[CMPE 257] Lab2 Report

Jay Bharadva

015958449

Table of Contents

[Task 1] Artificial Neural Network –

Dataset –

Dataset Source –

Click here to visit the site. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Dataset Preparation –

Dataset is divided into 5 different batches. So, I joined them and created one data frame. The initial shape of each image in the dataset was 3072, so I reshaped it to 32 X 32 X 3 dimensions (RGB channel) so that it will be easy to feed it to the Neural Network Model.

Definitions –

Gradient Descent –

It is an optimization algorithm that tries to find the value of coefficients of function such that it minimizes the cost function.

Starts with a small random value -> coefficient = 0.0 -> cost = f(coefficient) -> delta = derivative(cost) ->

coefficient = coefficient – (alpha * delta)

where alpha is a learning rate

Drop out –

This algorithm is used to avoid the co-adoption. It randomly drops the units from the neural network. Their contribution to the activation of downstream neurons is transiently evacuated on the forward pass and updated weights are not applied to the neuron on the backward pass.

Activation Function –

Activation functions decides whether a neuron will be activated or not. In other words, it decides which neurons will useful for the network's input. Examples..., sigmoid, tanh, etc...

Back Propagation –

Back propagation is an algorithm that tries to calculate the derivatives or the Artificial neural networks. ANN uses backpropagation as a learning algorithm to compute a gradient descent with respect to weights It evaluates the expression for the derivatives of the cost function as a product of derivatives between each layer backwards with gradient of the weights between each layer being a simple modification of the partial products.

Epochs –

1 Epoch means training the neural network with all the training data for one cycle. 1 cycle means a forward pass and backward pass in one go. We use all data during one cycle.

Iterations –

It is a number of passes, each pass using number of examples from dataset (batch size). One pass consists of one forward pass and one backward pass.

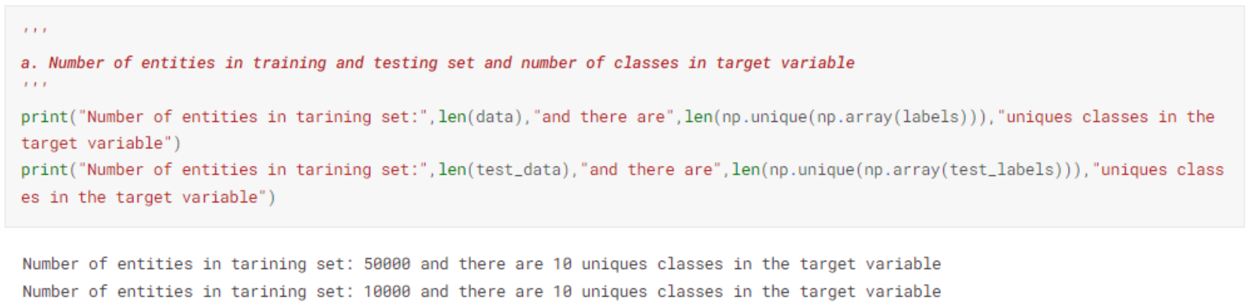
Batch size –

The batch size defines the number of samples that will be propagated through the network. The purpose is not to give much computational load to the model.

Visualize / Summarize data –

Summarize data –

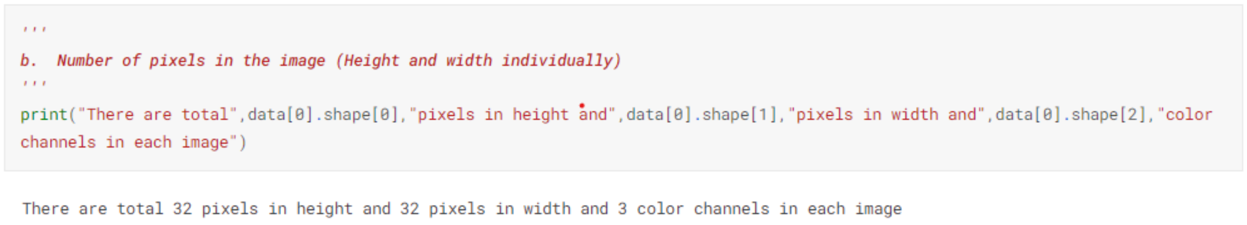
Number of entities in training and testing set and number of classes in target variable



```
'''
a. Number of entities in training and testing set and number of classes in target variable
'''
print("Number of entities in tarining set:",len(data),"and there are",len(np.unique(np.array(labels))),"uniques classes in the target variable")
print("Number of entities in tarining set:",len(test_data),"and there are",len(np.unique(np.array(test_labels))),"uniques classes in the target variable")
```

```
Number of entities in tarining set: 50000 and there are 10 uniques classes in the target variable
Number of entities in tarining set: 10000 and there are 10 uniques classes in the target variable
```

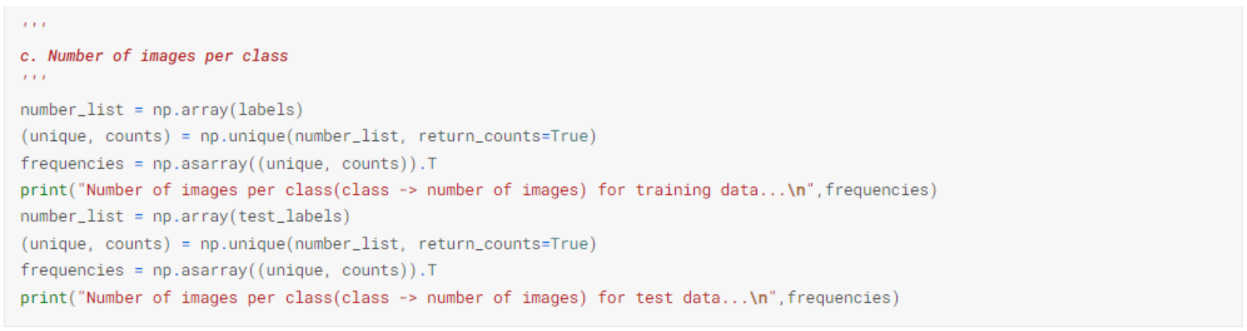
Number of pixels in the image (Height and width individually)



