Machine Learning Using Python Frameworks

Lorcan Williamson

Supervisor: Dr. K. Murphy

March 2, 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

1. Introduction

The aim of this project was to look at machine and deep learning frameworks in Python and the code development process to build and train various ML/DL models using them. To this end 3 of the main machine learning frameworks were looked at: Scikit-learn[1]; TensorFlow[2]; and Keras[3], using TensorFlow for the backend. Initially 3 example problems were 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 aimed to cover a variety of common machine learning models such as Support Vector Machines, Decision Trees, Random Forests and Multi-Layer Perceptrons. The work done for this example loosely followed the work done in the paper this data was originally collected for *Using Data Mining To Predict Secondary School Student Performance*[4]. The data mining models for this were developed using R, so this also allows for a comparison between Python and R machine learning 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. The data for this problem came from *Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification*[5].

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, running: Deep Learning AMI (Ubuntu) Version 24.1)

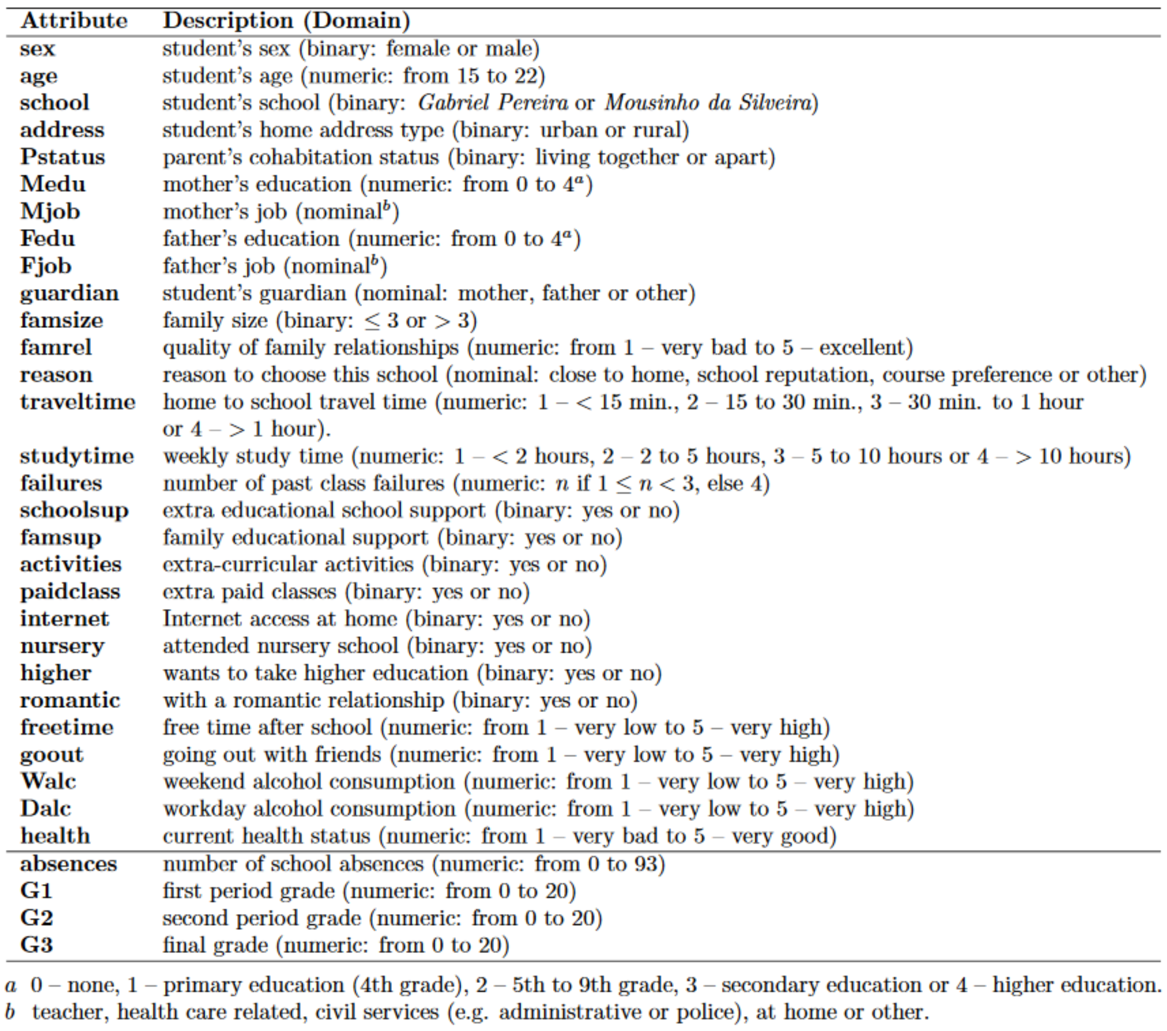
2. Problem 1: Predicting Student test Scores

