DEEP LEARNING ASSIGNMENT

Sentiment Analysis of Tweets using CNN

Name: Arjun Mallick
Roll: 001910501021
Class: BCSE , 7th Sem

—---

Dataset Description :

The dataset consists of 14640 airline tweets in the English Language. Besides the tweet text, its demographic, authors, retweet count and name of airline are present. The tweet text itself contains mentions and hashtags. Each tweet has been classified into positive, negative or neutral sentiment.
The class wise distribution of the tweet are given as :

Negative: 9178 (63%)
Neutral: 3099 (21%)
Positive: 2363 (16%)

The percentage of tweets in every class shows a skewness present towards the negative class.
So, stratified sampling is applied on the dataset to get the test set(20%). The distribution of tweets in each of them are:
Train set:

Negative	7343 (63%)
Neutral	2479 (21%)
Positive	1890 (16%)

Test Set :

Negative	1835 (63%)
Neutral	620 (21%)
Positive	473 (16%)

Pre-processing :

In this stage, different variations of pre-processing steps have been experimented on the text data. The tweet texts have been initially converted to lowercase characters & all kinds of line breaks have been removed.
This was followed by stopwords removal. However, the important punctuations have been kept(. , ; ? ! / etc).

Variations in the text data like removal of the mentions, hashtags or both to get the best possible text data for the CNN model have been tested.

The CNN model needs all the data to be of the same length. However the tweets present may not be so. Thus, there was a need for padding each text sequence. The maximum length of any text sequence in the dataset was 148. So, a padding of blank spaces till the length of all the tweets becomes 160 has also been tried. Different lengths of the blank spaces have been experimented with.

Input Representation :

The input to the CNN is a one hot encoded matrix for a single tweet. The matrix is obtained by taking the one hot encoding vector for each character in the tweet text and concatenating them to form a matrix representation for each tweet. Since the tweets have been padded to be of the same length(160), the matrix representation is consistent throughout. For encoding purposes, the lowercase ascii, digits and punctuations have been considered.
Given all these data, the dimensions of the training set is: **(11712, 160, 68)** and that of the test set is : **(2928, 160, 68).**

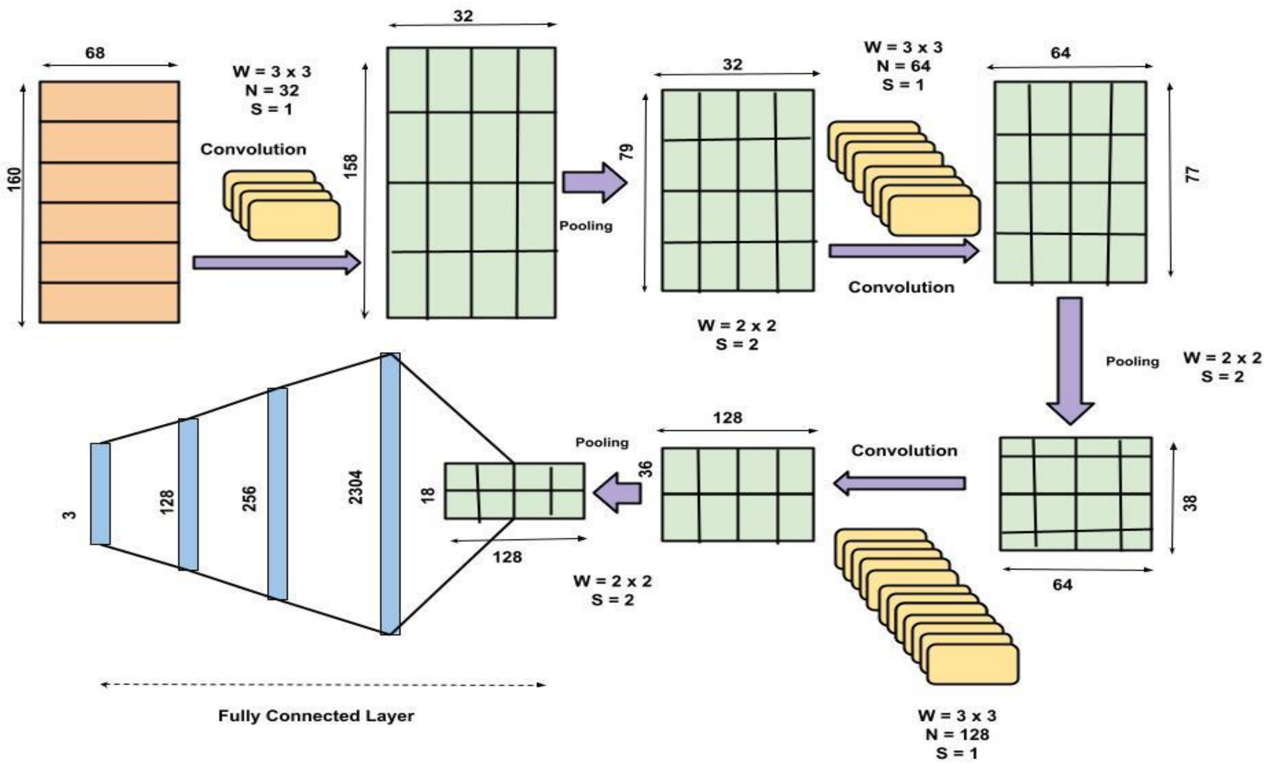
Model Architecture :