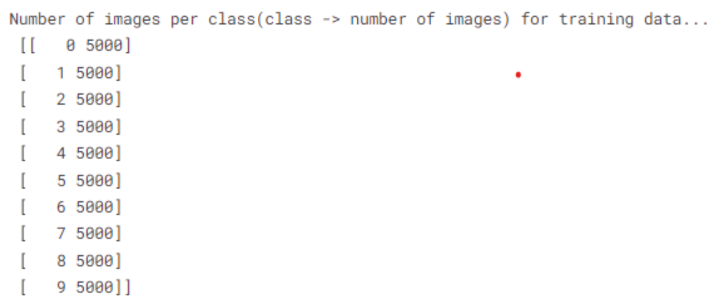
```
'''
b.  Number of pixels in the image (Height and width individually)
'''
print("There are total",data[0].shape[0],"pixels in height and",data[0].shape[1],"pixels in width and",data[0].shape[2],"color channels in each image")
```

```
There are total 32 pixels in height and 32 pixels in width and 3 color channels in each image
```

Number of images per class



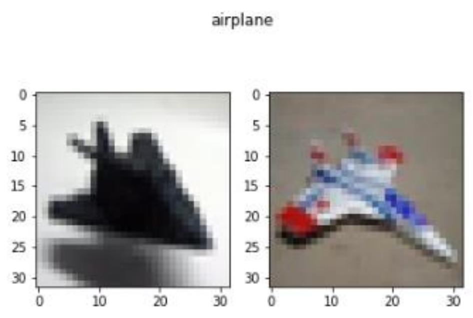
```
'''
c. Number of images per class
'''
number_list = np.array(labels)
(unique, counts) = np.unique(number_list, return_counts=True)
frequencies = np.asarray((unique, counts)).T
print("Number of images per class(class -> number of images) for training data...\n",frequencies)
number_list = np.array(test_labels)
(unique, counts) = np.unique(number_list, return_counts=True)
frequencies = np.asarray((unique, counts)).T
print("Number of images per class(class -> number of images) for test data...\n",frequencies)
```

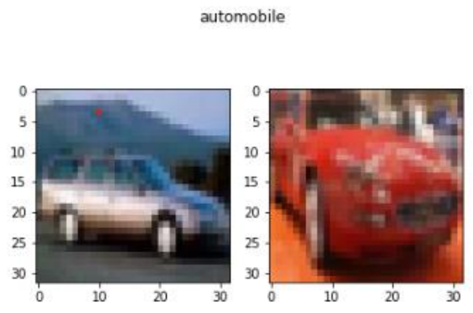


```
Number of images per class(class -> number of images) for training data...
 [[   0 5000]
 [   1 5000]
 [   2 5000]
 [   3 5000]
 [   4 5000]
 [   5 5000]
 [   6 5000]
 [   7 5000]
 [   8 5000]
 [   9 5000]]
```

Visualize data –

Display at least 2 images of each class. (only showing 2 images of 2 class in this doc, full result is in the notebook)





Optimizing the Neural Network –

Approach –

Model1 –

I started with a simple network without conv2D layer. It consist of only Flatten() and dense layers. It was predicting unseen data with 0.5024 accuracy which is very low.

Model2 –

Because of the low accuracy I decided to add convolution layers to the network. So, I added 2 new conv2D layers with 32 filter size and kernel size of 3. It boosted the accuracy and was giving 0.6269 accuracy on validation dataset. But after evaluating the model, it seemed like network was overfitting on the dataset. Used regularization over here to overcome this problem, and with this, I had to makes sure to decrease the trainable parameters of the model.

Model3 –

To introduce regularization, I used dropout method first and recorded the accuracy on it. Model with dropout layer was predicting validation data with 0.6609.

After adding dropout layer in the model, the accuracy boosted by around 6%, I think changing the model, and adding layers in the model will be useful to boost the accuracy more. Options we have as of now: Add more conv2D layers, add regularization for conv2D layers. With this, still trying to reduce the number of trainable parameters of the model.

Model4 –

Added more conv2D layers in the model and tried pooling operation in the neural network. Results are as below…

```
Pooling Operation Methods:
Max Pooling: accuracy (dropout with 0.2 rate) -> 0.7631
Average Pooling: accuracy -> 0.7574
Max Pooling: accuracy (dropout with 0.5 rate) -> 0.7760
```

Model5 –

I tested other 2 regularization method. Results are as follow…

```
Regularization
Activity regularization -> accuracy: 0.7818
Weight constraint -> accuracy: 0.7861
Weight constraint + activity regularization -> accuracy: 0.7793
```