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: SVMs; DTs; RFs; and NNs. 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]. The original paper used R to develop their models. By using similar models and methodology a caparison 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. Both files have 33 columns, explained in table 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 the non-numeric features, such as *school* and *Mjob* were converted to numeric using a one-hot-label encoding. This preprocessing was done using Scikit-learns `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: $16-20 :\rightarrow 5(A), 14-15 :\rightarrow 4(B), 12-13 :\rightarrow 3(C), 10-11 :\rightarrow$



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: *Gabriel Pereira* or *Mousinho da Silveira*)
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[a])
Mjob	mother's job (nominal[b])
Fedu	father's education (numeric: from 0 to 4[a])
Fjob	father's job (nominal[b])
guardian	student's guardian (nominal: mother, father or other)
famsize	family size (binary: ≤ 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

$2(D), 0-9 :\rightarrow 1(F)$

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, SVMs, DT, RF and ANN. 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 models were also tested on the maths dataset to see if they could perform well on unseen data from a different 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 putting 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 (relu) has become a common standard in recent years[7].

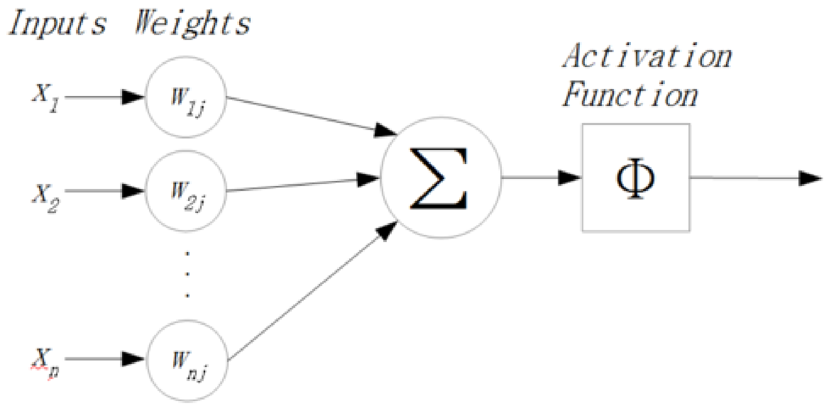


Figure 2.1: The McCulloch-Pitts neuron

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 hidden layer with H neurons; and an output layer. The hyper-parameter

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 defaults for the `nnet` package, the neural network package used by RMiner.

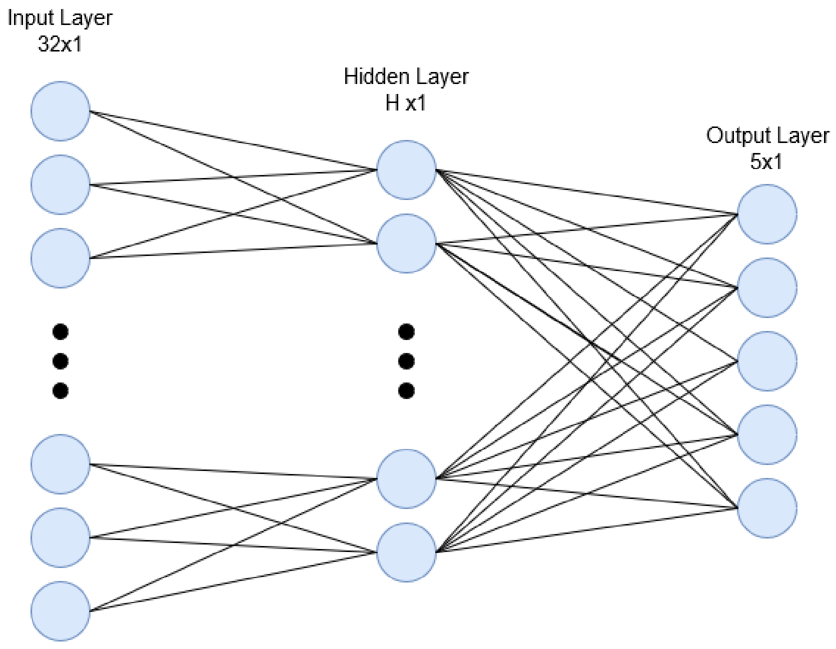


Figure 2.2: Multi-layer Perceptron architecture

Code Development

Code for this model was initially planned to be developed using the Keras API with a TensorFlow backend. However Keras has no built in BFGS optimizer. This meant base TensorFlow would need to be used as it easier to implement new optimisers. As the paper did not specify the loss function the mean squared error (MSE) was used[1] 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:

- **`value_and_gradients_function`**: A Python callable which takes a point (1D tensor) as input and returns the value of the function to be minimised and its gradient at that point.

[1]Implemented using `tf.keras.losses.MeanSquaredError()`

- **`initial_position`**: The starting position of the function.

One 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[2] were mapped to and from a 1D tensor using TensorFlows `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` and 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
- **`_make_1d_mapping(self)`**: Called by `__init__` to create a mapping to and from a 1D tensor for the model weights.

