Natural Language Understanding

Assignment 1

Submitted by:

Debonil Ghosh (M21AIE225)
Ravi Shankar Kumar (M21AIE247)
Saurav Chowdhury (M21AIE256)

Executive MTech
Artificial Intelligence
Indian Institute of Technology, Jodhpur

Problem Statement:

Dataset:

1. Choose any one text dataset from here.
2. Use accuracy, confusion matrix (class-wise) as a metric for multi-class classification.
3. Use accuracy, Precision, Recall, F1 score and confusion matrix as a metric for binary classification.
4. Report hyperparameters for all deep models, like learning rate, optimiser, number of epochs, and scheduler.
5. Show train/val loss and accuracy plots for deep neural networks.
6. Show some error examples where a wrong class is predicted by the best model in 4.

Tasks:

1. Define your own train-val-test split. [Report the split chosen.]
2. Define a text preprocessing pipeline, i.e., stopword removal, lower casing, punctuation removal etc. [Report your text preprocessing pipeline in the report.]
3. Developing ML methods:
   a. Model a Naive Bayes classifier.
      i. Count vectorizer features.
      ii. TF-IDF features.
   b. Model a decision tree with TF-IDF features. [Compare with 3.a.ii]
4. Developing Deep neural networks:
   a. RNN model.
      i. 64 hidden-vector dimension.
      ii. 256 hidden-vector dimension.

   b. 1-layer LSTM model. [choose 64 or 256 as hidden-vector representation based on the results from 4.a. Report the choice and its justification.]
   c. 2-layer LSTM model. [use the same hidden-vector representation as 4.b.]
   d. 1-layer Bi-LSTM model. [use same hidden-vector representation as 4.b. Report 4.b vs 4.d model performance.]
   e. Use Google word2vec embeddings as input embedding to model in 4.d. [Compare the performance 4.e vs 4.d]
   f. Use Glove embeddings as input embedding to model in 4.d. [Compare the performance 4.f vs 4.d]
   g. Compare 4.e vs 4.f

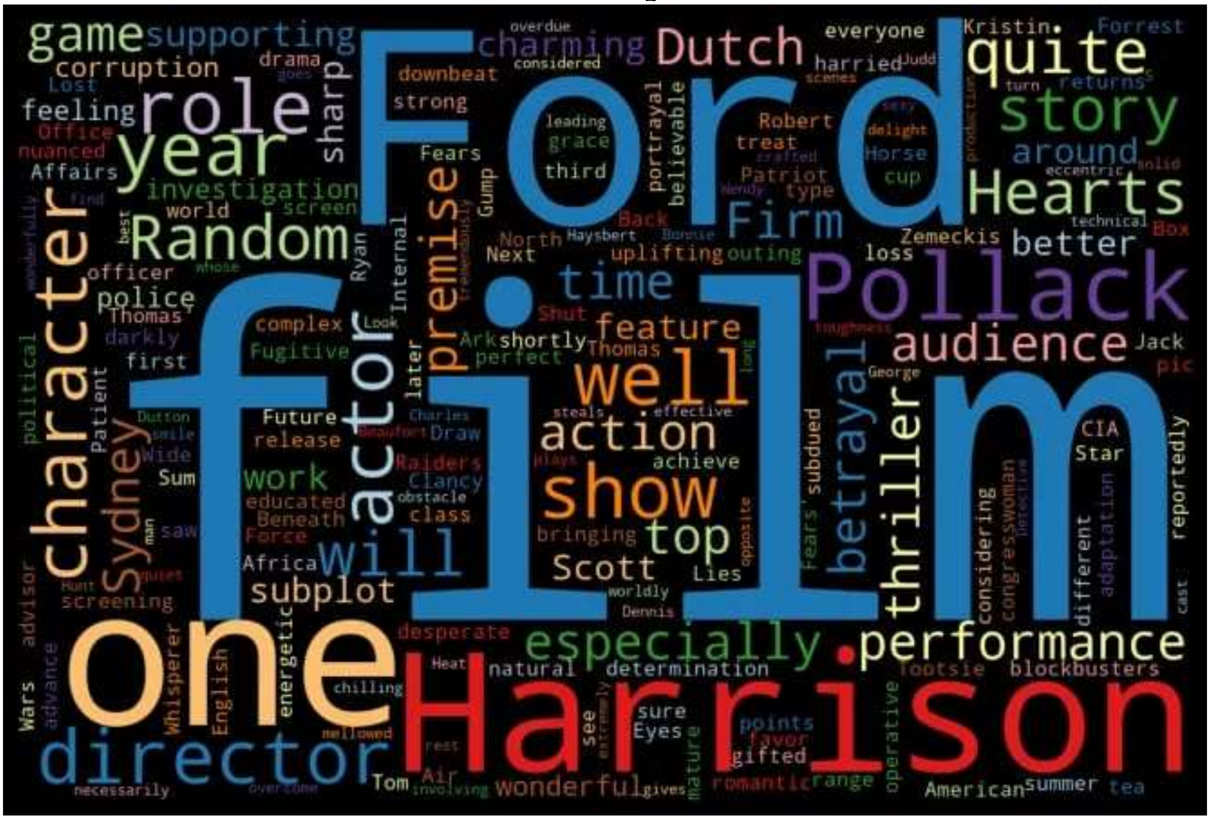
Chosen Dataset:

Dataset Name: IMDB Dataset of 50K Movie Reviews

Data size: 50000

Classes: 2 classes: positive, negative

Data Samples:



Q. 1. Define your own train-val-test split. [Report the split chosen.]

Train, Test, Validation set ratio:	**18:5:2**
Train set size:	**36000**
Test set size:	**10000**
Val set size:	**4000**

Q. 2. Define a text pre-processing pipeline, i.e., stopword removal, lower casing, punctuation removal etc.

<table>
<tr><td>Raw Text at the start</td></tr>
<tr><td>The movie is good , I loved the movie. Will audience like it ?!!</td></tr>
<tr><td>Step 1: Convert to lower case</td></tr>
<tr><td>the movie is good , i loved the movie. will audience like it ?!!</td></tr>
<tr><td>Step 2: Remove ('[/(){}\[\]\|@,;]')</td></tr>
<tr><td>the movie is good i loved the movie will audience like it</td></tr>
<tr><td>Step 3: Remove everything except 0-9a-z #+_</td></tr>
<tr><td>the movie is good i loved the movie will audience like it</td></tr>
<tr><td>Step 4: Remove numeric characters</td></tr>
<tr><td>the movie is good i loved the movie will audience like it</td></tr>
<tr><td>Step 4: Remove stop words</td></tr>
<tr><td>movie good loved movie audience like</td></tr>
</table>

Q.3. Developing ML methods:

a. Model a Naive Bayes classifier.

i. Count vectorizer features.

ii. TF-IDF features.