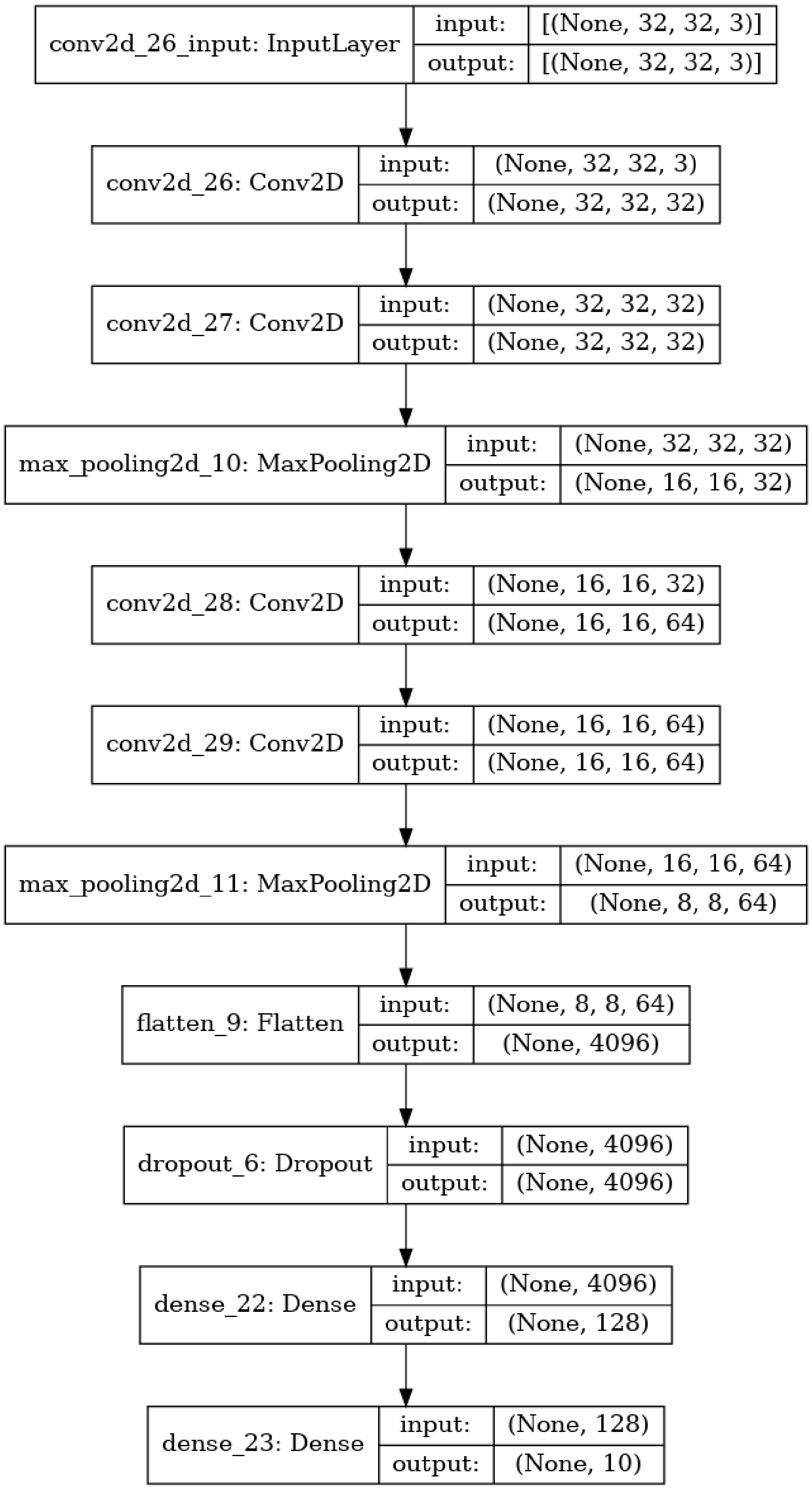
Model6 –

Tested neural network with different optimization methods. Results are as follow…

```
# SGD, RMSprop, Adadelta, Adagrad, Adamax, Nadam
# optimizer -> adam -> 0.7760
# optimizer -> SGD -> 0.7051
# optimizer -> RMSprop -> 0.7124
# optimizer -> Adagrad -> 0.4781
# optimizer -> Adamax -> 0.7700
# optimizer -> Nadam -> 0.7891
```

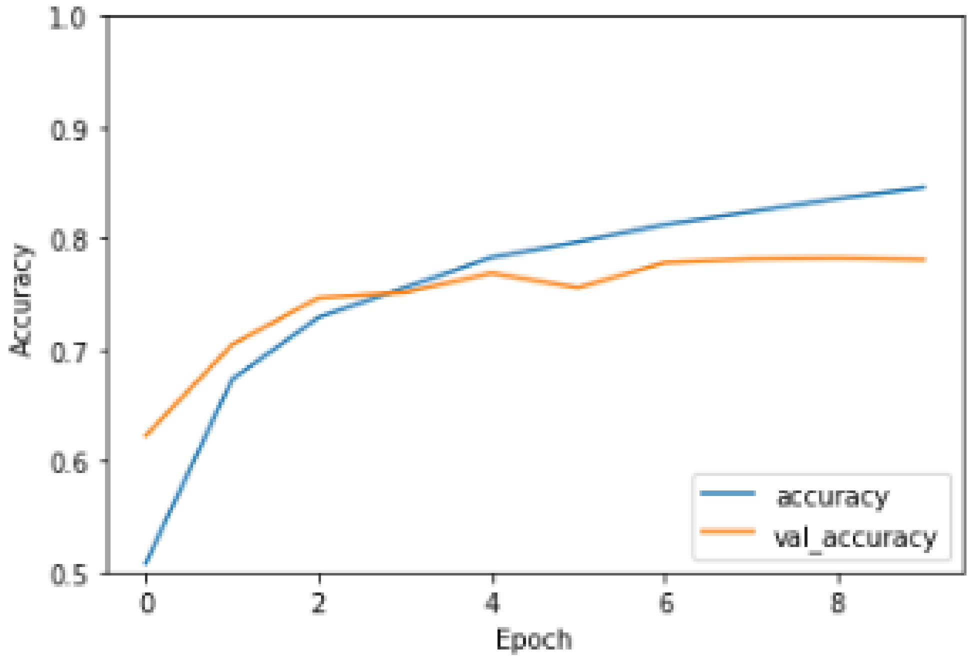
Final Model –

Final Model looks as follows and yields around 0.7808 accuracy on validation dataset. Used Nadam as the optimizer, sparse_categorical_crossentropy as the loss function and accuracy matrix to compile the model and to evaluate it.