The CNN model implemented in this work consists of three layers of convolution and max pooling. The output from the convolution layers is flattened to obtain a 1D feature representation followed by fully connected layers (with dropout regularisation) for classification purposes.

The convolution layers contain 32, 64 and 128 filters respectively each with a kernel size of 3 x 3 and stride = 1. The Max pooling layers are consistent throughout with neighbours of 2 x 2 and stride = 2. Each convolution operation is followed by a non-linear ReLU activation before passing it through max-pooling operation.

The fully connected layers have 128 and 256 hidden nodes followed by 3 output nodes with softmax activation. The hidden nodes of the fully connected layers also have ReLU activation. A dropout rate of 20% is applied to the first connection and 25% in between the second connection.

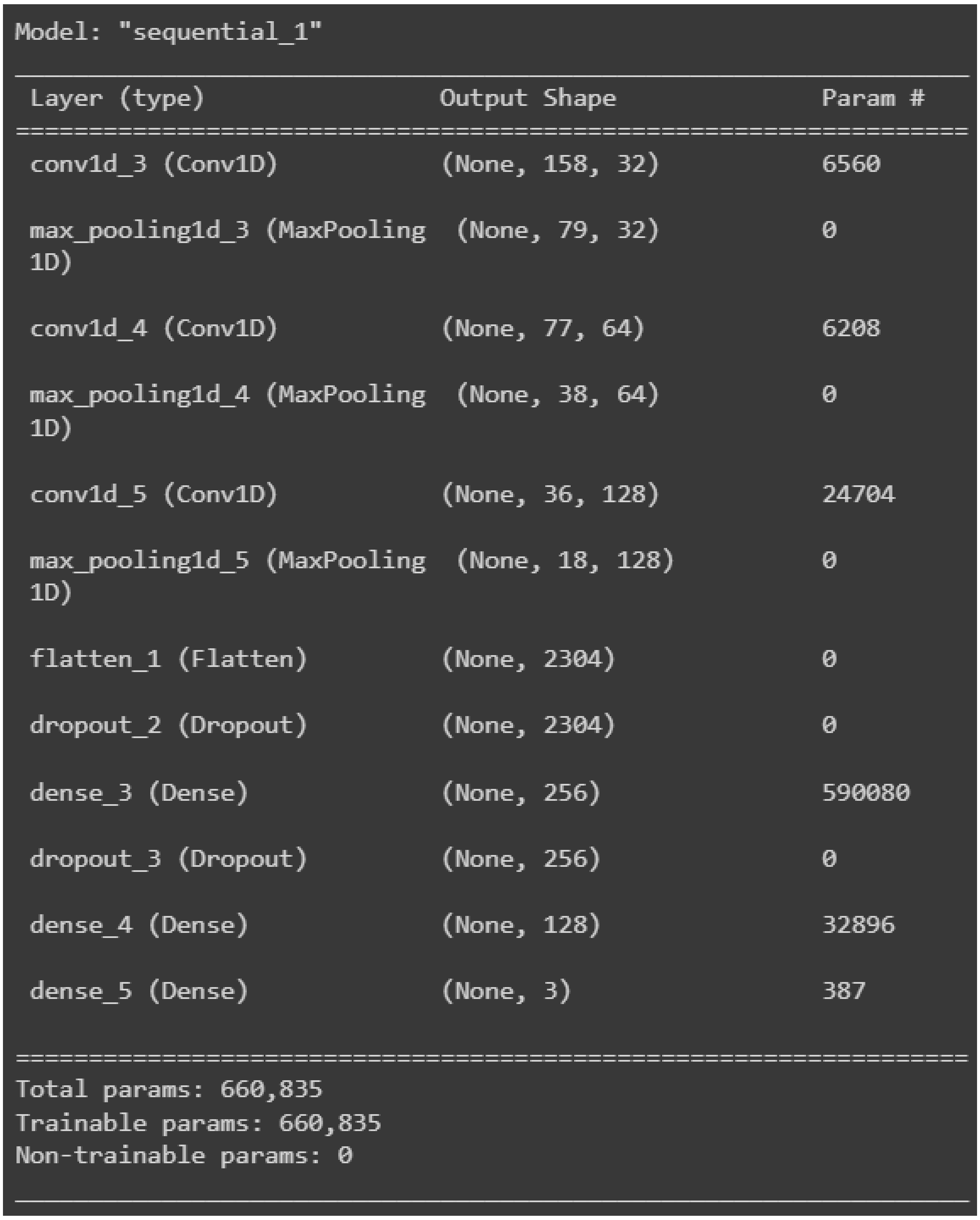
The block diagram for the model architecture is as shown :



Architecture overview of the model used

W : Width of the filter
N : Number of filters
S : Stride size

The model summary is given by :



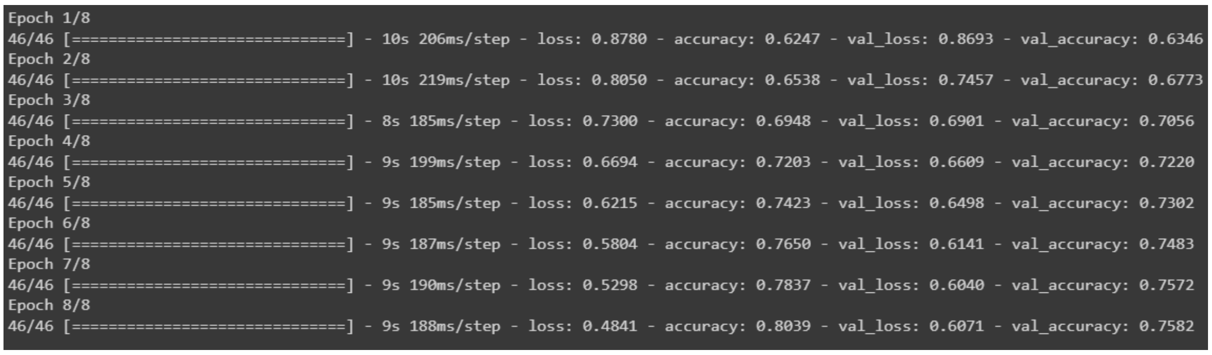
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_3 (Conv1D)           (None, 158, 32)           6560

 max_pooling1d_3 (MaxPooling  (None, 79, 32)           0
 1D)

 conv1d_4 (Conv1D)           (None, 77, 64)            6208

 max_pooling1d_4 (MaxPooling  (None, 38, 64)           0
 1D)

