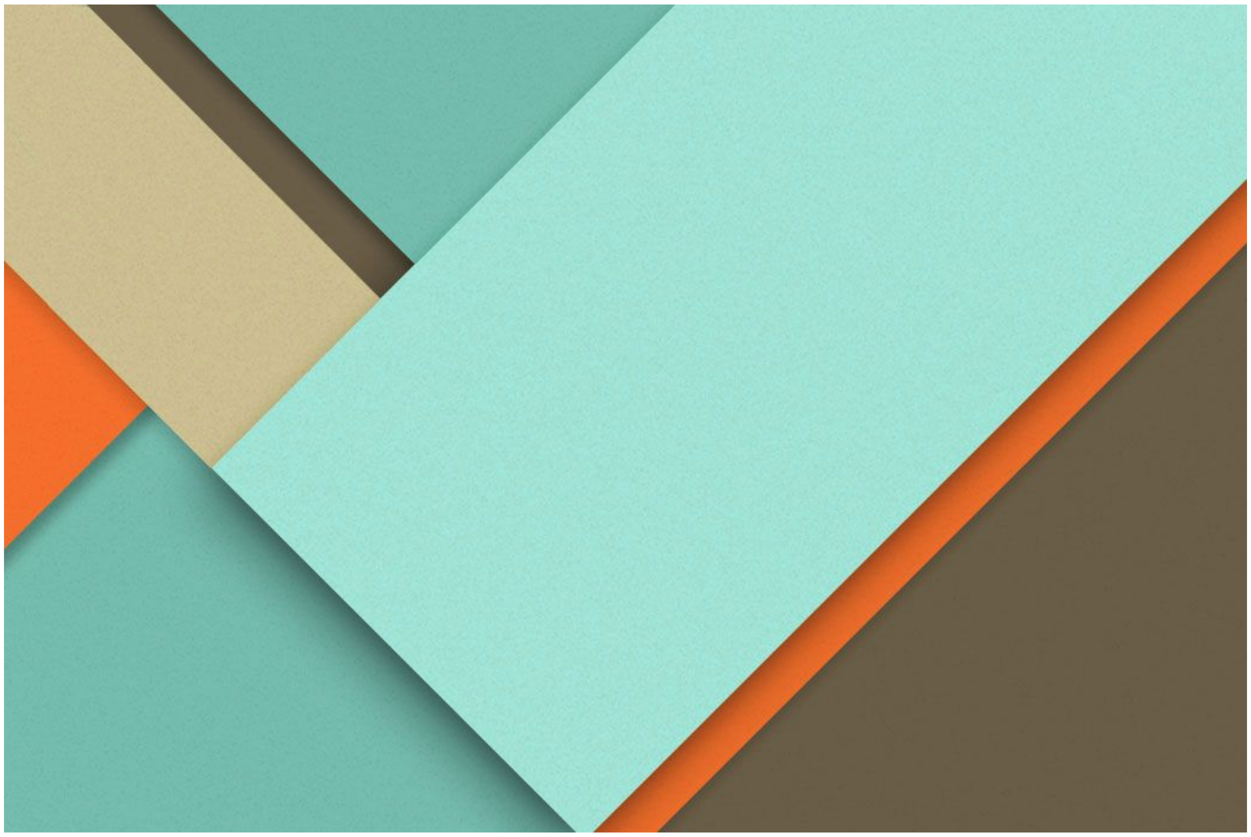


TEXT CLASSIFICATION

BBC NEWS

SAKSHI GATYAN

sakshigatyan1998@gmail.com
Ph: 9078067465

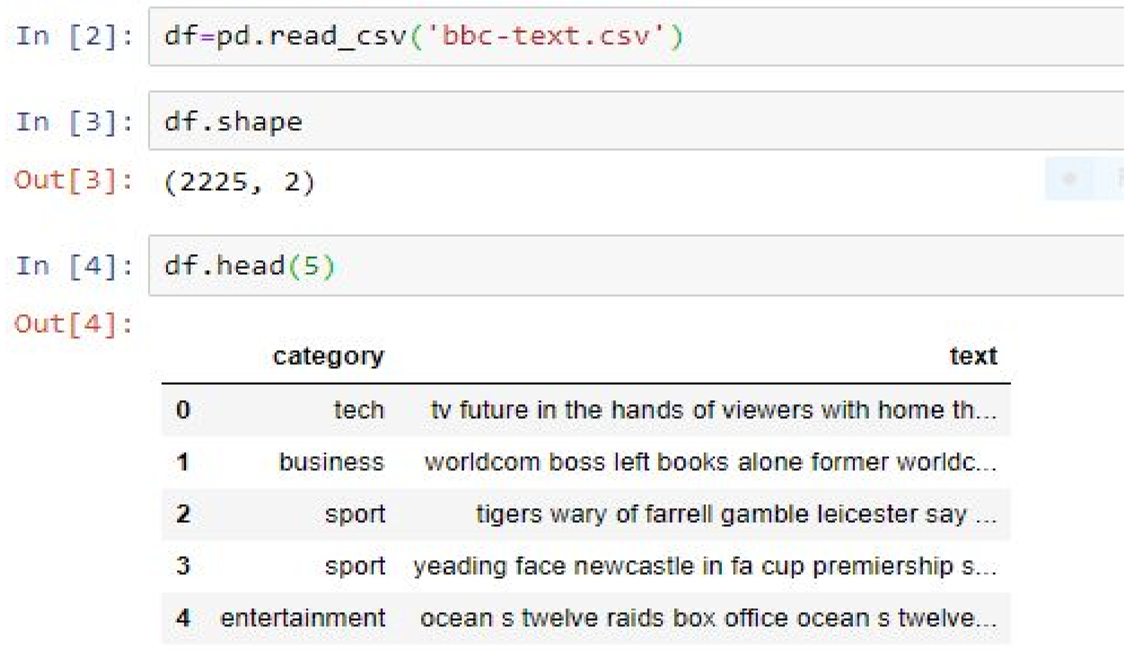
Overview

Text Classification is the task of assigning the right label to a given piece of text. This text can either be a phrase, a sentence or even a paragraph. The aim would be to take in some text as input and attach or assign a label to it. In this project, I would be using a simple deep learning model and TensorFlow as deep learning library.

Approach

STEP 1: Data Loading and Pre-processing

A. Read/Load the dataset (bbc-text.csv) using pandas