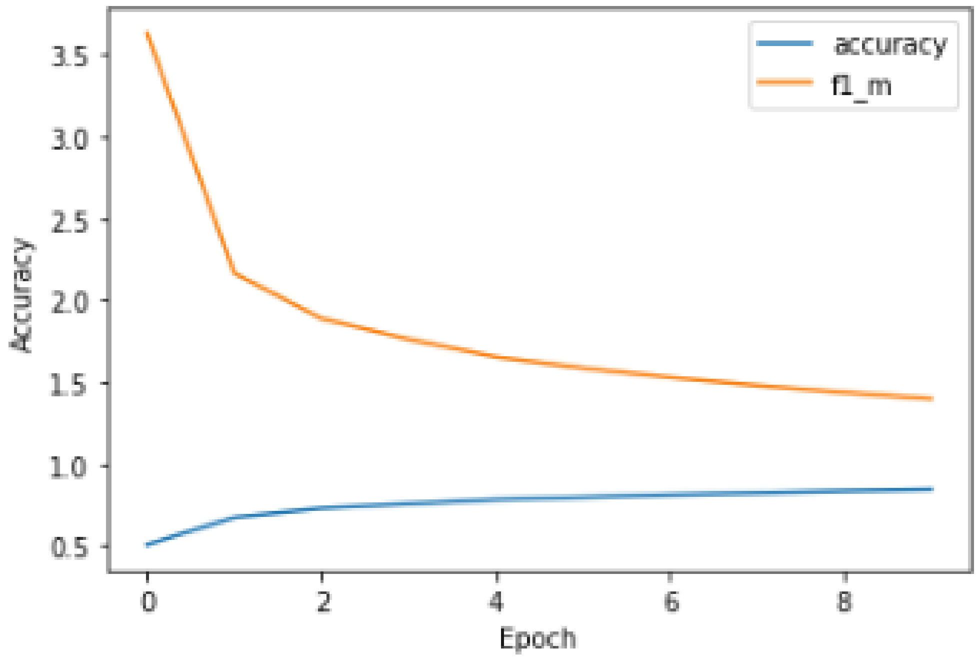

conv2d_26_input: InputLayer
input: [(None, 32, 32, 3)]
output: [(None, 32, 32, 3)]
conv2d_26: Conv2D
input: (None, 32, 32, 3)
output: (None, 32, 32, 32)
conv2d_27: Conv2D
input: (None, 32, 32, 32)
output: (None, 32, 32, 32)
max_pooling2d_10: MaxPooling2D
input: (None, 32, 32, 32)
output: (None, 16, 16, 32)
conv2d_28: Conv2D
input: (None, 16, 16, 32)
output: (None, 16, 16, 64)
conv2d_29: Conv2D
input: (None, 16, 16, 64)
output: (None, 16, 16, 64)
max_pooling2d_11: MaxPooling2D
input: (None, 16, 16, 64)
output: (None, 8, 8, 64)
flatten_9: Flatten
input: (None, 8, 8, 64)
output: (None, 4096)
dropout_6: Dropout
input: (None, 4096)
output: (None, 4096)
dense_22: Dense
input: (None, 4096)
output: (None, 128)
dense_23: Dense
input: (None, 128)
output: (None, 10)

Learning Curves –

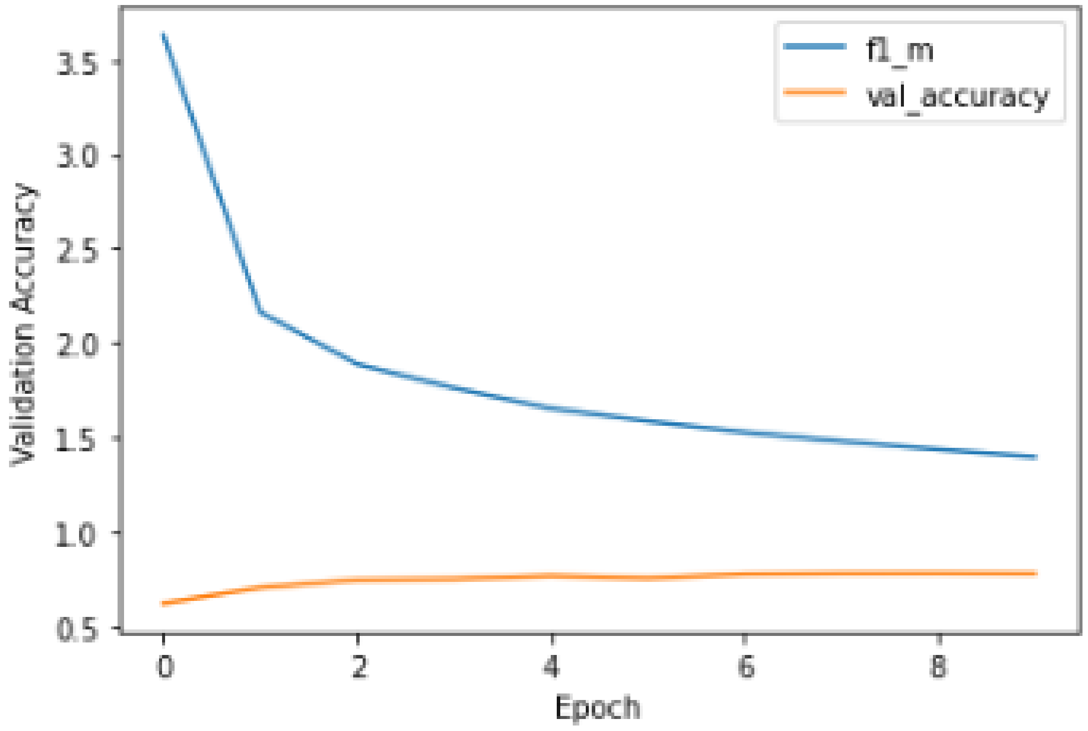
Training accuracy vs validation accuracy