Multinomial Naïve Bayes Classifier used.

Count Vectorizer:

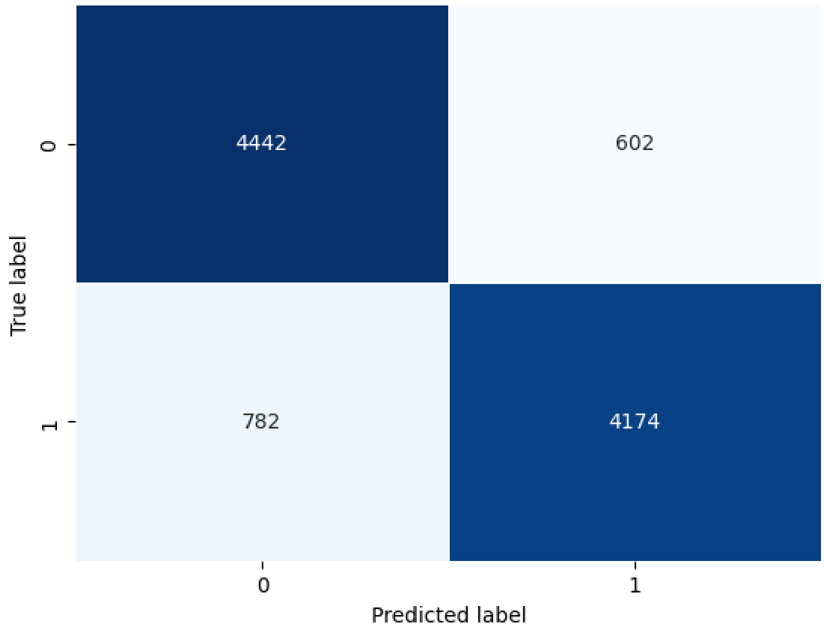
IT will create a matrix of count of each word in each of the reviews data

Results

Classification Report

	precision	recall	f1-score	support
0	0.85	0.88	0.87	5044
1	0.87	0.84	0.86	4956
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

Accuracy: 0.8616



TF-IDF:

TF= Number of time word repeated in a sentence/ Number of words in sentence

IDF = Log(Number of sentences/Number of sentences containing the word)

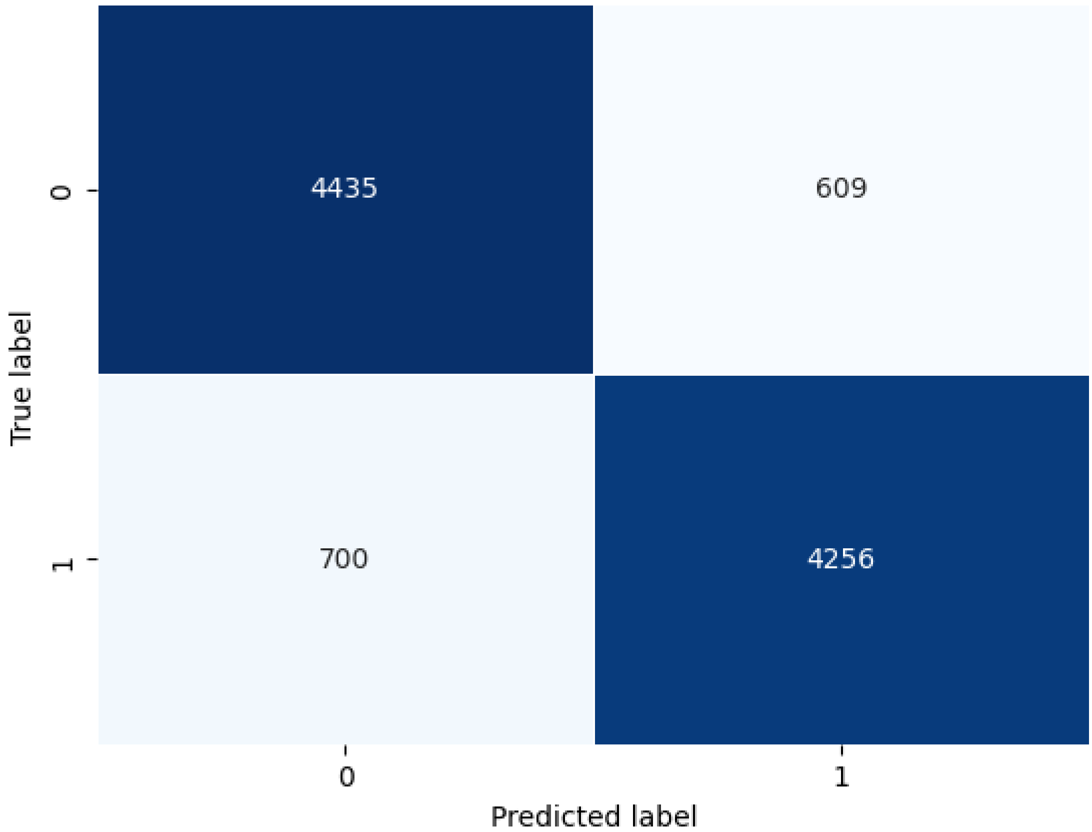
TF*IDF = Dependent Feature

Results

Classification Report

	precision	recall	f1-score	support
0	0.86	0.88	0.87	5044
1	0.87	0.86	0.87	4956
accuracy			0.87	10000
macro avg	0.87	0.87	0.87	10000
weighted avg	0.87	0.87	0.87	10000

