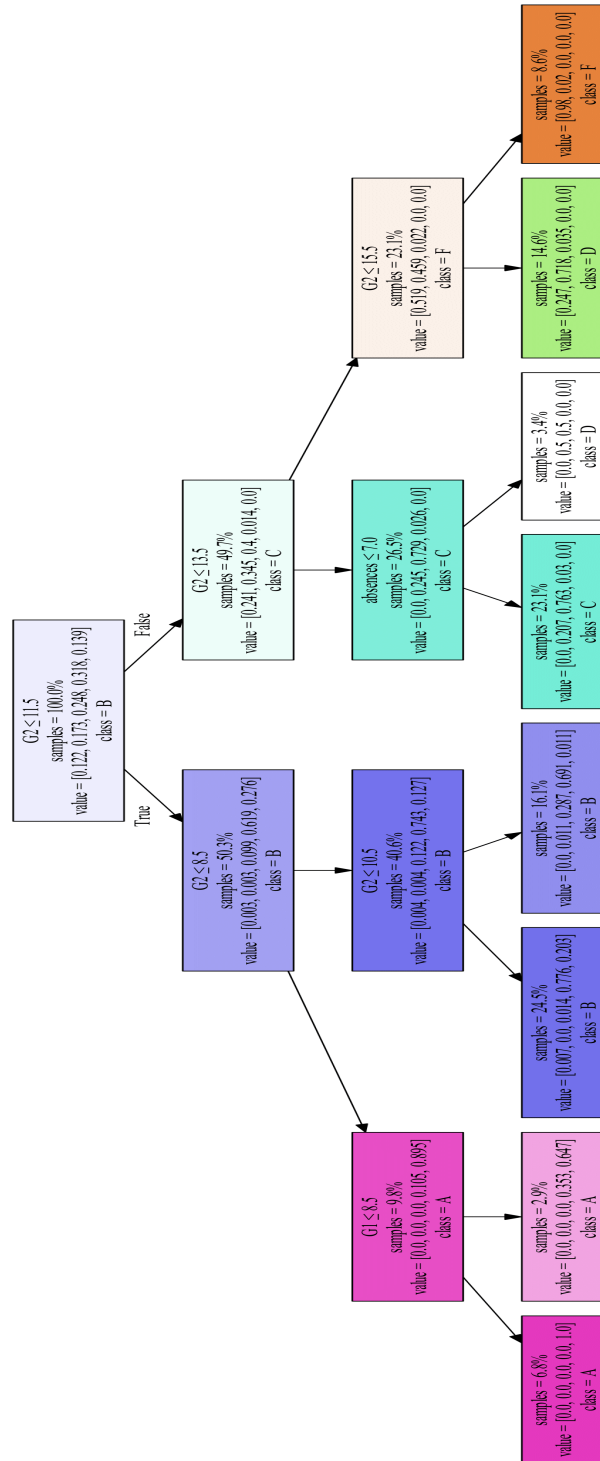


Figure A.1: Decision tree made with no cost complexity pruning

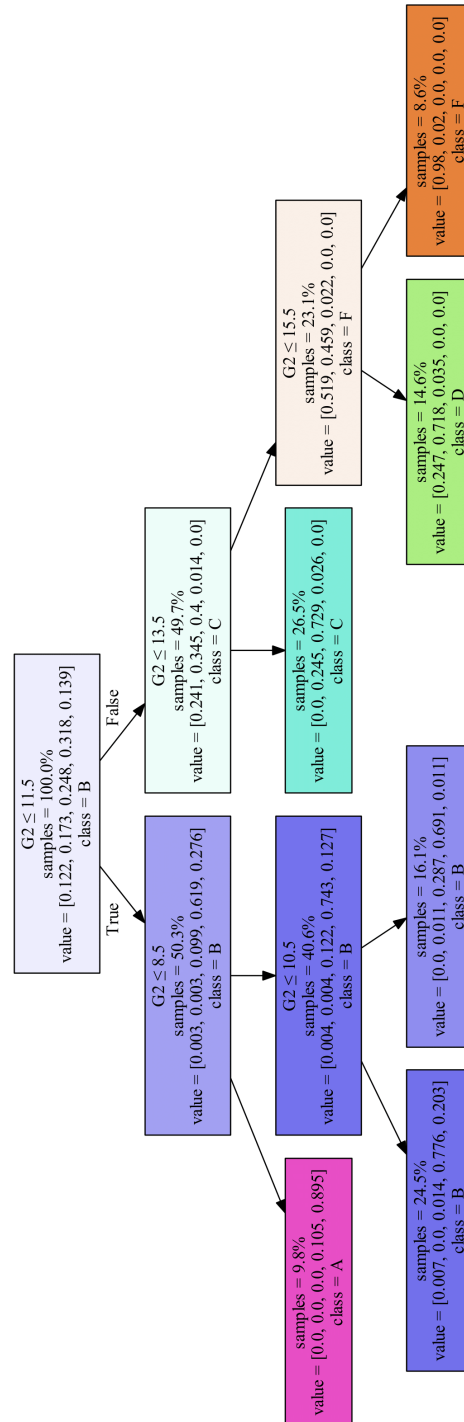


Figure A.2: Decision tree made with cost complexity pruning

## B. Example Problem 3 outline

### Project overview

This example problem was planned to use deep learning recurrent neural networks (RNN) to forecast Bitcoin prices using historic data. RNN and in particular LSTM models often perform very well at tasks with time-series data, i.e. datasets where the order of the presentations matters. This has made them quite common architectures for text or speech comprehension tasks, as well as forecasting. The reason for this problem being selected was to show how RNN and LSTM models can be implemented using the Python framework, Keras.

### Dataset

The dataset for this project was from Kaggle[41]. It consisted of 2 csv files containing records from the *Bitstamp* and *Coinbase* trading platforms for Bitcoin historic data in 1 minute intervals from 2012 and 2014 till 2019 respectively. The data present was arranged in 8 columns: **timestamp**, the time of the record; **open**, price at the start of that minute; **close**, price at the end of that minute; **high**, the max price for that minute; **low**, the minimum price for that minute; **volume\_btc**, amount of bitcoin traded in that minute; **volume\_currency**, amount of bitcoin in US dollars traded in that minute; and **weighted\_price**, volume-weighted average price for that minute.

### Code development

No code was developed for this problem, however it was planned to use some sort of RNN, most likely an LSTM network, as a number of other price forecasting projects have used LSTM models and achieved good results with them[42], [43]. Keras was to be used as it offers a number of RNN layer types such as RNN, LSTM, and GRU.

# Bibliography

- [1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. [Online]. Available: <https://www.tensorflow.org/>.
- [3] F. Chollet *et al.*, *Keras*, <https://keras.io>, 2015.
- [4] P. Cortez and A. M. G. Silva, “Using data mining to predict secondary school student performance,” 2008.
