BT4012 Assignment 2 report

Data processing

Initial investigations into the dataset revealed that the training dataset is significantly unbalanced[1], and hence either undersampling or oversampling was required. Since the dataset is not that large, the decision was made to do oversampling. I utilised SMOTE (Synthetic Minority Oversampling Technique), which works by utlising the k nearest neighbour (KNN) algorithm to create synthetic data. SMOTE first starts by choosing data at random from the minority, which are all the positive cases, then the k nearest neighbours are set. Synthetic data would then be made between the random data and the randomly selected k-nearest neighbour.

Feature Engineering

In order to transform the data into a form that is accepted into a CNN, the row of data is then reshaped from a single row into a 2D numpy array. The arrays were then normalised by dividing the values by 255, which is the maximum possible value for a light channel in the RGB spectrum.

Model Building

Model initially started with various low level machine learning algorithms such as logistic regression, XGboost, SVM. Attempts at stacking various models together for better performance were made before realising that further attempts proved fruitless as performance gains from such endeavours are minimal and do not reach the baseline model. Therefore, a switch to CNNs was made.

The model is relatively simple, made up of layers of convolutional 2D layers with differing kernel sizes ranging from (2,2) to (5,5). They all have an activation function of ReLU and padding around the sides in order to ensure that the sides of the image are not fully neglected when traversing across the image with. The number of filters in the CNN model is 16, which is the optimal number of filters used based on testing so far.

Validation

Validation was done using a train test split, where some of the dataset was not used for training purposes. The variation in AUC for train data and test data differs by 0.01, which is quite significant in the final model, although I was unsure of how I could further improve my model to reach more than 0.991 AUC score.

Model Selection

In order to select the best model possible, various hyperparameters of the CNN were played around with, in order to ensure the best model was selected. Initial number of filters was 4, before subsequent tests showed that there were improvements in the AUC score when the number of filters increased, up till a certain point, which in this case for me it was 16 filters. I originally intended to add a grid search to determine the kernel initialisers but I did not have enough time to implement it.

Kernel size was experimented with, and the more kernels meant more parameters to process. Kernels of sizes (2,2) and (3,3) were targeted at examining the small details, while kernels of sizes (4,4) and (5,5) were targeted at examining the larger details. Padding was also done to the images so as to ensure that the pixels in the corners and edges do not get neglected and used less. This improved the AUC score by 0.01-0.02 compared to without adding padding.

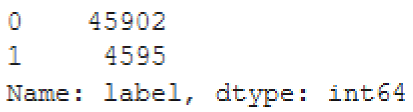
Pooling sizes were experimented with, along with average pooling and max pooling. Max pooling provided better results as it was better able to identify the key features that reflected a positive label, and resulted in a higher AUC score.

Various activation functions were experimented with, but ReLU was found to be the best one, and dropout value of 0.5 worked to curb overfitting the train dataset. The output layer of sigmoid activation function was the optimal choice based on various activation functions tested to give a binary result.

Adding a fully connected layer of 100 neurons was the optimal as tests of 50 and 150 neurons resulted in the model underfitting or overfitting, by comparing the differences in the AUC score.

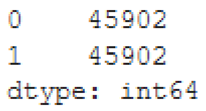
Appendix

Unbalanced Dataset



```
0    45902
1     4595
Name: label, dtype: int64
```

Figure 1: Before oversampling with SMOTE



```
0    45902
1    45902
dtype: int64
```

Figure 2: After oversampling with SMOTE