```
In [2]: df=pd.read_csv('bbc-text.csv')

In [3]: df.shape
Out[3]: (2225, 2)

In [4]: df.head(5)
Out[4]:
```

	category	text
0	tech	tv future in the hands of viewers with home th...
1	business	worldcom boss left books alone former worldc...
2	sport	tigers wary of farrell gamble leicester say ...
3	sport	yeading face newcastle in fa cup premiership s...
4	entertainment	ocean s twelve raids box office ocean s twelve...

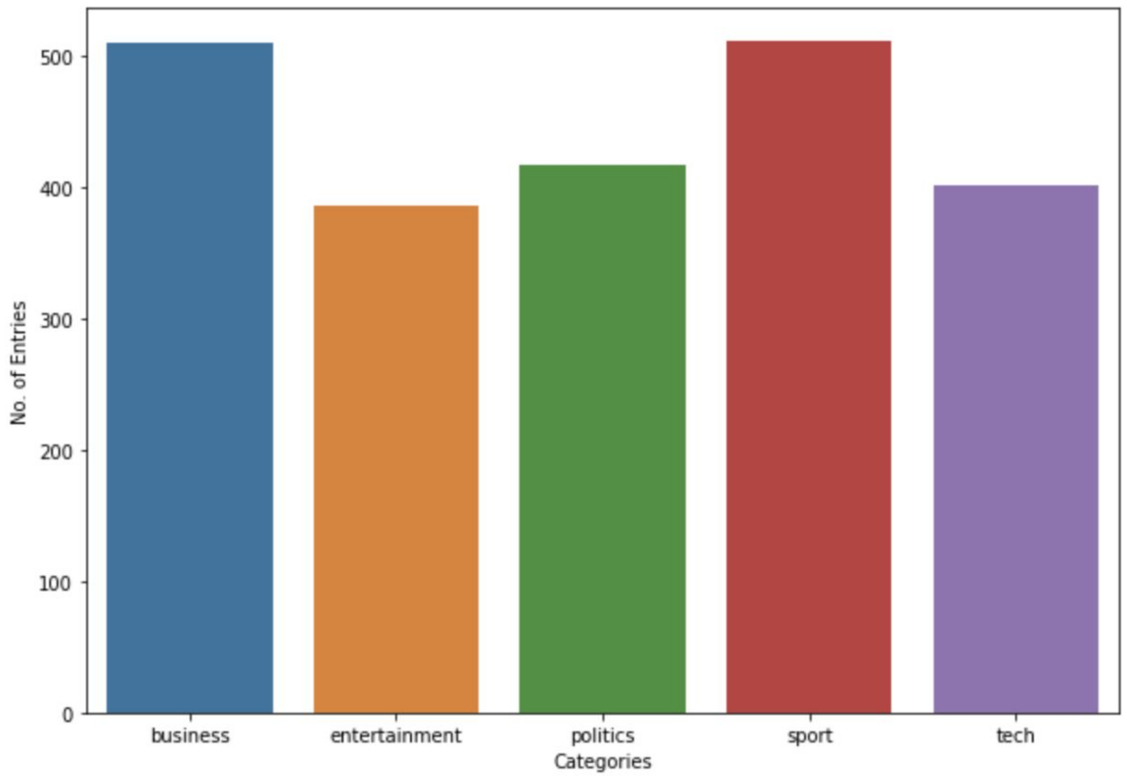
B. Created a table to hold different punctuation marks, and used method "remove_punctuation" to remove punctuations from the sentences