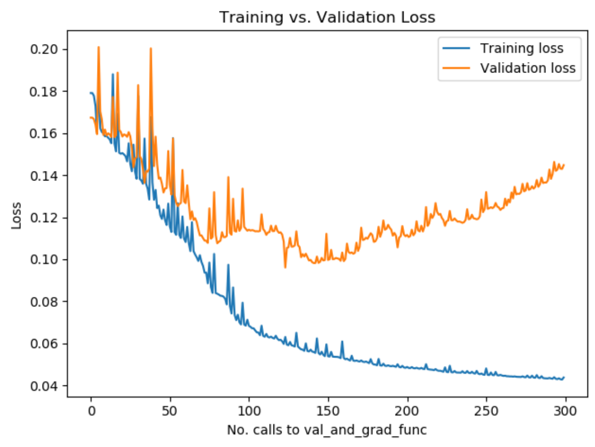
Using this model the hyper-parameter H was then tuned using 10-fold cross-validation, with the average validation accuracy for each value of H recorded (See results section). Once the best value of H was determined, the progression of the training and validation loss and accuracy over the 100 epochs. 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. 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

[2]2 weight matrices and 2 bias vectors

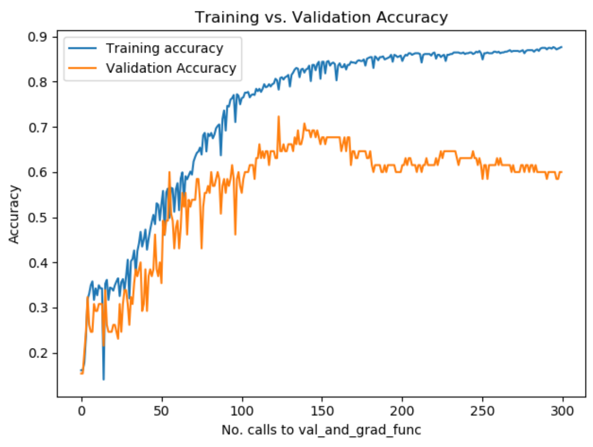
the model with each epoch, i.e. the training and validation datasets. This monitor can then be passed to the `BfgsMlp.fit function`, and uses a wrapper function `monitor(func)` to wrap the value_and_gradient function for the model. It then calls the models `evaluate(X, y)` function, and records the results 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. I could not find any reason for this either in 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 was 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, and an accuracy of 65.32% achieved.

Results



(a) Training vs. Validation loss



(b) Training vs. Validation accuracy

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

Note No. calls to val_and_grad_func is 3x No. epochs

H	0	2	4	6	8
Accuracy	28.92%	66.15%	67.85%	62.15%	64.62%

2.3.2 Support Vector Machine

Overview

Support vector machines (SVMs) 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, Figure 2.4. SVMs 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**multiclass-svm**. By themselves SVMs aren't of much use, as they can only solve linearly-separable problems. But by the use of the *'kernel trick'*[13]. This uses a kernel function to map the features into some higher dimensional space, where they are linearly-separable.

SVMs offer a number of hyper-parameters, such as: C a 'slack' variable; and γ a parameter used by the kernel functions. Only γ 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[14]. For this reason it was looked at to demonstrate Scikit-Learns various dimensionality reduction methods, and to see if it has any improvement on classification accuracy.