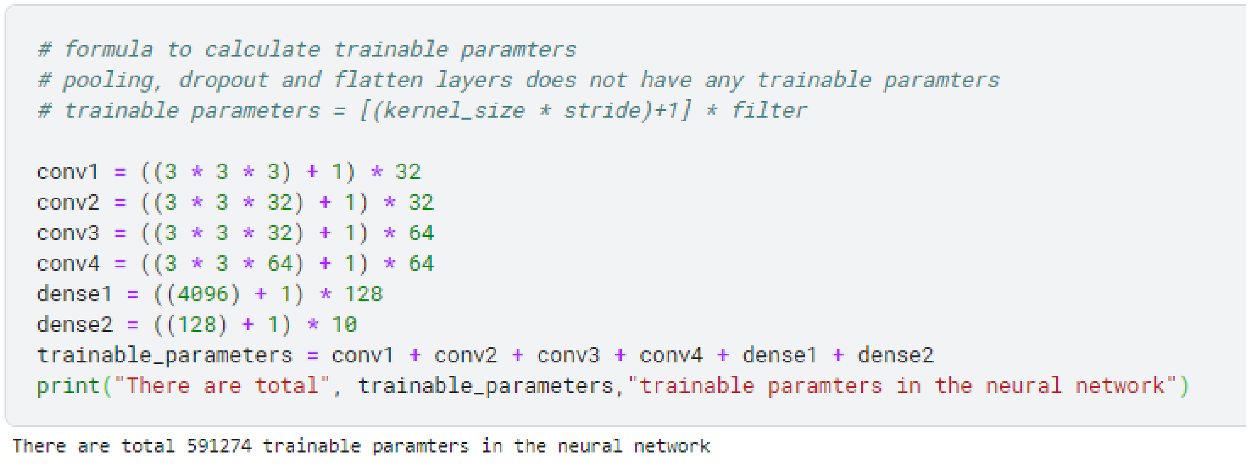
Training accuracy vs f1 score



Validation accuracy vs f1 score



Trainable Parameters of Neural Network –



```
# formula to calculate trainable paramters
# pooling, dropout and flatten layers does not have any trainable paramters
# trainable parameters = [(kernel_size * stride)+1] * filter

conv1 = ((3 * 3 * 3) + 1) * 32
conv2 = ((3 * 3 * 32) + 1) * 32
conv3 = ((3 * 3 * 32) + 1) * 64
conv4 = ((3 * 3 * 64) + 1) * 64
dense1 = ((4096) + 1) * 128
dense2 = ((128) + 1) * 10
trainable_parameters = conv1 + conv2 + conv3 + conv4 + dense1 + dense2
print("There are total", trainable_parameters,"trainable paramters in the neural network")
```

```
There are total 591274 trainable paramters in the neural network
```

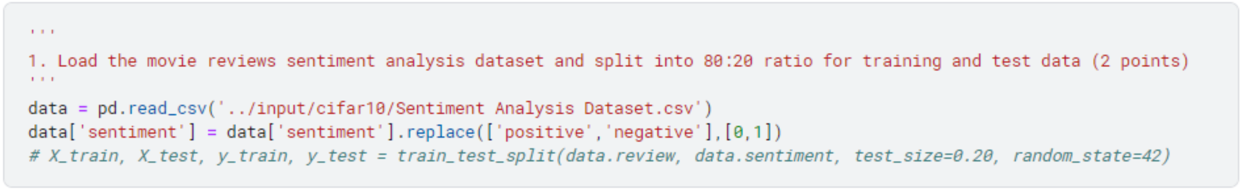
[Task 2] Natural Language Processing –

Dataset –

Dataset Source –

This dataset's original source is IMDB, and it consist of reviews of the real user and its sentiment whether the review was positive or negative. Both classes have the same distribution of the reviews. Each class have 25,000 reviews and the whole dataset consist of 50,000 reviews of real user.

Load Dataset –



```
'''
1. Load the movie reviews sentiment analysis dataset and split into 80:20 ratio for training and test data (2 points)
'''
data = pd.read_csv('../input/cifar10/Sentiment Analysis Dataset.csv')
data['sentiment'] = data['sentiment'].replace(['positive','negative'],[0,1])
# X_train, X_test, y_train, y_test = train_test_split(data.review, data.sentiment, test_size=0.20, random_state=42)
```

Lexical vs Semantic Text Analysis –

Lexical Analysis –

Lexical analysis is referred as the process of extracting an individual words or lexemes from an input stream of symbols and passing corresponding tokens back to the parser. The process starts defining a token in terms of language with a regular expression then it translates to the abstract computational model for recognizing tokens. Which the further translated to an implementation for recognizing the earlier defined tokens to which optimization can be made.

Semantic Analysis –

Semantic analysis, to put it simply, is the process of extracting meaning from text. It enables computers to comprehend and interpret sentences, paragraphs, or entire documents by evaluating their grammatical structure and recognizing relationships between individual words in a given context. Lexical semantics is crucial to semantic analysis because it allows machines to grasp relationships between lexical objects (words, phrasal verbs, and so on).