- [5] D. Kermany, K. Zhang, and M. Goldbaum, “Labeled optical coherence tomography (oct) and chest x-ray images for classification,” *Mendeley data*, vol. 2, 2018.
- [6] W. McCulloch and W. Pitts, “A logical calculus of the ideas imminent in nervous activity. bulletin of mathematical biophysics 5: 115-33,” 1943.
- [7] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.
- [8] R. P. Lippmann, “An introduction to computing with neural nets,” *IEEE Assp magazine*, vol. 4, no. 2, pp. 4–22, 1987.
- [9] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006, pp. 136–140.
- [10] P. Chuang, *Optimize tensorflow & keras models with l-bfgs from tensorflow probability*. [Online]. Available: <https://pychao.com/2019/11/02/optimize-tensorflow-keras-models-with-l-bfgs-from-tensorflow-probability/>.

- [11] G. Panchal, A. Ganatra, P. Shah, and D. Panchal, "Determination of over-learning and over-fitting problem in back propagation neural network," *International Journal on Soft Computing*, vol. 2, no. 2, pp. 40–51, 2011.
- [12] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [13] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *Journal of machine learning research*, vol. 2, no. Dec, pp. 265–292, 2001.
- [14] S. Marsland, *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2014.
- [15] L. Cao, K. S. Chua, W. Chong, H. Lee, and Q. Gu, "A comparison of pca, kpca and ica for dimensionality reduction in support vector machine," *Neurocomputing*, vol. 55, no. 1-2, pp. 321–336, 2003.
- [16] M. E. Tipping and C. M. Bishop, "Mixtures of probabilistic principal component analyzers," *Neural computation*, vol. 11, no. 2, pp. 443–482, 1999.
- [17] D. A. Jackson, "Stopping rules in principal components analysis: A comparison of heuristic and statistical approaches," *Ecology*, vol. 74, no. 8, pp. 2204–2214, 1993.
- [18] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.
- [19] L. Breiman, *Classification and regression trees*. Routledge, 2017.
- [20] A. Rényi *et al.*, "On measures of entropy and information," in *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, The Regents of the University of California, 1961.
- [21] *The european lung white book; respiratory health and disease in europe*, <https://www.erswhitebook.org/chapters/acute-lower-respiratory-infections/>, Accessed: 2020-02-03.