C. Initializing Stemmer

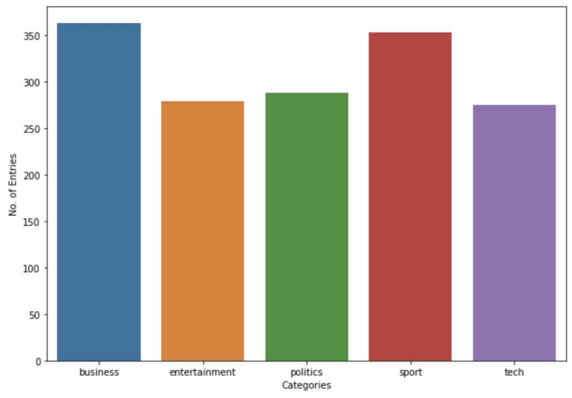
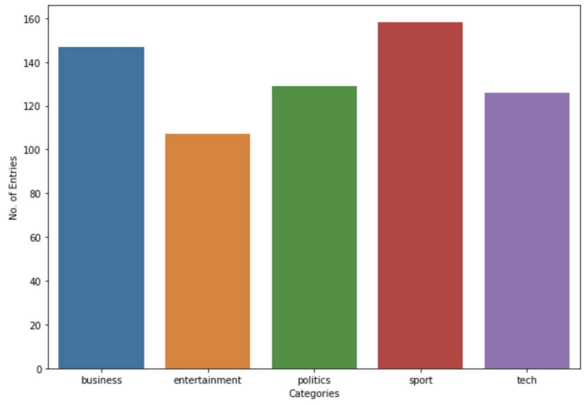
```
stemmer = LancasterStemmer()
```

Stemming – Stemming is a process applied to a single word to derive its root. Many words that are being used in a sentence are often inflected or derived. To standardize our process, we would like to stem such words and end up with only root words. For example, a stemmer will convert the following words “*walking*”, “*walked*”, “*walker*” to its root word “***walk***“.

D. Dataset split (70-30): The category (label) was evenly distributed in dataset, so to split the data 70-30, I sliced the dataframe. Since the categories were evenly distributed I took the first 70% entries as training and rest for testing and hence each label count in training and testing samples were significantly comparable. The dataframe was then converted into a dictionary where the keys represented the “category” and values represented “text” appended as lists.

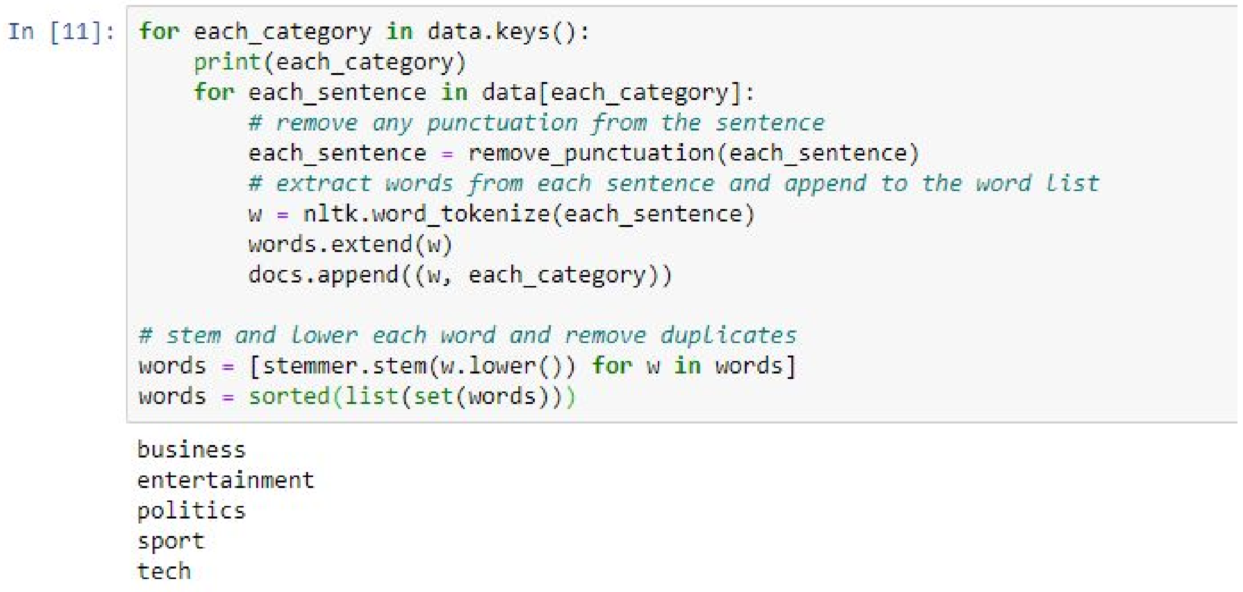


Distribution of categories in dataset

 **Training Data**	 **Testing Data**

E. Each sentence was tokenized and Multiple lists were created, One list “words” to hold all the unique stemmed words in all the sentences provided for training. Another list “categories” to hold all the different categories.