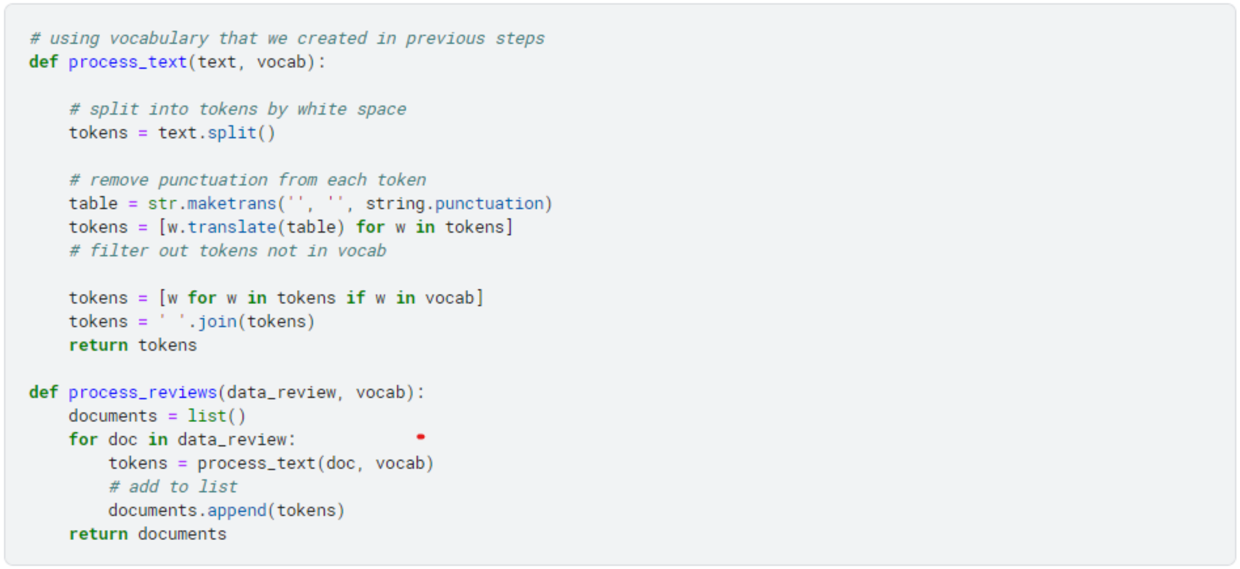
Preprocess the Dataset –

Approach –

- Created a vocabulary file from the all reviews. Now to create this vocabulary file, I parsed each text document to the function which cleans it and converts each word to a token and update the dictionary of vocabulary with each new token list. I used only works with >1 length.
- Saved vocabulary in the text file.
- Using text file created in above steps to clean the reviews. Here, I am first converting all text to a lower case. Then splitting each text into tokens and removing words not present in vocabulary, removing punctuations, non alphabetics and stop words from the tokens and creating new text review by joining all tokens back together.
- To use the keras embedding first we need to convert the cleaned text data to real-valued vector presentation. To do so I have used tokenizer from the keras library and encoded each text data to unique integer values.

- Now, to feed the data to the model we have to make sure that all the documents have same length. And to do so I have used pad_sequence method to pad extra zeros and truncate some wherever required to make sure that all documents have same length.

Code snippet to clean documents –



```
# using vocabulary that we created in previous steps
def process_text(text, vocab):

    # split into tokens by white space
    tokens = text.split()

    # remove punctuation from each token
    table = str.maketrans('', '', string.punctuation)
    tokens = [w.translate(table) for w in tokens]
    # filter out tokens not in vocab

    tokens = [w for w in tokens if w in vocab]
    tokens = ' '.join(tokens)
    return tokens

def process_reviews(data_review, vocab):
    documents = list()
    for doc in data_review:
        tokens = process_text(doc, vocab)
        # add to list
        documents.append(tokens)
    return documents
```

Build a Model –

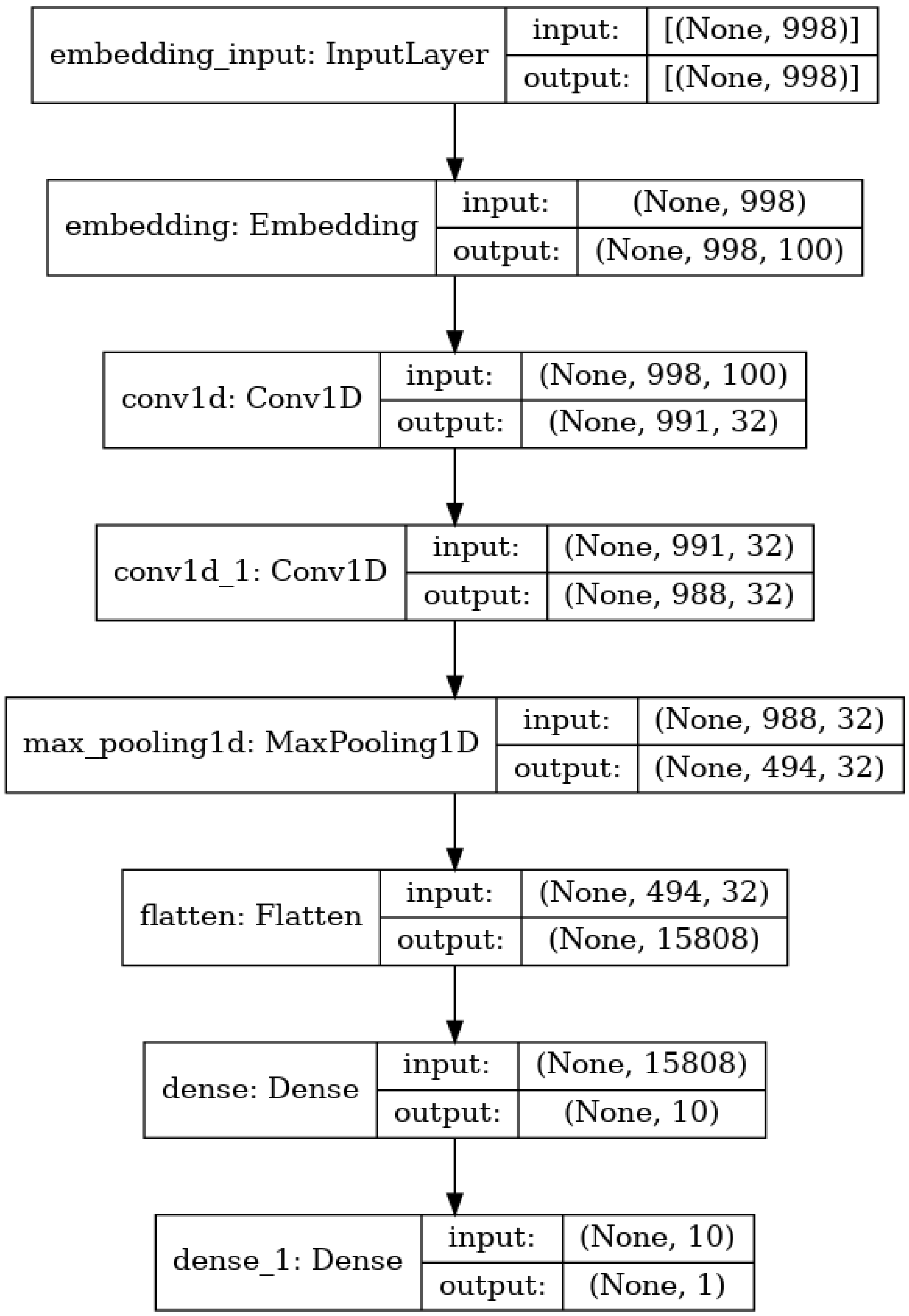
Random Forest –

Build and tested a RandomForestClassifier with a default parameter setting and predicted unseen data with 0.5362 accuracy. Code is in the notebook. Because of the low accuracy I build a Keras model and evaluated it on the processed data

Neural Network –

Build a Neural Network with Embedding layer, conv1D layer, used max pooling layer and dense layer for the output. Neural Network predicted unseen data with 0.8655 accuracy.