- [22] R. R. Watkins and T. L. Lemonovich, "Diagnosis and management of community-acquired pneumonia in adults," *American family physician*, vol. 83, no. 11, pp. 1299–1306, 2011.
- [23] L. L. G. Oliveira, S. A. e Silva, L. H. V. Ribeiro, R. M. de Oliveira, C. J. Coelho, and A. L. S. Andrade, "Computer-aided diagnosis in chest radiography for detection of childhood pneumonia," *International journal of medical informatics*, vol. 77, no. 8, pp. 555–564, 2008.
- [24] M. Bakator and D. Radosav, "Deep learning and medical diagnosis: A review of literature," *Multimodal Technologies and Interaction*, vol. 2, no. 3, p. 47, 2018.
- [25] M. Thoma, "Analysis and optimization of convolutional neural network architectures," Masters's Thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany, Jun. 2017. [Online]. Available: <https://martin-thoma.com/msthesis/>.
- [26] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.
- [27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," pp. 1097–1105, 2012.
- [28] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.
- [29] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.
- [30] *Keras-nn*  $\Gamma^{026b30dx} - ray\Gamma^{026b30d}predictpneumonia\Gamma^{026b30d}86.54\%$ , Accessed 07/11/2019. [Online]. Available: <https://www.kaggle.com/kosovanolexandr/keras-nn-x-ray-predict-pneumonia-86-54>.
- [31] G. E. Hinton, A. Krizhevsky, I. Sutskever, and N. Srivastva, *System and method for addressing overfitting in a neural network*, US Patent 9,406,017, Aug. 2016.



- [32] P. Y. Simard, D. Steinkraus, J. C. Platt, *et al.*, “Best practices for convolutional neural networks applied to visual document analysis,” in *Icdar*, vol. 3, 2003.
- [33] L. Perez and J. Wang, “The effectiveness of data augmentation in image classification using deep learning,” *arXiv preprint arXiv:1712.04621*, 2017.
- [34] E. Castro, J. S. Cardoso, and J. C. Pereira, “Elastic deformations for data augmentation in breast cancer mass detection,” in *2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, IEEE, 2018, pp. 230–234.
- [35] S. Stirenko, Y. Kochura, O. Alienin, O. Rokovyi, Y. Gordienko, P. Gang, and W. Zeng, “Chest x-ray analysis of tuberculosis by deep learning with segmentation and augmentation,” in *2018 IEEE 38th International Conference on Electronics and Nanotechnology (ELNANO)*, IEEE, 2018, pp. 422–428.
- [36] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [37] S. Rajaraman, S. Candemir, I. Kim, G. Thoma, and S. Antani, “Visualization and interpretation of convolutional neural network predictions in detecting pneumonia in pediatric chest radiographs,” *Applied Sciences*, vol. 8, no. 10, p. 1715, 2018.
- [38] *Evaluate\_generator produces wrong accuracy scores? #6499*, Open Issue. [Online]. Available: <https://github.com/keras-team/keras/issues/6499>.
- [39] *Why the return of predict and predict\_generator are different? #3477*, Closed Issue. [Online]. Available: <https://github.com/keras-team/keras/issues/3477>.
- [40] *Correct use of fit\_generator, predict\_generator, and evaluate\_generator #11754*, Open Issue. [Online]. Available: <https://github.com/keras-team/keras/issues/11754>.
- [41] *Bitcoin historical data*, Available under CC 4.0. [Online]. Available: <https://www.kaggle.com/mczielinski/bitcoin-historical-data>.

- [42] S. Selvin, R. Vinayakumar, E. Gopalakrishnan, V. K. Menon, and K. Soman, “Stock price prediction using lstm, rnn and cnn-sliding window model,” in *2017 international conference on advances in computing, communications and informatics (icacci)*, IEEE, 2017, pp. 1643–1647.
- [43] E. Karakoyun and A. Cibikdiken, “Comparison of arima time series model and lstm deep learning algorithm for bitcoin price forecasting,” in *The 13th multidisciplinary academic conference in prague 2018 (the 13th mac 2018)*, 2018, pp. 171–180.