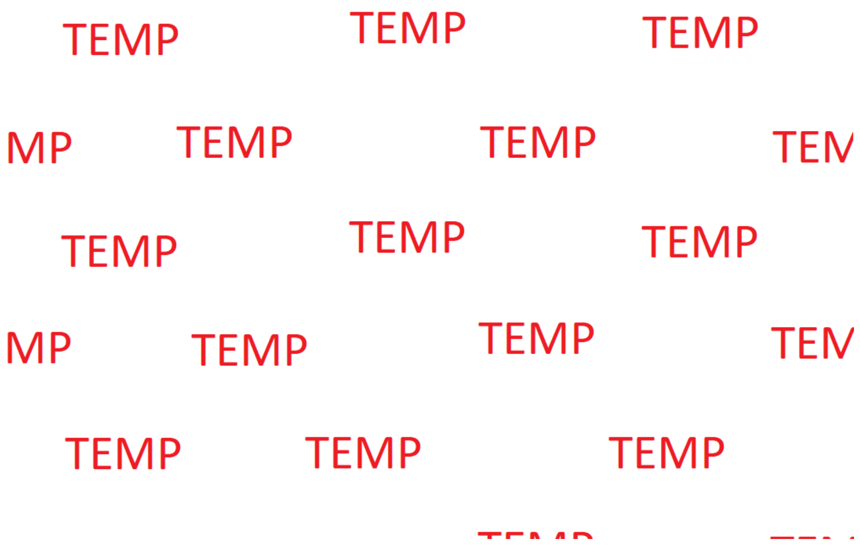


Figure 2.4: Optimal decision boundary made by SVM

Code Development

Code for this model was made using Scikit-Learn's `sklearn.svm.SVC` class. To tune the γ value, scikit-learn's `sklearn.model_selection.GridSearchCV` was used. This allows for a user to tune a models 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 model and some training data. Once training of the models is done, various information about the results can be accessed, along with the best performing model. For this grid-search, both the γ value and the kernel functions were varied. The results for this can be seen in Tables 2.3– 2.5 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[15]. 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[16]. Kaiser's stopping rule states that only factors with an eigenvalue of greater than 1.00 should be used. Table 2.2 shows the top 11 eigenvalues, of these 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, as seen in Figure 2.5, this happens after the first 3 features. An identical approach was then used to performing hyper-parameter tuning for the models using datasets of this dimensionality.

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 2.2: Factors and Eigenvalues

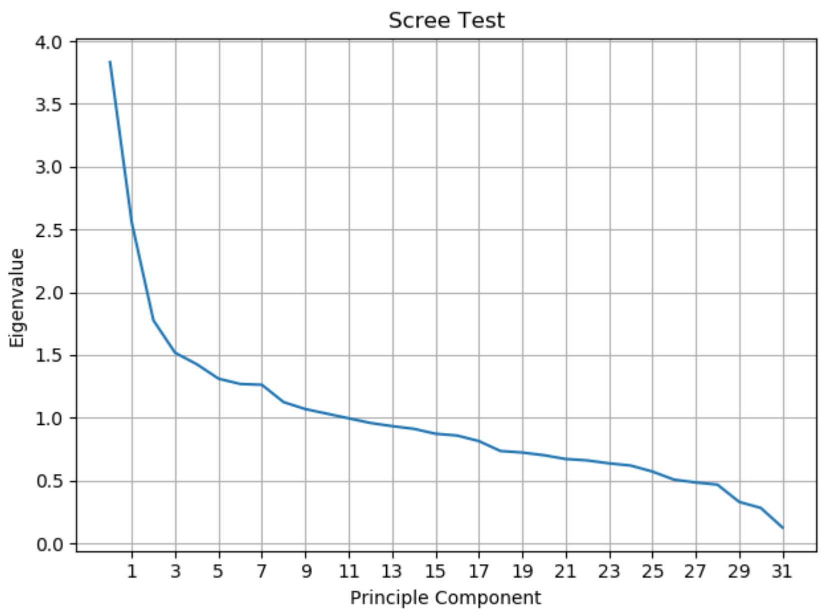


Figure 2.5: Scree Test showing factor vs. eigenvalue. Notice the drop off after 3 factors

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 γ of 0.002. It also means that γ was unused by the best model, so tuning it was not needed. It can also be seen for all data inputs, Tables 2.3 – 2.5, that for $\gamma < 2^{-5}$ no difference was seen in accuracy for any kernel.

		Gamma γ				
		2^{-9}	2^{-7}	2^{-5}	2^{-3}	2^{-1}
Kernel	**Linear**	$\mathbf{67.62\% \pm 4.8}$	–	–	–	–
	RBF	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$45.23\% \pm 6.6$	$32.37\% \pm 2.0$
	Poly	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$50.02\% \pm 6.5$
	Sigmoid	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$38.00\% \pm 4.9$	$51.73\% \pm 4.4$

Table 2.3: Results using no PCA (Best accuracy in bold)

		Gamma γ				
		2^{-9}	2^{-7}	2^{-5}	2^{-3}	2^{-1}
Kernel	**Linear**	$\mathbf{56.51\% \pm 4.8}$	–	–	–	–
	RBF	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$41.81\% \pm 5.2$	$47.93\% \pm 6.2$
	Poly	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$46.75\% \pm 7.2$
	Sigmoid	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$37.66\% \pm 4.3$	$48.29\% \pm 6.5$

Table 2.4: Results using Kaiser PC selection (Best accuracy in bold)

		Gamma γ				
		2^{-9}	2^{-7}	2^{-5}	2^{-3}	2^{-1}
Kernel	**Linear**	$45.54\% \pm 6.2$	–	–	–	–
	RBF	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$40.77\% \pm 5.1$	$\mathbf{46.39\% \pm 6.1}$
	Poly	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$42.47\% \pm 7.0$
	Sigmoid	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$31.84\% \pm 0.7$	$37.14\% \pm 3.9$	$33.08\% \pm 6.1$

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

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. This is because as seen Figure 2.5, the majority of principle components had an eigenvalue of greater than 0.5, and thus were non-negligible in terms of the amount of variance they represented. Thi can be seen more clearly in Figure 2.6, which shows that each feature explains about an equal percent of the variance, indicated by the plot being close to linear.

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.