Neural Network I build is as followed.


embedding_input: InputLayer
input: [(None, 998)]
output: [(None, 998)]
embedding: Embedding
input: (None, 998)
output: (None, 998, 100)
conv1d: Conv1D
input: (None, 998, 100)
output: (None, 991, 32)
conv1d_1: Conv1D
input: (None, 991, 32)
output: (None, 988, 32)
max_pooling1d: MaxPooling1D
input: (None, 988, 32)
output: (None, 494, 32)
flatten: Flatten
input: (None, 494, 32)
output: (None, 15808)
dense: Dense
input: (None, 15808)
output: (None, 10)
dense_1: Dense
input: (None, 10)
output: (None, 1)

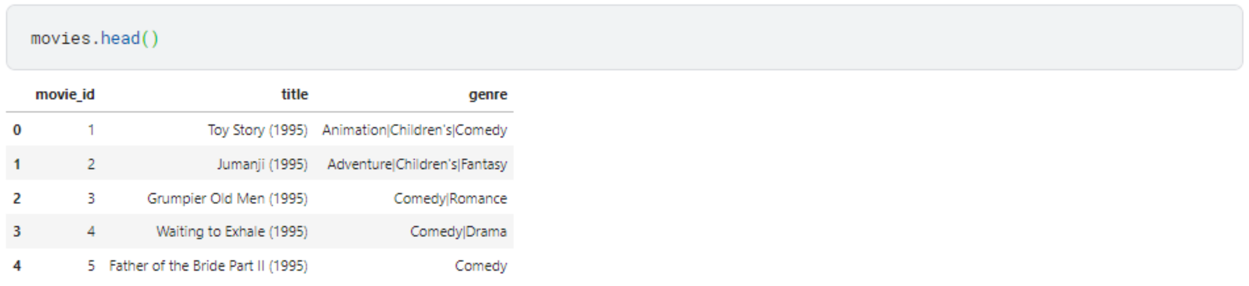
[Task 3] Recommender System –

Dataset –

Dataset Source –

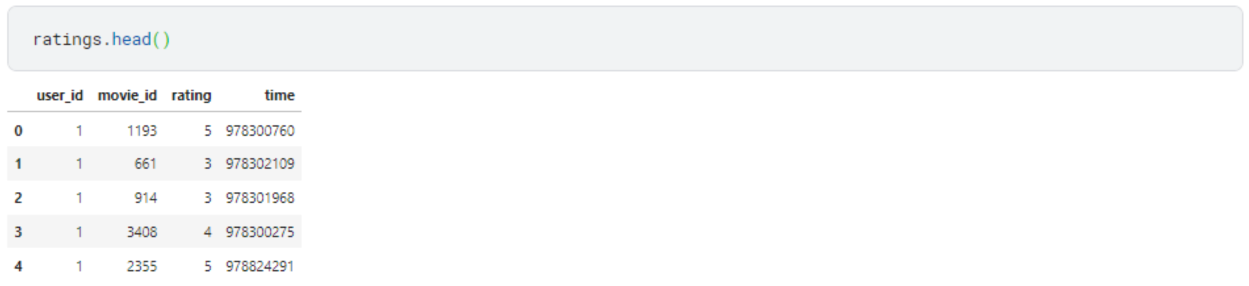
Dataset can be found here. This dataset consists of 1 million ratings provided by 6000 users on 4000 movies. This dataset was published on February 2003.

Movies and Ratings –



```
movies.head()
```

	movie_id	title	genre
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy



```
ratings.head()
```

	user_id	movie_id	rating	time
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291

Definition –

Singular Value Decomposition –

- The Singular Value Decomposition (SVD) of a matrix is a factorization of that matrix into three matrices Given m X n matrix, A, then SVD convert this matrix into 3 matrices as below...

  A = U W V^T

  Here,

  U is a mxn matrix of the orthonormal eigenvectors of A A^T.

  VT is a transpose of a nxn matrix containing the orthonormal eigenvectors of A^T A.

  W is a nxn diagonal matrix of the singular values which are the square roots of the eigenvalues of A^T A.

  A (m X n) = U (m X r) X W (r X r) X V^T (r X n)

- SVD factorizes the matrix into singular vectors and singular values. It provides the information using the eigen decomposition

  To understand the SVD first we need to understand the eigenvalues and eigenvector

  Vector is produced when we multiply Matrix with a Vector. This multiplication is defined as the transformation of the

  vector w.r.t a matrix in the given vector space. However, some vectors for some matrix does not change the direction

  after the transformation is applied to the vector. Such vectors are called eigenvectors.

  The scaled values of the vector after transformation is called eigenvalues.

Principal Component Analysis –

- PCA is a dimensional reduction technique used in the machine learning field to tackle the dataset with many features. When tackling the machine learning task, it is required that we feed only important features to it. Sometimes there are datasets with 100s of feature and in such scenario, it is crucial to get the information about the variables(features) which directly affects the target variable. And some features are not that important and will burden on the model and will make a model calculation much complex. In this case, we require a dimensionality reduction technique to extract important features. In a nutshell, PCA — reduce the number of variables of a data set, while preserving as much information as possible.

