Student Number: 21906942 & Kaggle Team Name: kebabgage

1. APPROACH

To solve the binary classification problem, a Multi-Layer Perceptron was employed to train on the training data and predict the classes of the test data.

A Multi-Layer Perceptron describes a network of many perceptron classifiers, a supervised method of machine learning capable of deciding whether an input value belongs to a specified class. The decision, or output, of a perceptron is driven by the input weights and the specified non-linear activation function [1].

In a Multi-Layer Perceptron, the network is comprised of multiple layers, with each layer containing a row of perceptions. Each node in one layer connects and passes on a certain weight to every node in the following layer. The first layer of a Multi-Layer Perceptron, the input layer, accepts values within a data set and passes each value onto the next layer. The final layer, the output layer, produces a number of values that correspond to the data set that was input in a form that is denoted by an activation function of the output layer. The layers between the input and output layers are not directly exposed to the input and are instead passed weighted input signals and output signals denoted by the activation function of the layer.

2. METHODOLOGY

2.1. Data Preprocessing

Both the training and the test datasets were standardized by using the `StandardScaler` class from the `sci-kit learn` python library [2]. This was done to ensure that the Multi-Layer Perceptron can estimate more effectively, as the data is closer to a standard normally distributed dataset.

2.2. Model Selection and Validation

The Multi-Layer Perceptron was developed using `Keras`, a neural-network python library using `TensorFlow` as the back-end [3, 4]. The model was trained using a batch-size of 10 and was trained across 150 epochs. An early stopping callback was utilized to ensure that the model was not overfitted through training. The model was to stop training after 50 epochs of no change.

In the development of the model, different combinations of layer types with varying activation functions were tested, with the loss and accuracy of the measured and compared. Activation functions applied to the perceptron layers employed were *rectified linear unit (ReLU)* and *sigmoid* functions. As the training and test datasets contain only two labels - a binary classification model. Thus, a binary cross-entropy function was used to measure loss within the model. Training of the machine learning model was computationally restrained, thus the optimizer chosen was an `Adam` optimizer - a computationally efficient optimizer [5]

Due to the relatively small size of the training dataset, the model was cross validated using k-fold cross-validation - more specifically 10-fold cross validation. This was done to ensure that estimates of the model during development were not biased and were continually tested on varying test sets.

2.2.1 Additional Training Data

Additional training data was made available, however, there was missing values within this dataset. These missing values made the data incompatible with the Multi-Layer Perceptron. If rows and columns that contain missing values were merely discarded, this would result in the entire additional dataset being abandoned.

To combat this whilst conserving the data, the `SimpleImputer` class from the `sci-kit learn` python library was used to infer missing values within the dataset. The values were imputed using the mean of each column.

The imputation of missing values in the additional dataset increases size of the training dataset from 247 to 2466.

3. RESULTS

Several models of increasing layer complexity and layer size were compiled. The Perceptron count for each layer and the activation functions of each layer are detailed in Table 1. 10-Fold Cross Validation of each of the models was performed and the accuracy and loss were compared.

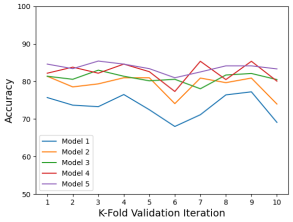
Layer	Model 1	Model 2	Model 3	Model 4	Model 5
1	4 (*Sigmoid*)	4 (*ReLU*)	8 (*ReLU*)	16 (*ReLU*)	500 (*ReLU*)
2	1 (*Sigmoid*)	1 (*ReLU*)	4 (*ReLU*)	8 (*ReLU*)	250 (*ReLU*)
3			1 (*Sigmoid*)	4 (*ReLU*)	250 (*ReLU*)
4				1 (*ReLU*)	1 (*ReLU*)

Table 1: The layers of each Multi-Layer Perceptron model that was trained and validated through K-Fold Cross Validation. Each layer is described by the Perceptron count total contained in each layer and the activation function that was applied to each Perceptron.