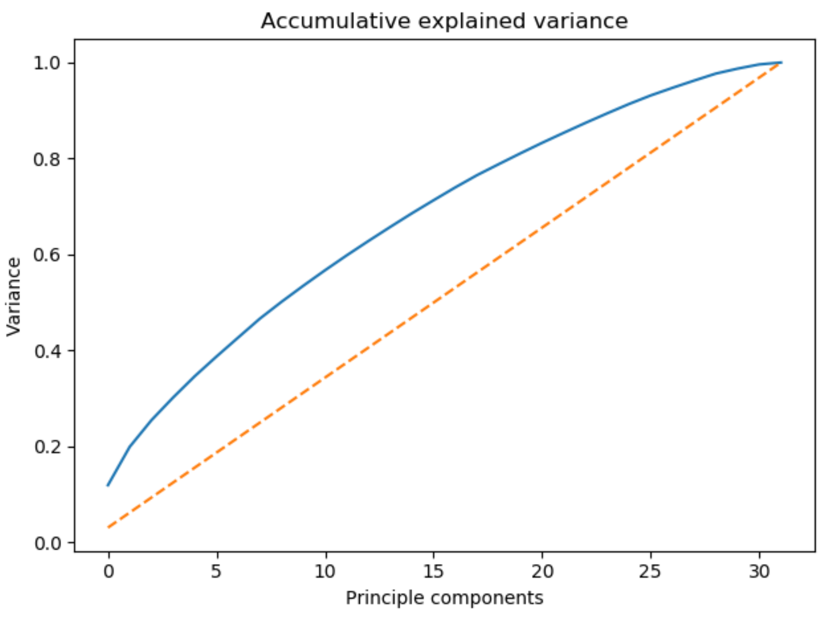


Figure 2.6: Plot showing accumulative explained variance for the principle components (Blue) Orange represents perfectly equal explained variance by each PC

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[13]. 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[17] or CART[18]. 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 is split in half[19].

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:

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

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 is the total misclassification rate of tree T, and $|T|$ is the number of terminal nodes in T.

$$R_\alpha(T) = R(T) + \alpha|T| \tag{2.2}$$

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 α value. The optimal α value was calculated as follows:

1. Using `DecisionTreeClassifier.cost_complexity_pruning_path(X, y)` the effective α of each node in a tree can be calculated. This is the α 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 α, as seen in Figure 2.7, the depth and number of nodes quickly drop off as α increases.

3. Using these models we can also plot the training and validation accuracy against α, as seen in Figure 2.8. From this the best α to prevent over-fitting can be seen to be around 0.005. This value maximises validation accuracy, and as seen in Figure 2.7 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 2.9 and 2.10. 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[3]. 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.

[3]A full page version of these trees can be seen in the appendix, as they are quite large