The output of this step was the “docs” list which contains the words from each sentence and which category the sentence belongs.



```
In [11]: for each_category in data.keys():
             print(each_category)
             for each_sentence in data[each_category]:
                 # remove any punctuation from the sentence
                 each_sentence = remove_punctuation(each_sentence)
                 # extract words from each sentence and append to the word list
                 w = nltk.word_tokenize(each_sentence)
                 words.extend(w)
                 docs.append((w, each_category))

         # stem and lower each word and remove duplicates
         words = [stemmer.stem(w.lower()) for w in words]
         words = sorted(list(set(words)))

         business
         entertainment
         politics
         sport
         tech
```

STEP 2: Feature Extraction- Data for Training

The documents in the previous steps are in text format, but Tensorflow being a math library accepts the data in the numeric form. So, before proceeding with text classification, the text form (sentences) must be converted into a numeric representation by using the **bag of words** model. The labels/category are also stored as a numeric values. The array so formed for each category act as a feature and is fed to the deep learning model for training purposes.

NOTE: **Bag of Words –** The Bag of Words model in Text Processing is the process of creating a unique list of words. This model is used as a tool for feature generation.

STEP 3: Training the data model

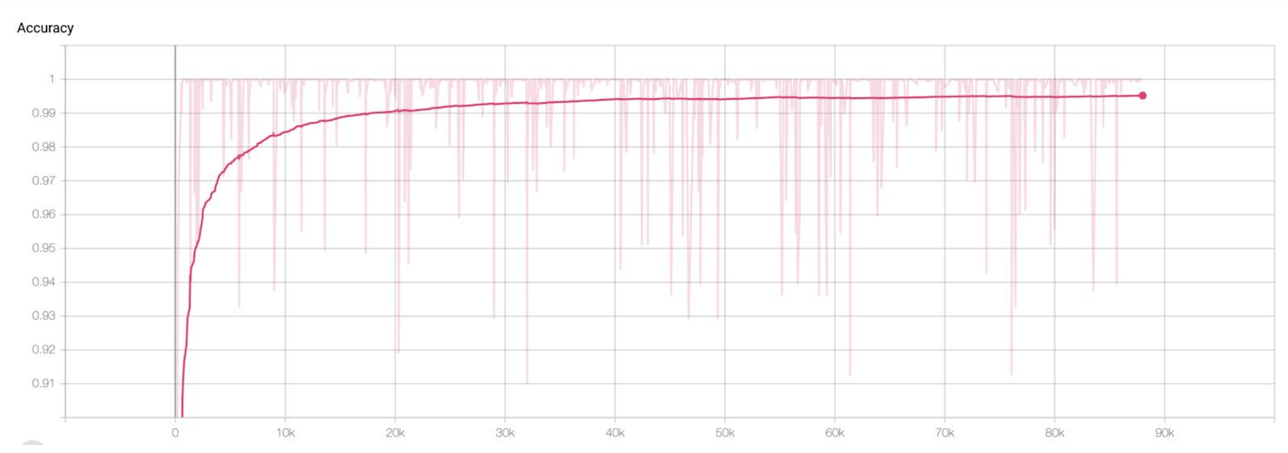
The training dataset was fed to a simple deep neural network having three fully connected layers. Tensorboard was also set up for obtaining accuracy- loss graphs.

Validation Split: 10%
Training samples: 1402
Validation samples: 156

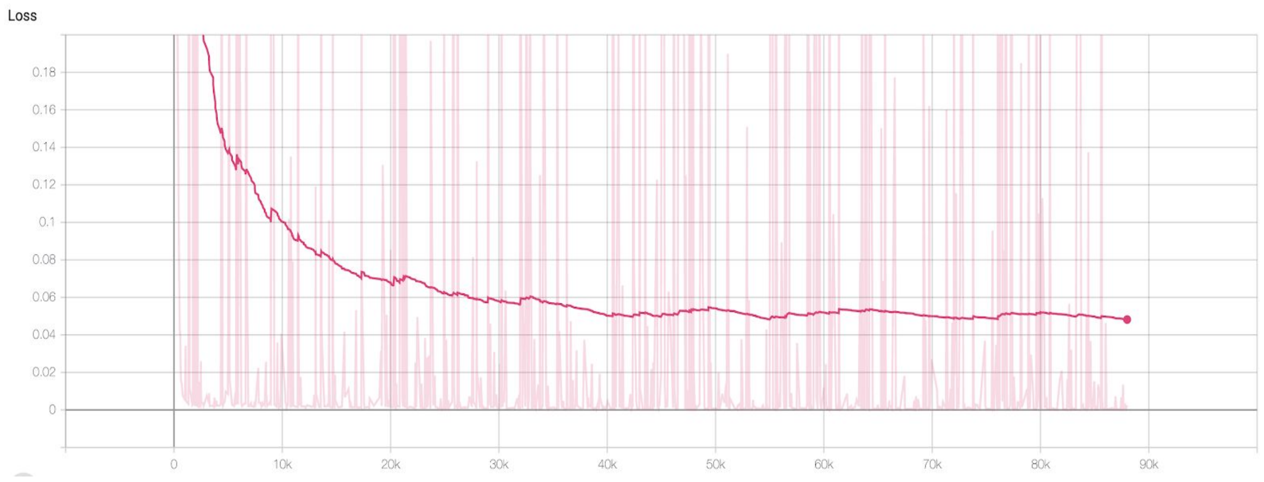
I ran the code for 500 epochs but it was observed that after around 50 epochs the model started to overfit.