 conv1d_5 (Conv1D)           (None, 36, 128)           24704

 max_pooling1d_5 (MaxPooling  (None, 18, 128)          0
 1D)

 flatten_1 (Flatten)         (None, 2304)              0

 dropout_2 (Dropout)         (None, 2304)              0

 dense_3 (Dense)             (None, 256)               590080

 dropout_3 (Dropout)         (None, 256)               0

 dense_4 (Dense)             (None, 128)               32896

 dense_5 (Dense)             (None, 3)                 387

=================================================================
Total params: 660,835
Trainable params: 660,835
Non-trainable params: 0
_________________________________________________________________
```

Hyperparameter Tuning :

- The number of CNN layers have been varied from 2 to 4. With 4 layer deep CNN, there was a bit of vanishing gradient problem as the model was giving bad training accuracy.
- Among the optimizers : Adam and RMSprop, Adam seemed to provide better performance even though RMSprop was less prone to overfitting.
- Kernel Sizes and Number of Filters: Different kernel sizes, (3 x 3, 5 x 5 etc), and the number of filters have also been varied from 16 to 256.
- Pooling size: Filter size: 2 x 2, stride = 2.
- Number of epochs : seemed to converge around 6 to 7 epochs and started to overfit after that. Here, it has been trained for 8 epochs.
- Dropout rate: 20% in the flattened layer and 256 hidden node layer & 25% between 256 and 128 hidden node layer.
- Batch size: Experimented with 128 and 256 sized batches and had a more stable convergence with 256 sized batches.

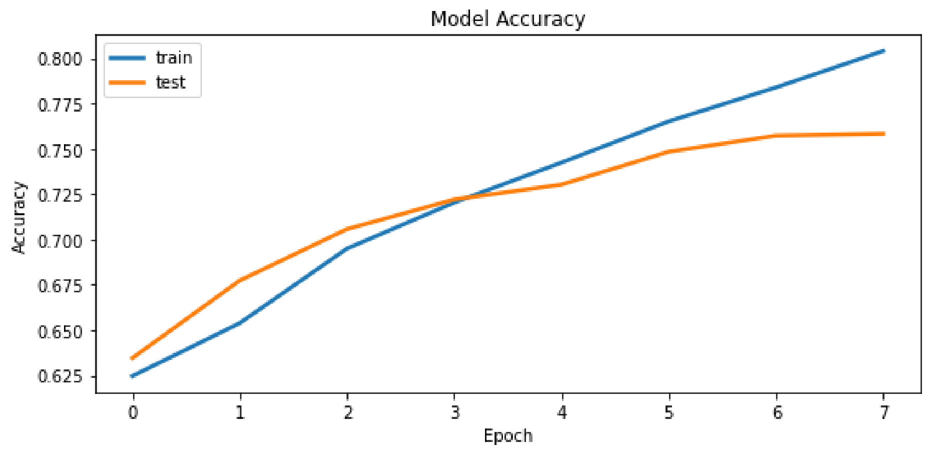
Results :

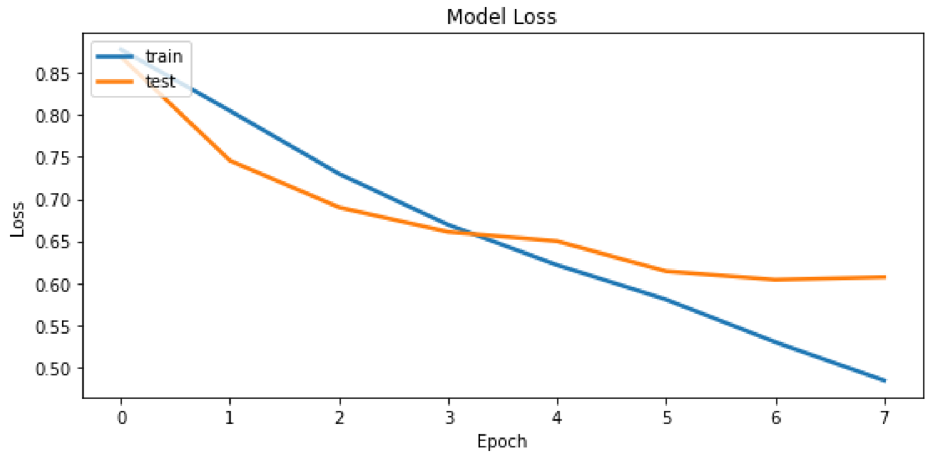
The results across epochs are given below :