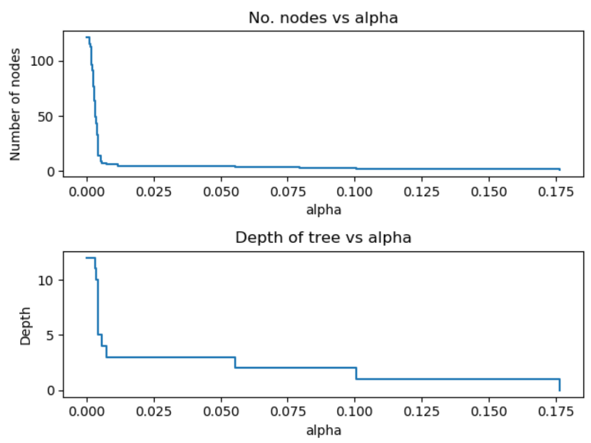


Figure 2.7: α vs. Size of tree

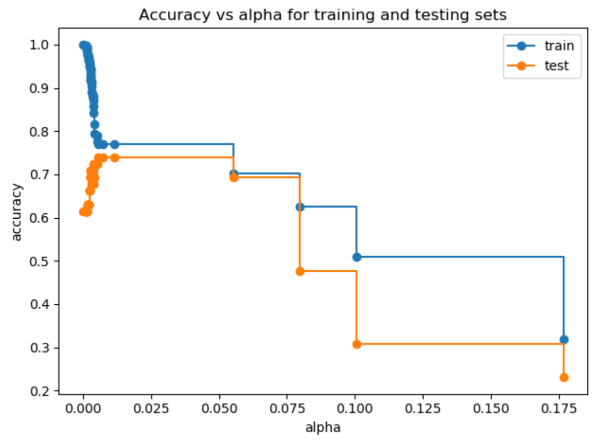


Figure 2.8: α vs. Training and Validation accuracy

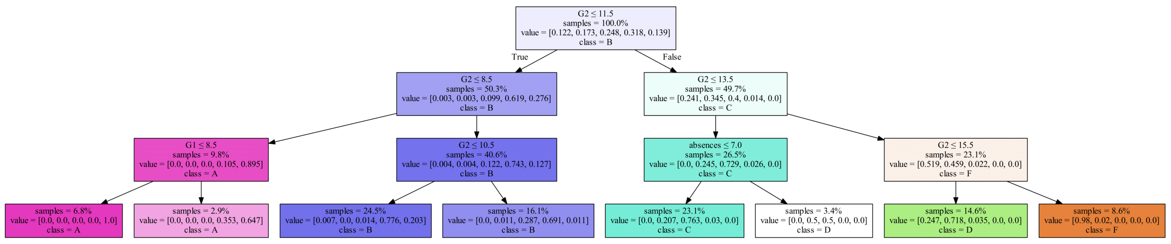

G2 ≤ 11.5
samples = 100.0%
value = [0.122, 0.173, 0.248, 0.318, 0.139]
class = B
True
False
G2 ≤ 8.5
samples = 50.3%
value = [0.003, 0.003, 0.099, 0.619, 0.276]
class = B
G2 ≤ 13.5
samples = 49.7%
value = [0.241, 0.345, 0.4, 0.014, 0.0]
class = C
G1 ≤ 8.5
samples = 9.8%
value = [0.0, 0.0, 0.0, 0.105, 0.895]
class = A
G2 ≤ 10.5
samples = 40.6%
value = [0.004, 0.004, 0.122, 0.743, 0.127]
class = B
absences ≤ 7.0
samples = 26.5%
value = [0.0, 0.245, 0.729, 0.026, 0.0]
class = C
G2 ≤ 15.5
samples = 23.1%
value = [0.519, 0.459, 0.022, 0.0, 0.0]
class = F
samples = 6.8%
value = [0.0, 0.0, 0.0, 0.0, 1.0]
class = A
samples = 2.9%
value = [0.0, 0.0, 0.0, 0.353, 0.647]
class = A
samples = 24.5%
value = [0.007, 0.0, 0.014, 0.776, 0.203]
class = B
samples = 16.1%
value = [0.0, 0.011, 0.287, 0.691, 0.011]
class = B
samples = 23.1%
value = [0.0, 0.207, 0.763, 0.03, 0.0]
class = C
samples = 3.4%
value = [0.0, 0.5, 0.5, 0.0, 0.0]
class = D
samples = 14.6%
value = [0.247, 0.718, 0.035, 0.0, 0.0]
class = D
samples = 8.6%
value = [0.98, 0.02, 0.0, 0.0, 0.0]
class = F

Figure 2.9: Best decision tree without pruning

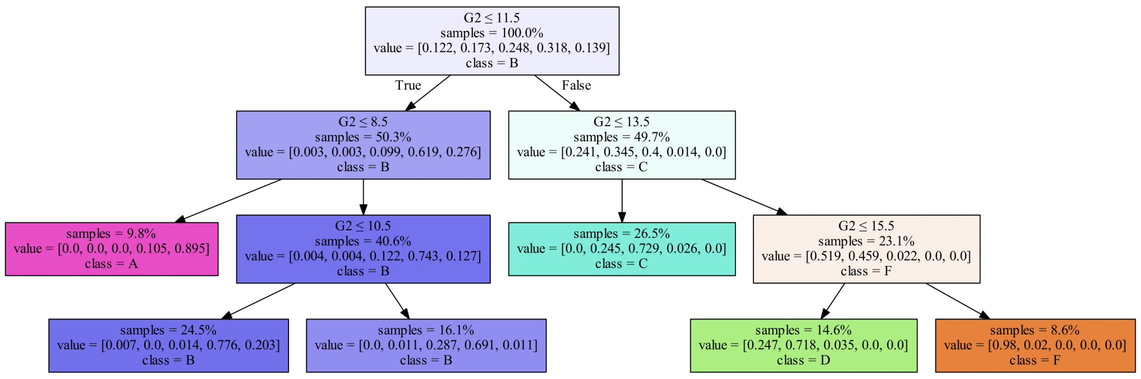

G2 ≤ 11.5
samples = 100.0%
value = [0.122, 0.173, 0.248, 0.318, 0.139]
class = B
True
False
G2 ≤ 8.5
samples = 50.3%
value = [0.003, 0.003, 0.099, 0.619, 0.276]
class = B
G2 ≤ 13.5
samples = 49.7%
value = [0.241, 0.345, 0.4, 0.014, 0.0]
class = C
samples = 9.8%
value = [0.0, 0.0, 0.0, 0.105, 0.895]
class = A
G2 ≤ 10.5
samples = 40.6%
value = [0.004, 0.004, 0.122, 0.743, 0.127]
class = B
samples = 26.5%
value = [0.0, 0.245, 0.729, 0.026, 0.0]
class = C
G2 ≤ 15.5
samples = 23.1%
value = [0.519, 0.459, 0.022, 0.0, 0.0]
class = F
samples = 24.5%
value = [0.007, 0.0, 0.014, 0.776, 0.203]
class = B
samples = 16.1%
value = [0.0, 0.011, 0.287, 0.691, 0.011]
class = B
samples = 14.6%
value = [0.247, 0.718, 0.035, 0.0, 0.0]
class = D
samples = 8.6%
value = [0.98, 0.02, 0.0, 0.0, 0.0]
class = F