Accuracy: **0.8691**



II. Model a ***decision tree*** *with TF-IDF features:*

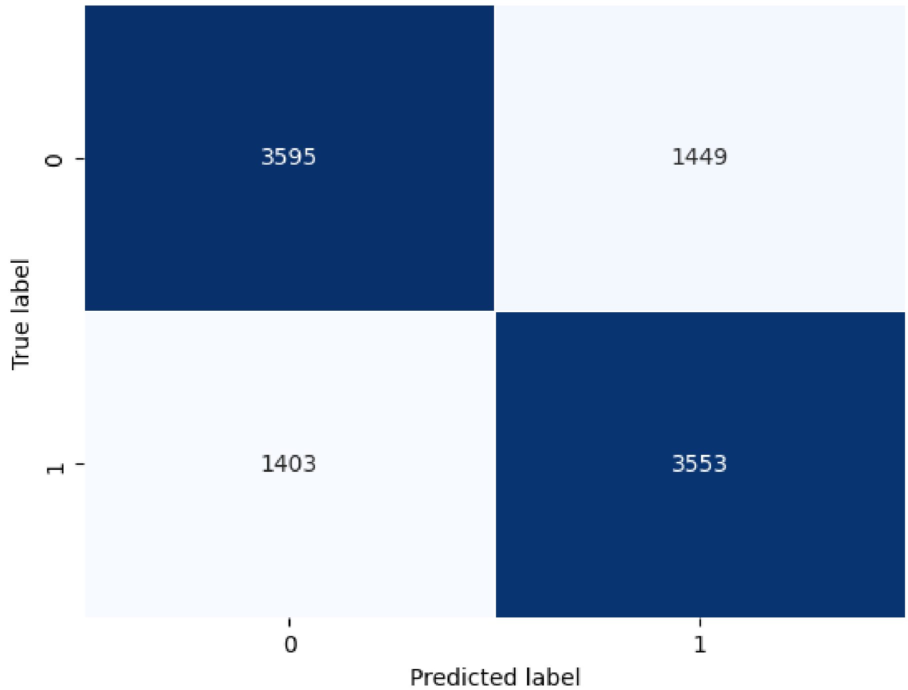
To minimise the dimensionality of the feature space, we can preprocess the text data by tokenizing it into words, deleting stop words and punctuation, and using stemming or lemmatization. The TF-IDF transformation can then be used to generate a matrix of TF-IDF scores for each word in each review. To classify incoming reviews as positive or negative based on the TF-IDF features that are most discriminative for each class, we can lastly train a decision tree on the TF-IDF matrix. The decision tree that is produced can be used to examine the most significant words and phrases that affect the reviews' sentiment.

Results:

Classification Report

	precision	recall	f1-score	support
0	0.72	0.71	0.72	5044
1	0.71	0.72	0.71	4956
accuracy			0.71	10000
macro avg	0.71	0.71	0.71	10000
weighted avg	0.71	0.71	0.71	10000

Accuracy : 0.7148



Comparing TF*IDF and Decision tree with TF_IDf : We see that the Accuracy of TF*IDF is more.

Q.4 Developing Deep neural networks:

a. RNN model.

i. 64 hidden-vector dimension.

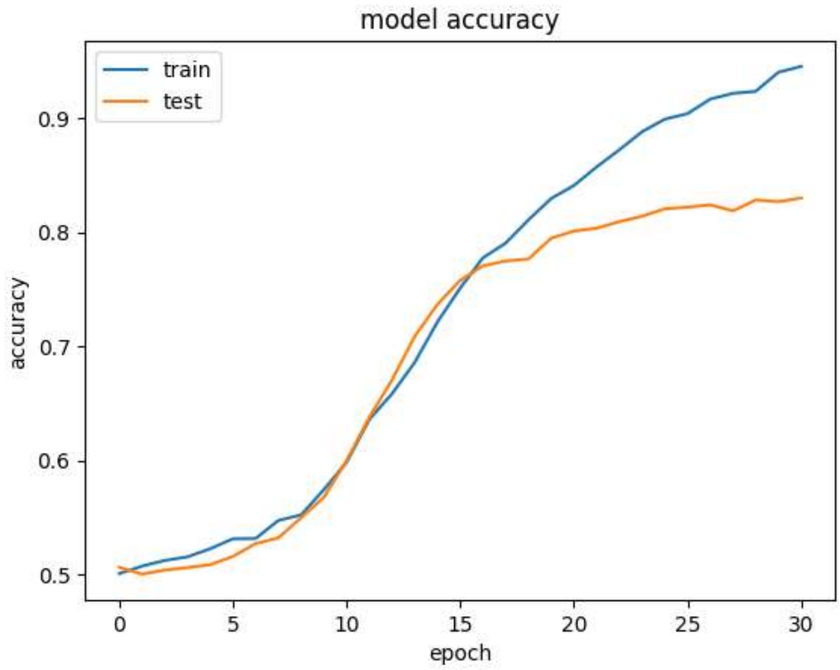
Simple RNN model with 64 hidden layers.