Below are the plots when model was trained with 500 epochs

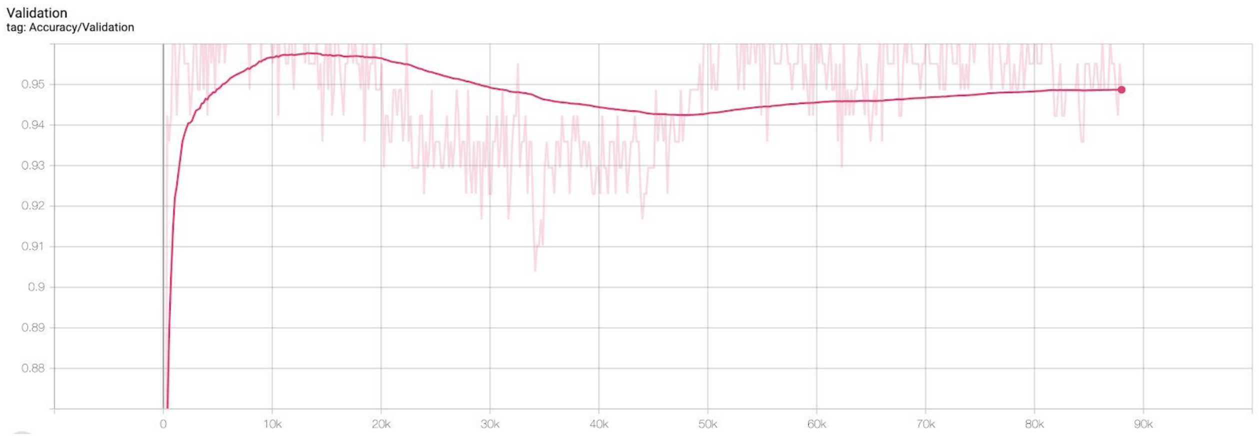
(X-axis shows the no. of steps)