- Mathematically, PCA consist of 3 steps.,

  STANDARDIZATION: value - mean / standard deviation

  COMPUTE COVARIANCE MATRIX: To identify the correlation between variables we compute covariance matrix

  COMPUTE EIGENVALUES AND EIGENVECTORS FROM COVARIANCE MATRIX TO FIND PRINCIPAL COMPONENT

- Principal components are new variables that are created by combining or mixing the basic variables in a linear way. The new variables (i.e., main components) are uncorrelated, and the majority of the information contained in the initial variables is squeezed or compressed into the first components.

Content Based Recommendation –

Content based recommendation uses features to recommend other items similar to the features what user likes, based on user's previous actions and/or feedbacks. All the data provided by the uses either by click events or other entities, are used to provide the recommendation to the user. System improves by taking feedback implicitly whether user went with that recommendation or not and improves each time with new data availability.

Collaborative Recommendation –

This recommendation process takes input from similar users and provide the recommendation to the user. The inputs usually are what other users like and purchased, view etc. It works by searching through a huge group of people and finds the smaller set of users with a similar liking, tastes to a particular user. It does so by looking at each item they like and combine them to create a ranked list of suggestions.

Coding Tasks –

Create m x u matrix with movies as row and users as column –

```
import numpy as np
ratings_mat = np.ndarray(
    shape=(np.max(ratings.movie_id.values), np.max(ratings.user_id.values)),
    dtype=np.uint8)
ratings_mat[ratings.movie_id.values-1, ratings.user_id.values-1] = ratings.rating.values
```

Transform the Matrix –

```
normalised_mat = ratings_mat - np.asarray([(np.mean(ratings_mat, 1))]).T
A = normalised_mat.T / np.sqrt(ratings_mat.shape[0] - 1)
```

Perform SVD to get S U and V –

```
U, S, V = np.linalg.svd(A)
```

Top 50 component from V^T –

```
# top 50 components from V.T
k = 50
movie_id = 10 # Grab an id from movies.dat
top_n = 10

sliced_svd = V.T[:, :k] # representative data
```

Calculate Covariance Matrix –

```
normalised_mat = ratings_mat - np.matrix(np.mean(ratings_mat, 1)).T
cov_mat = np.cov(normalised_mat)
```

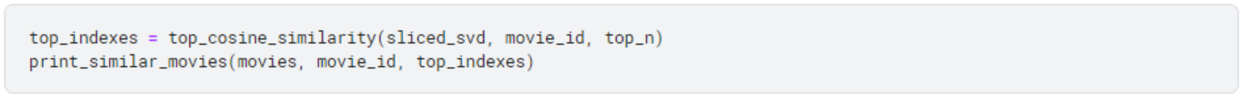
Eigen vector from Covariance Matrix –

```
evals, evecs = np.linalg.eig(cov_mat)
print(evecs)
```

Top 50 eigen vectors using eigen values –

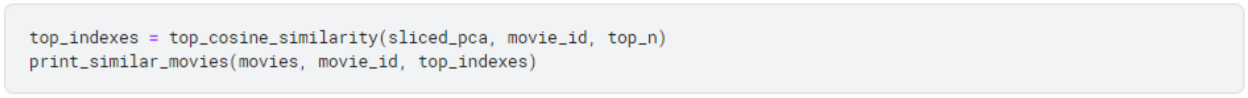
```
k = 50
movie_id = 10 # Grab an id from movies.dat
top_n = 10
sliced_pca = evecs[:, :k] # representative data
print("Top 50 eigen vectors...\n",sliced_pca)
```

Recommend 10 movies using 50 Components from SVD and Cosine Similarity –



```
top_indexes = top_cosine_similarity(sliced_svd, movie_id, top_n)
print_similar_movies(movies, movie_id, top_indexes)
```

Recommend 10 movies using 50 Components from PCA and Cosine Similarity –



```
top_indexes = top_cosine_similarity(sliced_pca, movie_id, top_n)
print_similar_movies(movies, movie_id, top_indexes)
```

Results –

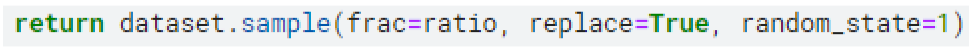
- Recommendations were same for both PCA and SVD techniques.
- Two movies which were used for the prediction has 10 as id in the dataset
- Top 10 recommendations were same.

[Task 4] Random Forest – Self Implementation –

Code Snippets –

Creates subsample of a dataset with replacement –

Using the sample method of pandas library to create subsample of the dataset with replacement



```
return dataset.sample(frac=ratio, replace=True, random_state=1)
```

Create a random selection of n features –

Utilizing a random library of the python and making use of randrange to select random features from the total features. Now appending the feature from index generated from the randrange and appending it to the resulting list.

Implementation of train test split –

Shuffling the dataset instances and selecting len(dataset)-test_size instances from the head of the shuffled dataset for train_data. And for test_data selecting test_size instances from the tail of the dataset.

Implementation of Random Forest Algorithm –

- Using the outputs of Subsample and subsample1 function for this algorithm. Now creating one DecisionTreeClassifier instance.
- subsample gives the output as subsample of a dataset with replacement of instances. And subsample2 gives the subsample of a dataset by selecting random features from dataset obtained from the subsample.
- Output of the above step is fed to the instance of Decision tree created in first step.
- Returning the model and the features (output of subsample1).

Implementation of Prediction function of Random Forest –

- For each tree classifier obtained from the random_forest_predict function, predict features of the output of the same function and store it in y_pred.
- Appending y_pred to the all_preds list.
- Creating a temporary dataframe from the all_preds list.
- Creating a predictions for each column of the dataframe created in above step.
- Returning all_preds and predictions.

Result –

Predicting unseen data with 98% accuracy. Only 2 instances were miss predicted using the implemented RandomForestClassifier.

[Github] Repository Link:

https://github.com/0xLighty/CMPE-257-Labs