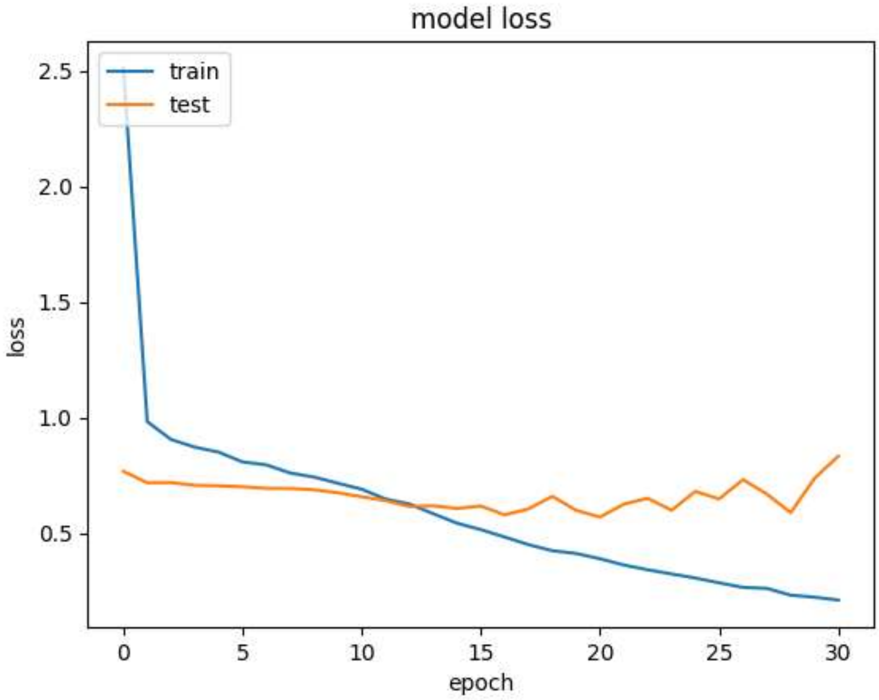
Model: "SimpleRNNModel64"

Layer (type)	Output Shape	Param #
embedding_6 (Embedding)	(None, 64, 64)	8900032
simple_rnn_6 (SimpleRNN)	(None, 64)	8256
dense_6 (Dense)	(None, 64)	4160
dropout (Dropout)	(None, 64)	0
dense_7 (Dense)	(None, 1)	65

Total params: 8,912,513
Trainable params: 8,912,513
Non-trainable params: 0

The layers of the RNN are shown above



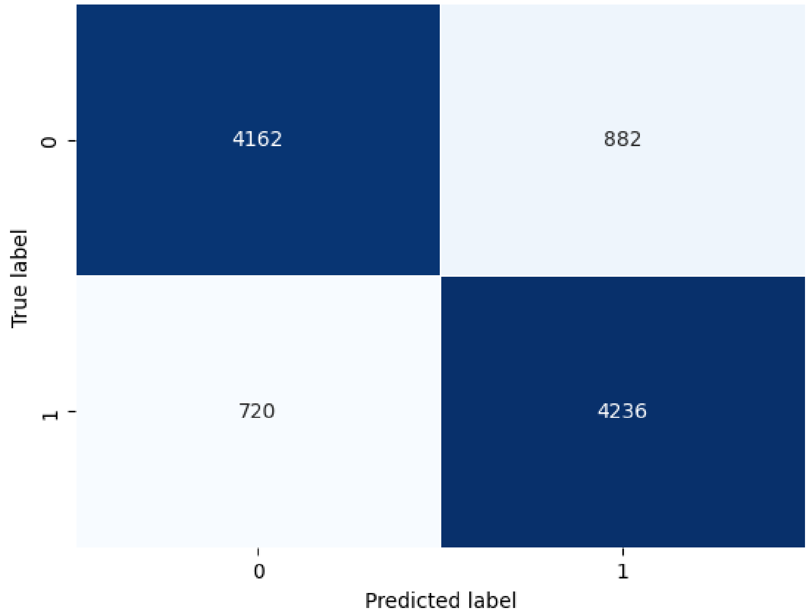


Classification Report

	precision	recall	f1-score	support
0	0.85	0.83	0.84	5044
1	0.83	0.85	0.84	4956
accuracy			0.84	10000
macro avg	0.84	0.84	0.84	10000

weighted avg	0.84	0.84	0.84	10000

Accuracy : 0.8398
Accuracy :83.98 %



Confusion Matrix

iii. 256 hidden-vector dimension.

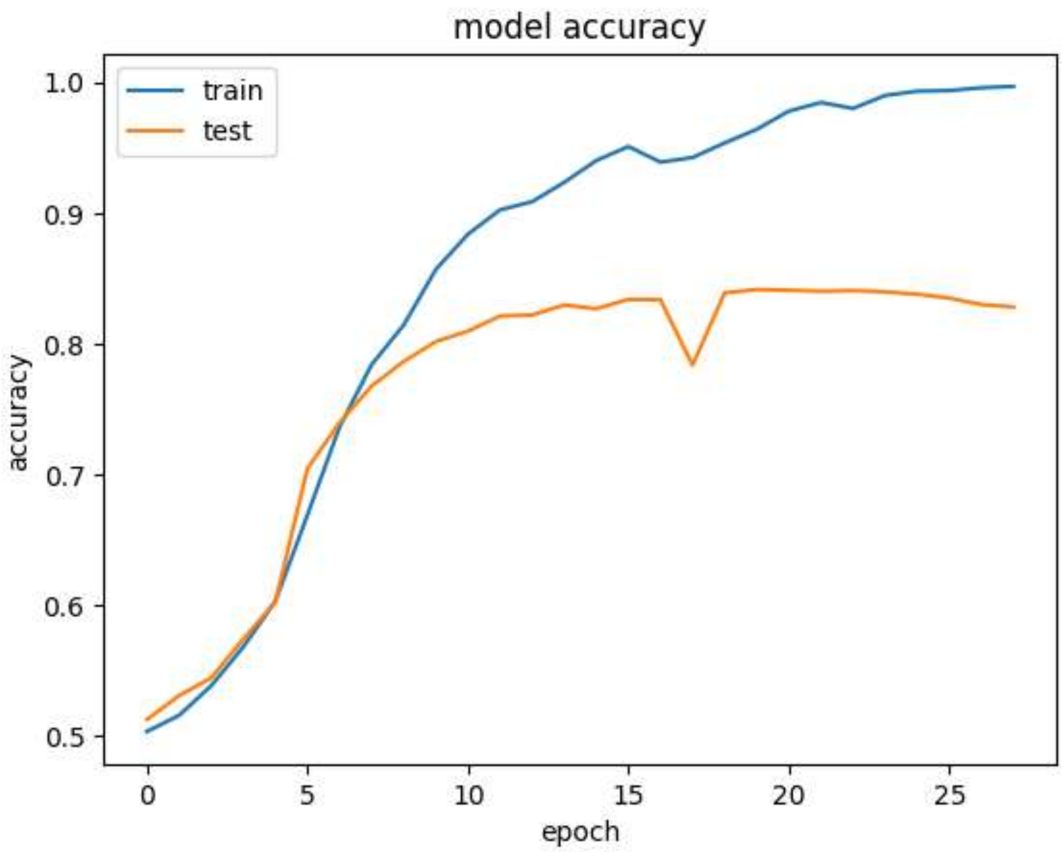
Simple RNN model with 256 hidden layers.