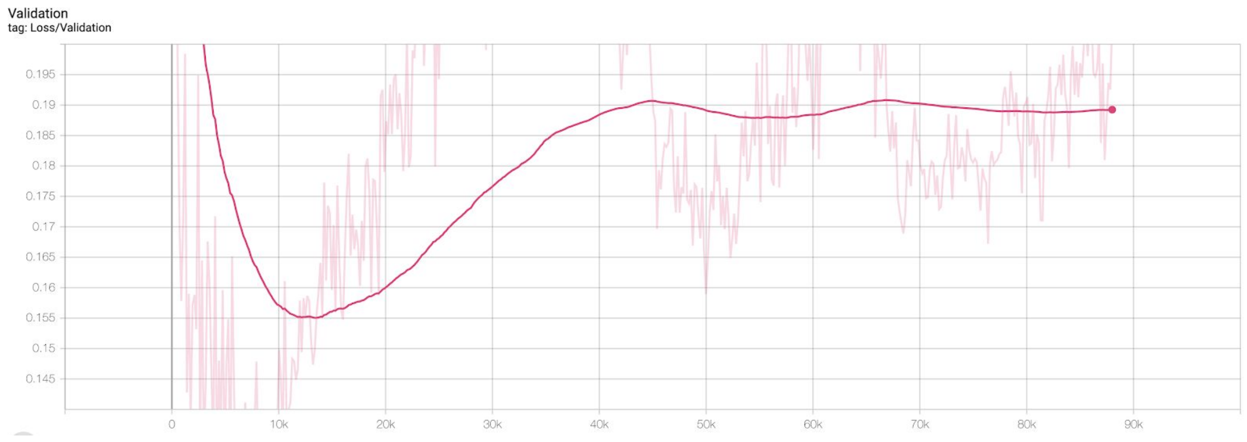
Training Accuracy



Training Loss

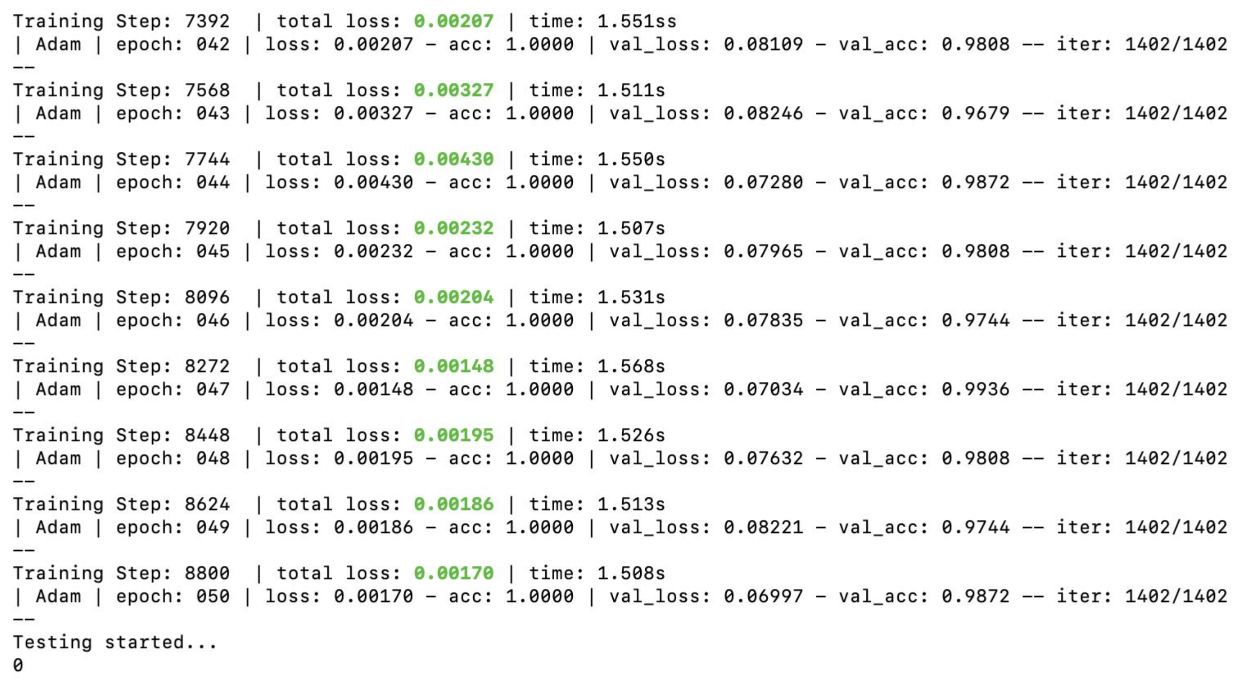


Validation Accuracy



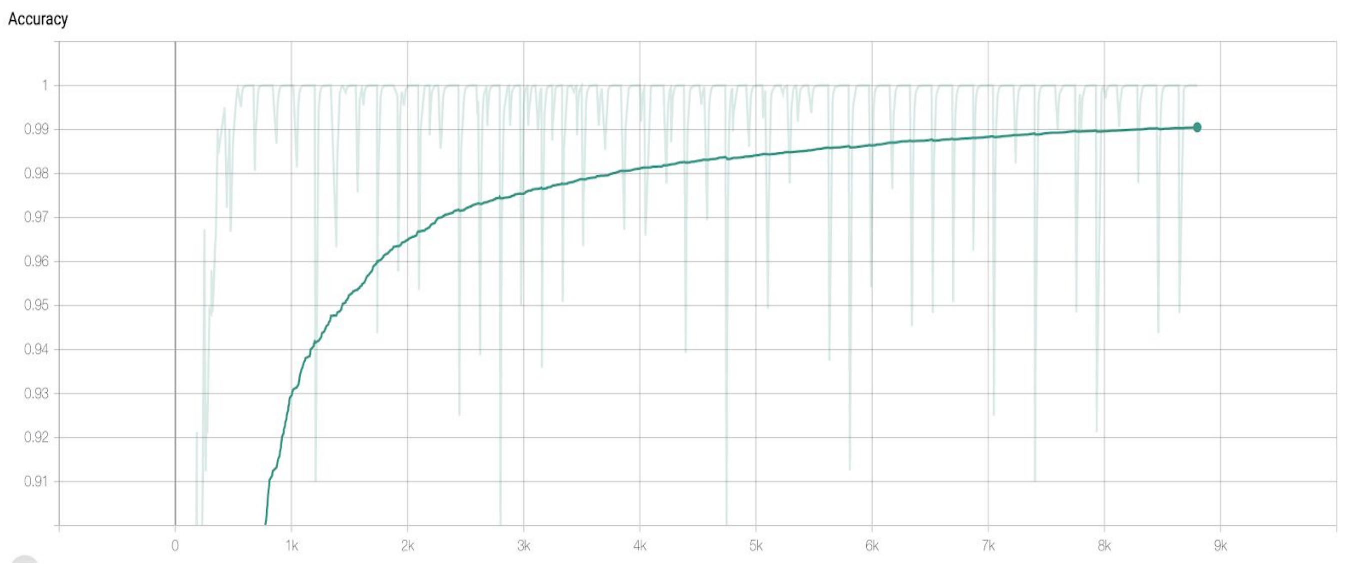
Validation loss

So I reruned the model with 50 epochs to avoid overfitting, the screeenshot of the training is attached below achieving the validation accuracy of 98.72%.