Figure 2.10: Best decision tree with pruning

Results

The results from the hyper-parameter grid-search using 10-fold cross-validation can be seen below, Table 2.6. From this a number of observations can be made about the effect of both the max depth and α 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	α	**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%
3	**0.005**	**0.0025**	**76.34%**	**5.8%**
5	0.005	0.0032	73.93%	7.2%
–	0.005	0.0050	72.38%	9.1%

Table 2.6: 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[4], each classifying the data in a different way, and combining there predictions using some algorithm[13]. 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 will take significant time, it was decided to also seem how limiting the size using both `max_depth` and the `min_samples_leaf` of the trees affected the accuracy. This was checked using a grid-search method.

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 to-

[4]A weak learner is a model whose accuracy is slightly better than chance

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	71.32%	6.6%

tal dataset. Here the hyper-parameters to be tuned were `n_estimators` $\in \{100, 500\}$, `max_depth` $\in \{3, \texttt{None}\}$ and `nim_samples_leaf` $\in \{1, 0.05\}$.

Results

2.3.5 Comparison of Results

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 causesource. 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[20], 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. More prevalent in children and elderly.

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[21]. There has been a great amount of research into the use of deep learning in medical diagnosis in recent years, in particular for use image-based diagnosis, such as MRI, CAT or X-Rays[22]. Examples of ML for diagnosis. 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[22].

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 $m \in \mathbb{R}^{w \times h \times d}$[23], 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.

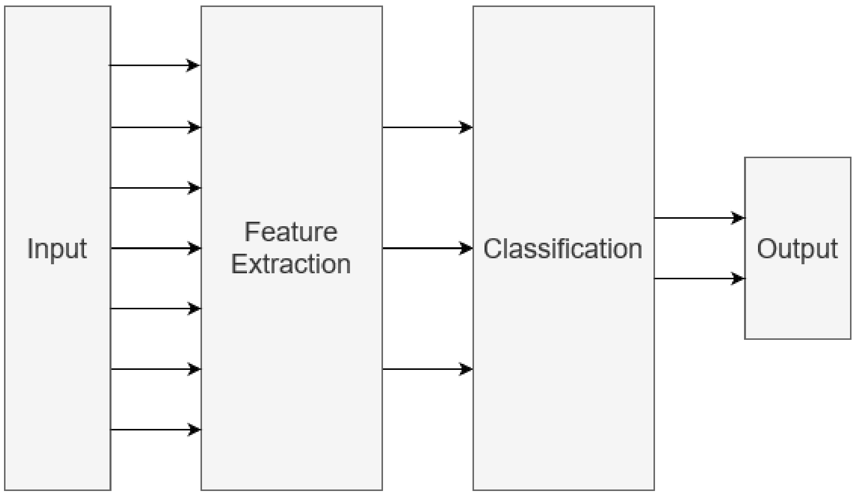


Figure 3.1: Block diagram of typical CNN

Despite being shown to be effective at solving machine vision tasks and fully trainable by back-propagation since 1989[24], 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[25] 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[23]. 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[26]. 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 $k_w, k_h \in \mathbb{N}_{\geq 1}$[23]. 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 hyperparmeters 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 separably, 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[27]. 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.[23].

3.2.3 Normalization Layers

3.3 Dataset

Images from dataset, Examples of pneumonia vs normal, etc. It uses chest x-rays from the *Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification* dataset[5][1] produced by Daniel Kermany, Kang Zhang and Michael Goldbaumusing 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 50% of the test presentations.

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

[1]Version 2

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 256×256.

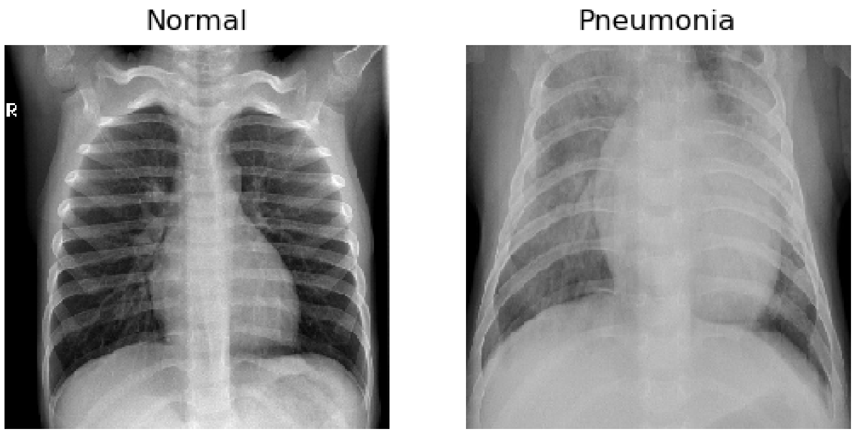


Figure 3.2: Example of x-ray with and without pneumonia

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 similar model on kaggle. This was composed of 3 convolutional layers, with a 3×3 kernel size, and rectified linear activation function, each followed by a max-pooling layer, with a 2×2 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.