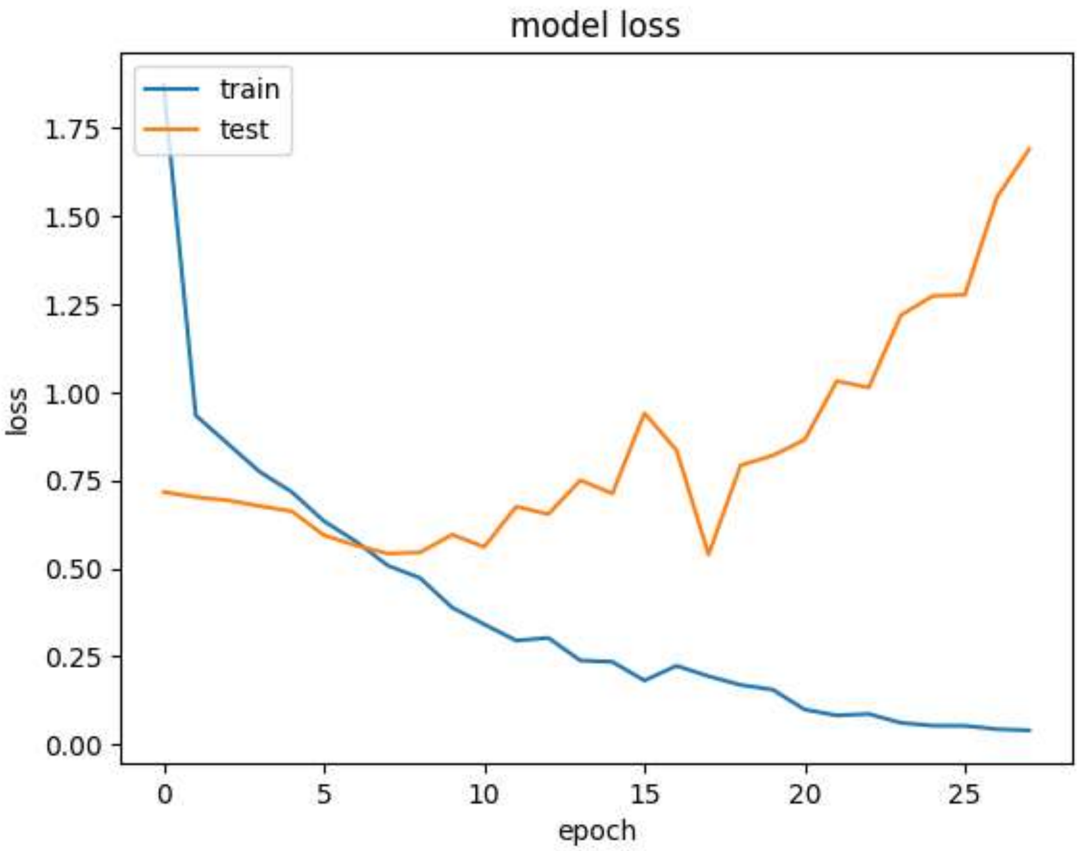
Model: "SimpleRNNModel256"

Layer (type)	Output Shape	Param #
embedding_10 (Embedding)	(None, 64, 256)	35600128
simple_rnn_10 (SimpleRNN)	(None, 256)	131328
dense_14 (Dense)	(None, 256)	65792
dropout_4 (Dropout)	(None, 256)	0
dense_15 (Dense)	(None, 1)	257