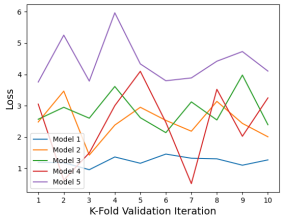
Each of the five models were compiled and trained. The average accuracy and loss of each of the five models is sum-

Model	Accuracy		Loss	
	mean	*st. dev.*	*mean*	*st. dev.*
1	73.36	3.04	1.23	0.14
2	79.08	2.66	2.50	0.55
3	80.94	1.27	2.85	0.54
4	82.40	2.44	2.40	1.15
5	83.66	1.19	4.40	0.69

Table 2: The average accuracy and loss across all cross validation splits of each test model at the conclusion of 150 epochs.



(a) Accuracy of Models per Validation



(b) Loss of Models per Validation

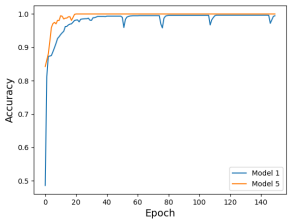
Figure 1: The accuracy and loss of each of the five models across the 10-Fold Cross Validation process.

marized in Table 2; the loss and the accuracy comparison is visualized in Figure 1. The simplest Multi-Layer Perceptron, *Model 1*, proved to be the model that produced the least amount of loss, on average, across the 10-Fold Cross Validation. The larger and most complex Multi-Layer Perceptron, *Model 2*, produced the best result in accuracy on average across the 10-Fold Cross Validation.

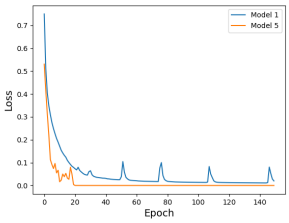
Model 1 and Model 5 were closely examined to identify potential over fitting tendencies. This was done by investigating the accuracy and loss across the total Epochs that the model was trained. This is visualised in Figure 2. As we want to maximise accuracy, and the tendency of decreased loss is common across both models, Model 5 seems to be the better model. However, at the 20th Epoch, the accuracy plateaus at 100%. This is indicative that the model begins to become too familiar with the dataset it is being trained on and may lose the capability to generalize.

The inability for the model to generalize and be over-fit on the training data loses sight of the aim of model: to predict and classify objects from an unseen domain. To combat this potential over-fitting, an earlier stopping monitor was placed on the model. This will prevent the model from over training and allow the test data to be better predicted on the whole.

Through the development of the model, technological limitations (hard-drive size) prevented the direct application of specific libraries that can solve domain adaptation. Thus, it was paramount to reduce over fitting the model to



(a) Accuracy of Model per Epoch



(b) Loss of Model per Epoch

Figure 2: Accuracy and loss metrics collected throughout the training duration, for Model 1 and 5. Each metric was calculated at each Epoch that the model was trained.

the training dataset. The test dataset contains data that is derived from a different domain and an overfitted model may not perform as strongly with the test dataset.

Due to the time constraints of this investigation, a more fine tuned Mutli-Layer Perceptron may have been identified. This would require more rigorous testing and evaluation of the performance of the model. In addition to this, although use of a Multi-Layer Perceptron was suitable, further solutions to this problem may include different machine learning models.

Model	Accuracy		Loss	
	mean	*st. dev.*	*mean*	*st. dev.*
1	73.36	3.04	1.23	0.14
2	79.08	2.66	2.50	0.55
3	80.94	1.27	2.85	0.54
4	82.40	2.44	2.40	1.15
5	83.66	1.19	4.40	0.69

Table 3: The average accuracy and loss across all cross validation splits of each test model at the conclusion of 150 epochs.

References

[1] Yang, Z. R. (2010). Multi-layer Perceptron. In *Machine Learning Approaches to Bioinformatics* (pp. 133-153) World Scientific Publishing Co. Pte. Ltd. https://doi.org/10.1142/9789814287319$_0$0101

[2] Pedregosa *et al.* (2011). *Scikit-learn: Machine Learning in Python* (pp. 2825-2830). JMLR 1

[3] Chollet, François *et al.* (2015). *Keras*. https://keras.io 1

[4] Abadi, Martin *et al.* (2015). *Large-Scale Machine Learning on Heterogeneous Distributed Systems* https://www.tensorflow.org/ 1

[5] Kingma, Diederik P. and Ba, Jimmy. (2014) *Adam: A Method for Stochastic Optimization* 1