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 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.

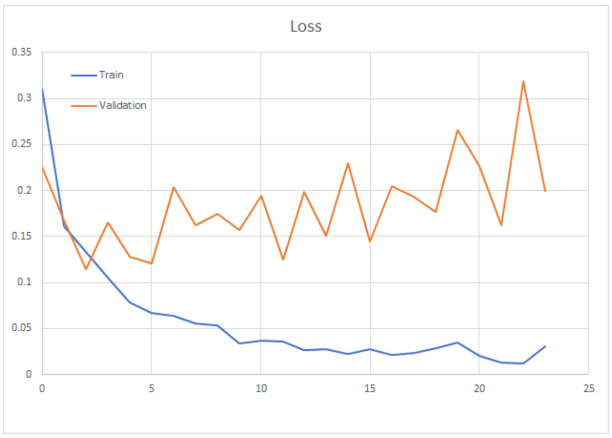


Figure 3.3: Training and validation loss

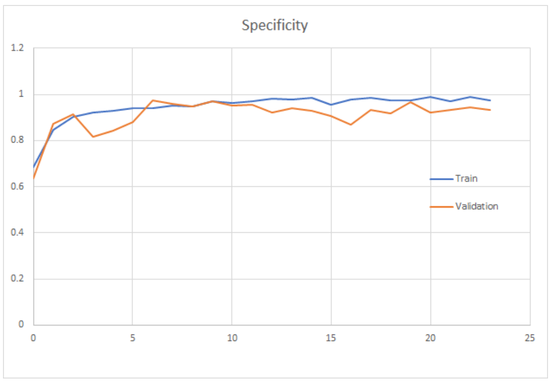


Figure 3.4: Training and validation specificity

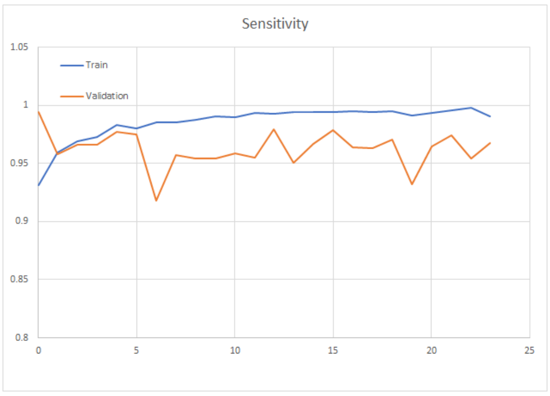


Figure 3.5: Training and validation sensitivity

3.4.2 Image Augmentation

Image augmentation is the find word, technique? of performing transformations on the training images, to try stop the model from learning the *'noise'* in the dataset, and instead learn the desired featuressource. 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. The Keras *ImageDataGenerator* has a number of arguments that can be used to apply transformations to images as they are presented, such as:

- 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

- 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.
- Mirroring; The image may 50% of the time be mirrored around either the vertical or horizontal axis using the `horizontal_flip` and `vertical_flip` keyword arguments.
- maybe add in shear, etc.

By using these, the performance of the simple CNN described previously can be marginally (?) improved.
For this example problem, both vertical and horizontal shifts were applied, as well as mirroring across the vertical axis. These transformations were chosen as x-rays may not be centred, so one should still create a plausible input. Similarly as there is no difference between pneumonia in the left and the right lung, flipping the image vertically should not matter. info about this training

3.4.3 Transfer Learning

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. Pittc, "A logical calculus of the ideas imminent in nervous activity. bulletin of mathematical biophisics 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] S. Marsland, *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2014.

[14] 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.

[15] M. E. Tipping and C. M. Bishop, "Mixtures of probabilistic principal component analyzers," *Neural computation*, vol. 11, no. 2, pp. 443–482, 1999.

[16] J. Brown, "Choosing the right number of components or factors in pca and efa," *JALT Testing & Evaluation SIG Newsletter*, vol. 13, no. 2, 2009.

[17] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[18] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[19] 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.

[20] *The european lung white book; respiratory health and disease in europe*, `https://www.erswhitebook.org/chapters/acute-lower-respiratory-infections/`, Accessed: 2020-02-03.

[21] 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.

[22] 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.

[23] M. Thoma, “Analysis and optimization of convolutional neural networkarchitectures,” Masters?s Thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany, Jun. 2017. [Online]. Available: `https://martin-thoma.com/msthesis/`.

[24] 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.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” pp. 1097–1105, 2012.

[26] 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.

[27] 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.