Total params: 35,797,505
Trainable params: 35,797,505
Non-trainable params: 0

Layers of the RNN are shown above

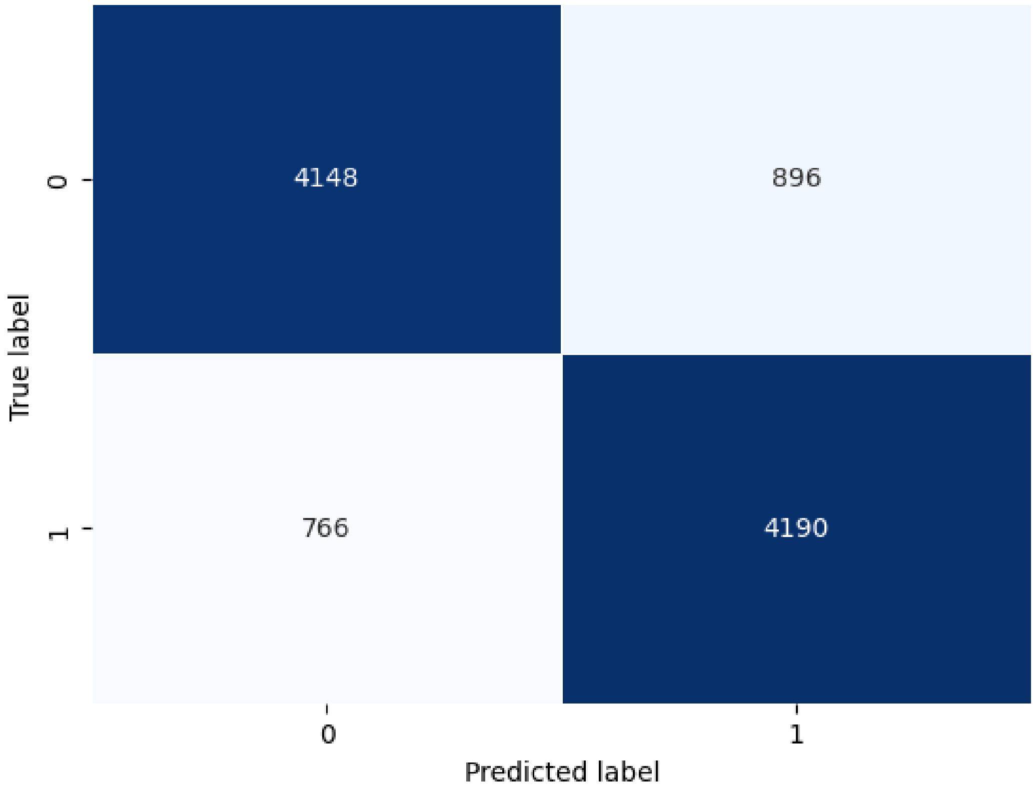

model accuracy
train
test
accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
epoch


model loss
train
test
loss
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
25
epoch

Classification Report

	precision	recall	f1-score	support
0	0.84	0.82	0.83	5044
1	0.82	0.85	0.83	4956
accuracy			0.83	10000
macro avg	0.83	0.83	0.83	10000
weighted avg	0.83	0.83	0.83	10000

Accuracy : 83.38 %

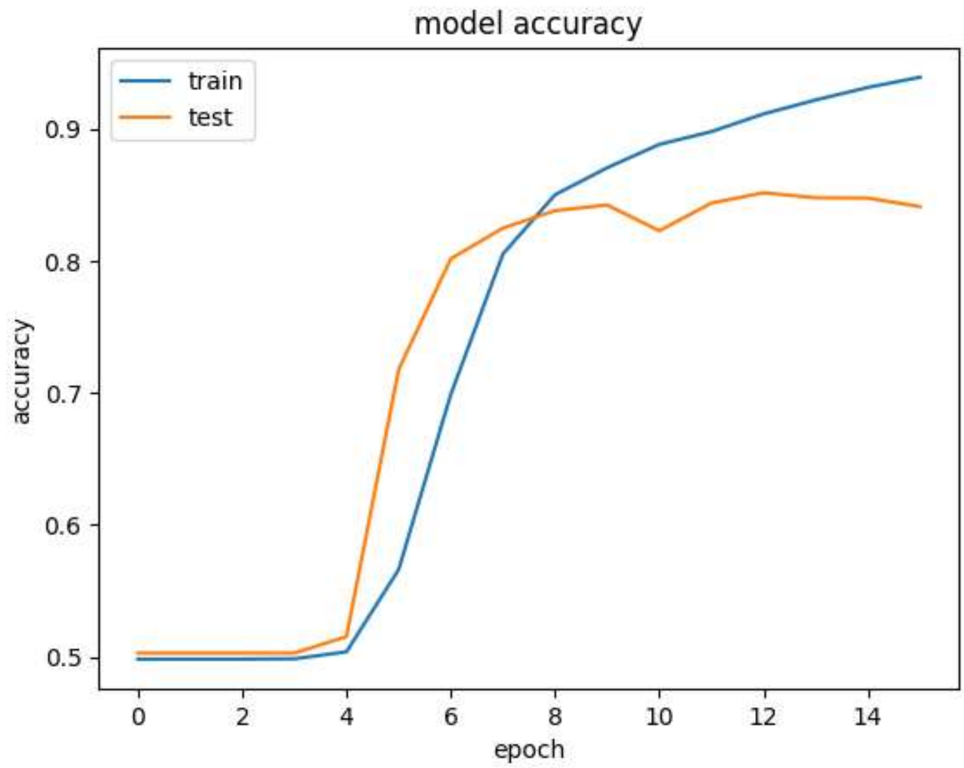


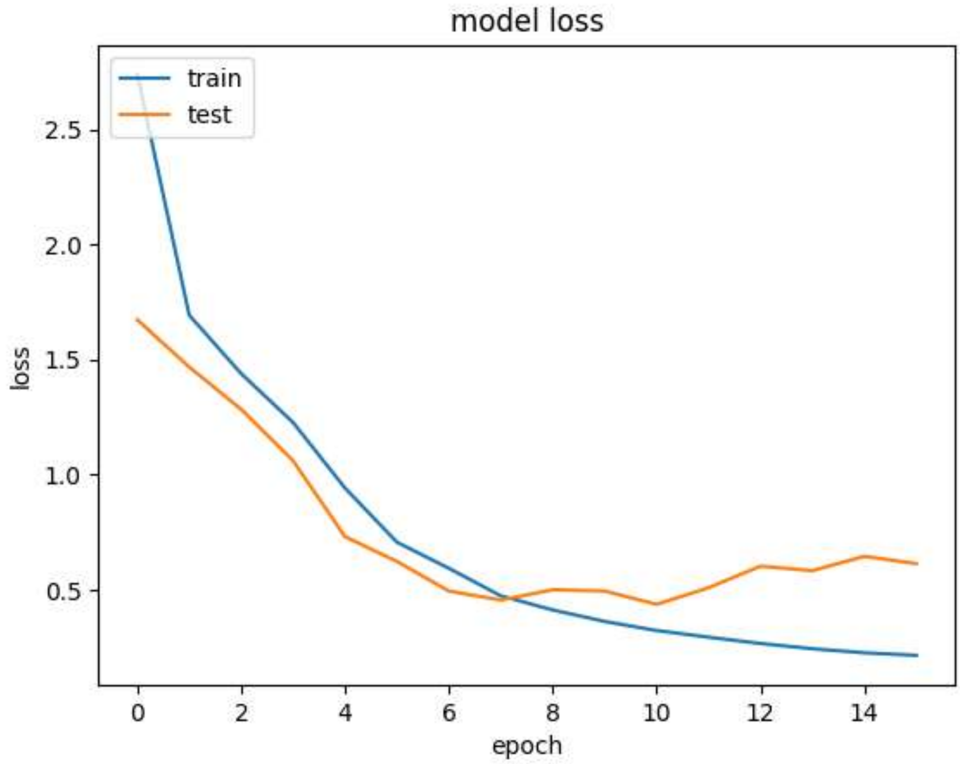
Confusion Matrix

4.b 1-layer LSTM model. [choose 64 or 256 as hidden-vector representation based on the results from 4.a. Report the choice and its justification.]