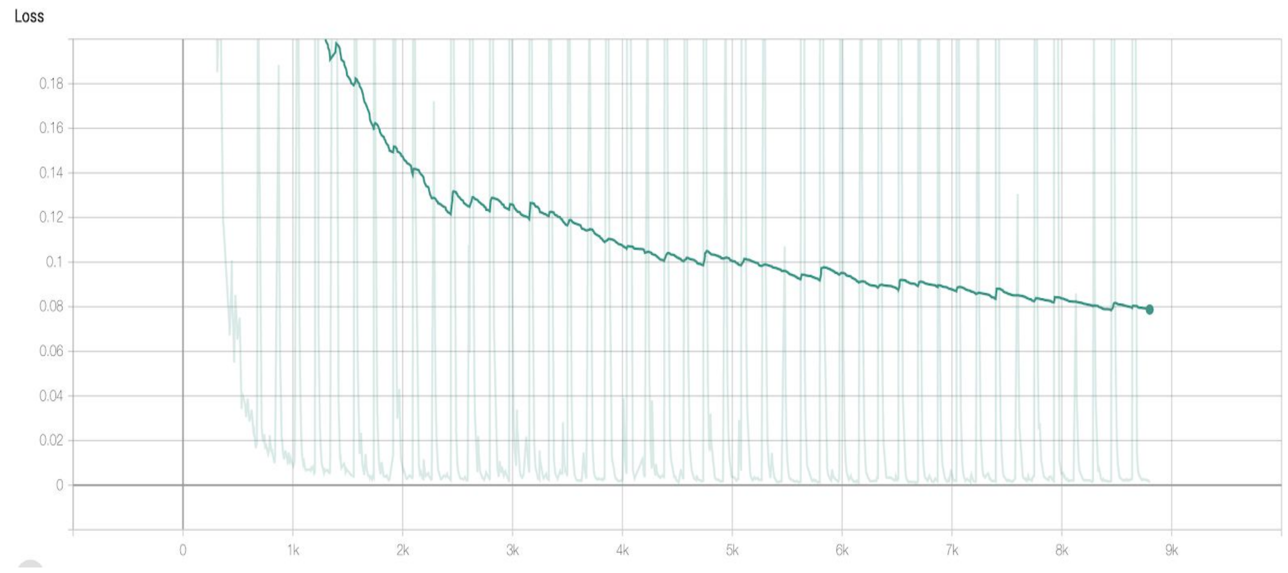
```
Training Step: 7392  | total loss: 0.00207 | time: 1.551ss
| Adam | epoch: 042 | loss: 0.00207 - acc: 1.0000 | val_loss: 0.08109 - val_acc: 0.9808 -- iter: 1402/1402
--
Training Step: 7568  | total loss: 0.00327 | time: 1.511s
| Adam | epoch: 043 | loss: 0.00327 - acc: 1.0000 | val_loss: 0.08246 - val_acc: 0.9679 -- iter: 1402/1402
--
Training Step: 7744  | total loss: 0.00430 | time: 1.550s
| Adam | epoch: 044 | loss: 0.00430 - acc: 1.0000 | val_loss: 0.07280 - val_acc: 0.9872 -- iter: 1402/1402
--
Training Step: 7920  | total loss: 0.00232 | time: 1.507s
| Adam | epoch: 045 | loss: 0.00232 - acc: 1.0000 | val_loss: 0.07965 - val_acc: 0.9808 -- iter: 1402/1402
--
Training Step: 8096  | total loss: 0.00204 | time: 1.531s
| Adam | epoch: 046 | loss: 0.00204 - acc: 1.0000 | val_loss: 0.07835 - val_acc: 0.9744 -- iter: 1402/1402
--
Training Step: 8272  | total loss: 0.00148 | time: 1.568s
| Adam | epoch: 047 | loss: 0.00148 - acc: 1.0000 | val_loss: 0.07034 - val_acc: 0.9936 -- iter: 1402/1402
--
Training Step: 8448  | total loss: 0.00195 | time: 1.526s
| Adam | epoch: 048 | loss: 0.00195 - acc: 1.0000 | val_loss: 0.07632 - val_acc: 0.9808 -- iter: 1402/1402
--
Training Step: 8624  | total loss: 0.00186 | time: 1.513s
| Adam | epoch: 049 | loss: 0.00186 - acc: 1.0000 | val_loss: 0.08221 - val_acc: 0.9744 -- iter: 1402/1402
--
Training Step: 8800  | total loss: 0.00170 | time: 1.508s
| Adam | epoch: 050 | loss: 0.00170 - acc: 1.0000 | val_loss: 0.06997 - val_acc: 0.9872 -- iter: 1402/1402
--
Testing started...
0
```

Accuracy and loss plots after 50 epochs:



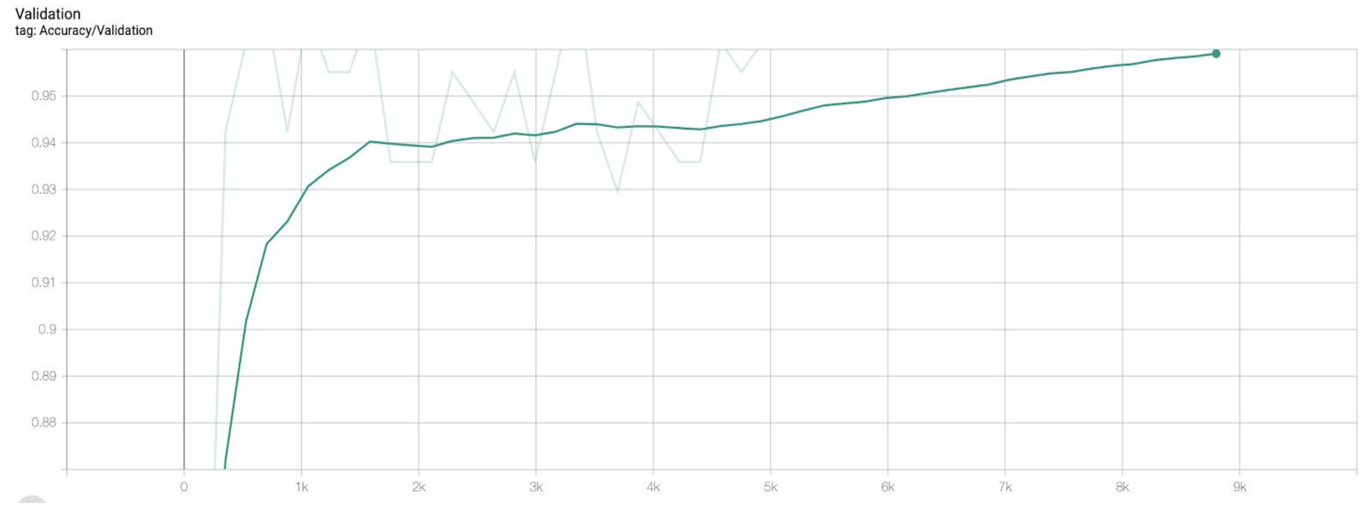
Training Accuracy

X-axis: no. of steps



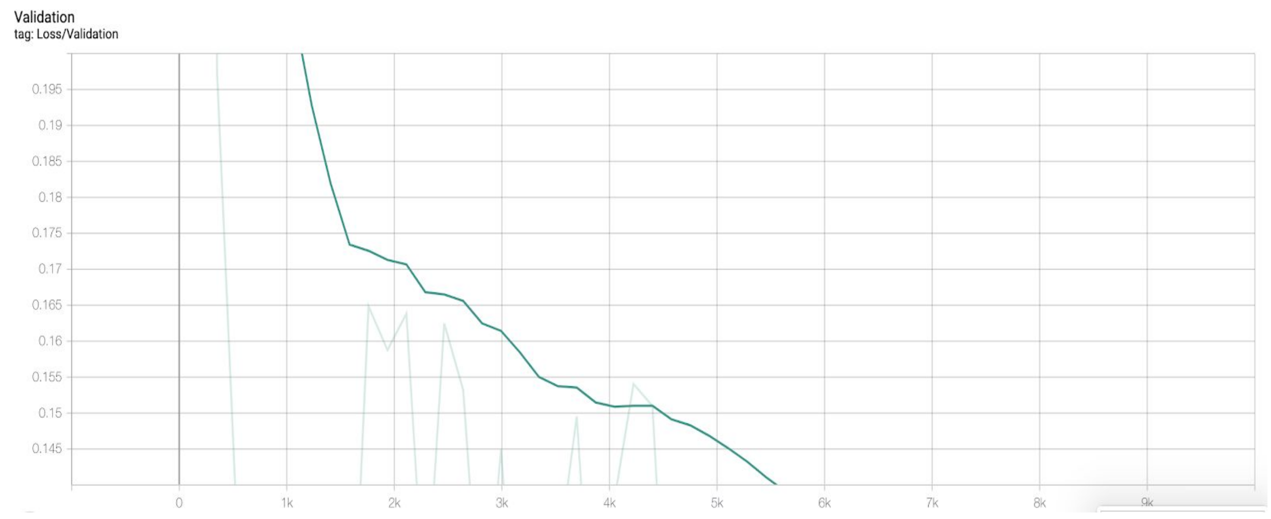
Training Loss

X-axis: no. of steps



Validation Accuracy

X-axis: no. of steps



Validation loss

X-axis: no. of steps

Results after 50 epochs:

Metric	**Value**
Training Accuracy	1.00
Training Loss	0.00170
Validation Accuracy	0.9872
Validation Loss	0.06997

STEP 4: Testing

The model was tested for the testing data we sliced earlier (30% of the entire dataset).

With the provided training, the model was able to correctly classify with an accuracy of 96.2518%.

TESTING ACCURACY= 0.962518