```
Epoch 1/8
46/46 [==============================] - 10s 206ms/step - loss: 0.8780 - accuracy: 0.6247 - val_loss: 0.8693 - val_accuracy: 0.6346
Epoch 2/8
46/46 [==============================] - 10s 219ms/step - loss: 0.8050 - accuracy: 0.6538 - val_loss: 0.7457 - val_accuracy: 0.6773
Epoch 3/8
46/46 [==============================] - 8s 185ms/step - loss: 0.7300 - accuracy: 0.6948 - val_loss: 0.6901 - val_accuracy: 0.7056
Epoch 4/8
46/46 [==============================] - 9s 199ms/step - loss: 0.6694 - accuracy: 0.7203 - val_loss: 0.6609 - val_accuracy: 0.7220
Epoch 5/8
46/46 [==============================] - 9s 185ms/step - loss: 0.6215 - accuracy: 0.7423 - val_loss: 0.6498 - val_accuracy: 0.7302
Epoch 6/8
46/46 [==============================] - 9s 187ms/step - loss: 0.5804 - accuracy: 0.7650 - val_loss: 0.6141 - val_accuracy: 0.7483
Epoch 7/8
46/46 [==============================] - 9s 190ms/step - loss: 0.5298 - accuracy: 0.7837 - val_loss: 0.6040 - val_accuracy: 0.7572
Epoch 8/8
46/46 [==============================] - 9s 188ms/step - loss: 0.4841 - accuracy: 0.8039 - val_loss: 0.6071 - val_accuracy: 0.7582
```

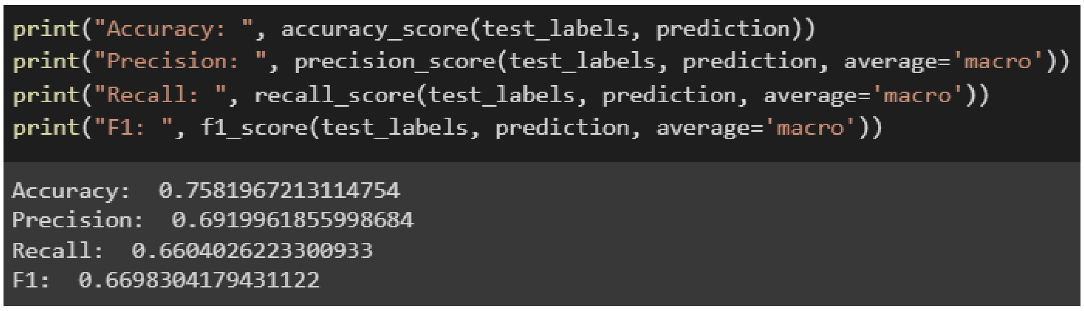
The training and validation accuracy and loss across epochs are given below :





Thus, more epochs would have led to model overfitting.

The accuracy, precision, recall and F1 score are given below. They have been micro-averaged :



```
print("Accuracy: ", accuracy_score(test_labels, prediction))
print("Precision: ", precision_score(test_labels, prediction, average='macro'))
print("Recall: ", recall_score(test_labels, prediction, average='macro'))
print("F1: ", f1_score(test_labels, prediction, average='macro'))
```

```
Accuracy:  0.7581967213114754
Precision:  0.6919961855998684
Recall:  0.6604026223300933
F1:  0.6698304179431122
```

Arjun Mallick

Signature