We have chosen **64-layer LSTM** because accuracy and the performance on both metric 64 hidden vectored model outperformed 256 hidden vector model. Moreover, the training duration of 64 Layer LSTM is less compared to 256 Layer LSTM. From the training graphs it can be understood that 256 hv model was overfitted on train data, as it has more number of parameter.

```
Model: "SingleLSTMLayer64"
_________________________________________________________________
 Layer (type)            Output Shape            Param #
=================================================================
 embedding_14 (Embedding)    (None, 64, 256)        35600128

 lstm_3 (LSTM)             (None, 64)             82176

 dense_20 (Dense)           (None, 64)             4160

 dropout_7 (Dropout)        (None, 64)             0

 dense_21 (Dense)           (None, 1)              65

=================================================================
Total params: 35,686,529
Trainable params: 35,686,529
Non-trainable params: 0
```

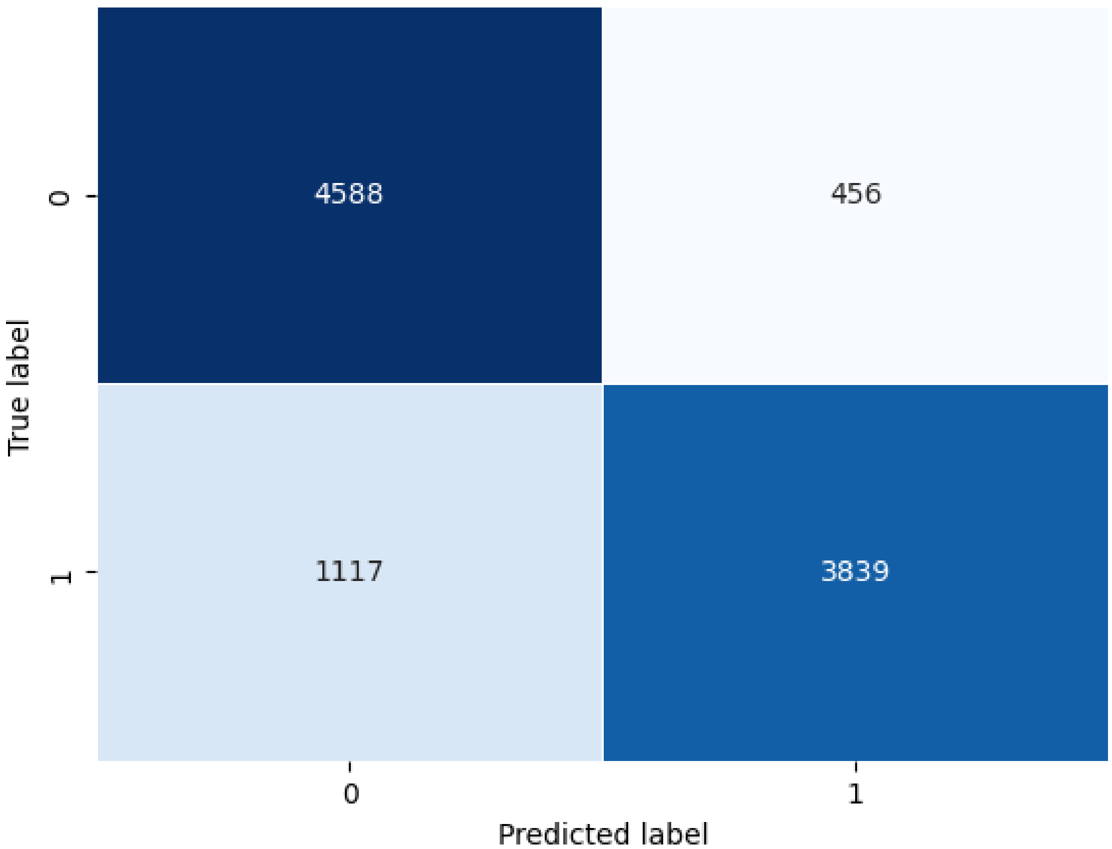

model accuracy
train
test
accuracy
epoch
0.5
0.6
0.7
0.8
0.9
0
2
4
6
8
10
12
14


model loss
train
test
loss
epoch
0.5
1.0
1.5
2.0
2.5
0
2
4
6
8
10
12
14

Classification Report

	precision	recall	f1-score	support
0	0.80	0.91	0.85	5044
1	0.89	0.77	0.83	4956
accuracy			0.84	10000
macro avg	0.85	0.84	0.84	10000
weighted avg	0.85	0.84	0.84	10000

Accuracy 84.27 %

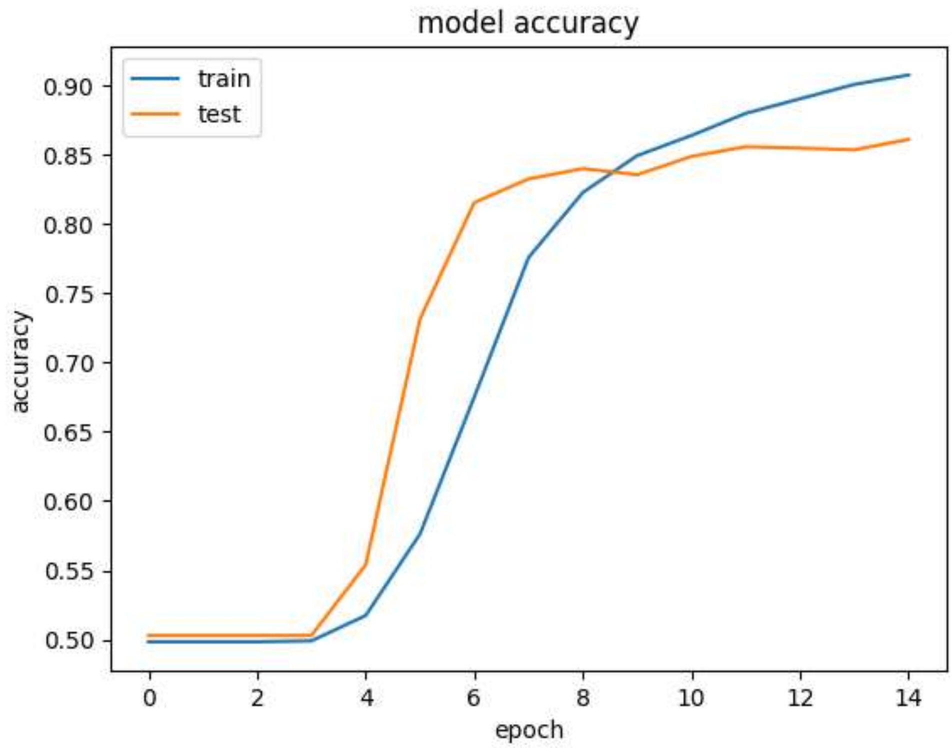


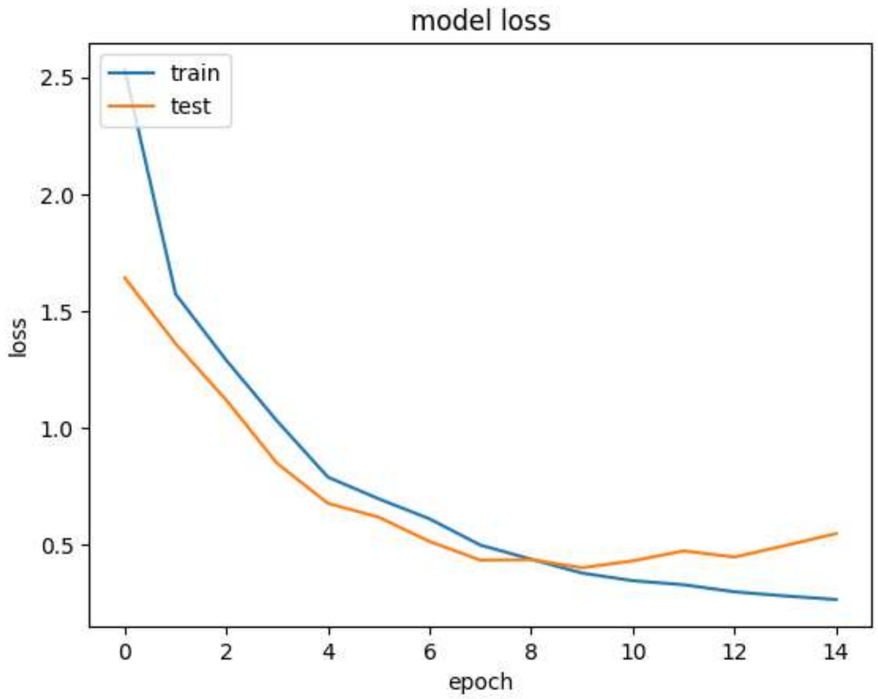
Q.4. Developing Deep neural networks:

c. 2-layer LSTM model

Layer (type)	Output Shape	Param #
embedding_15 (Embedding)	(None, 64, 256)	35600128
lstm_4 (LSTM)	(None, 64, 64)	82176
lstm_5 (LSTM)	(None, 32)	12416
dense_22 (Dense)	(None, 32)	1056
dropout_8 (Dropout)	(None, 32)	0
dense_23 (Dense)	(None, 1)	33

Total params: 35,695,809
Trainable params: 35,695,809
Non-trainable params: 0

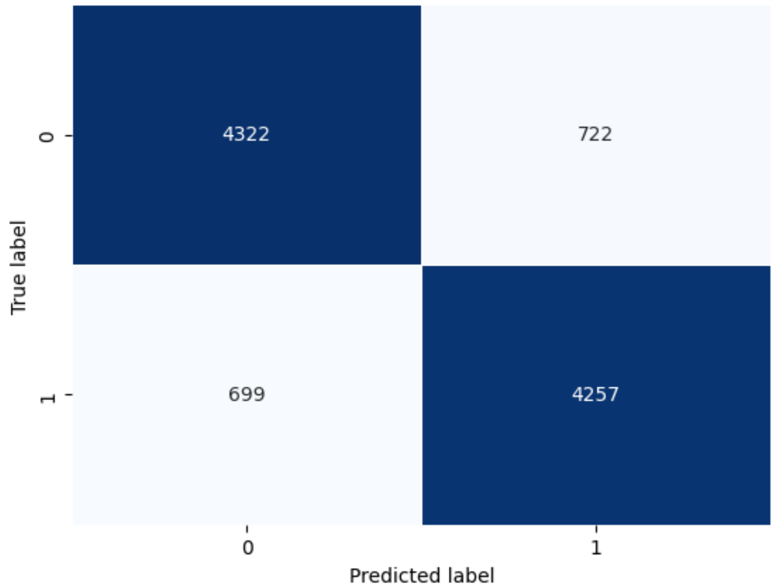

model accuracy
train
test
accuracy
0.50
0.55
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0
2
4
6
8
10
12
14
epoch


model loss
train
test
loss
0.5
1.0
1.5
2.0
2.5
0
2
4
6
8
10
12
14
epoch

Classification Report

	precision	recall	f1-score	support
0	0.86	0.86	0.86	5044
1	0.85	0.86	0.86	4956
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

Accuracy : 85.79 %



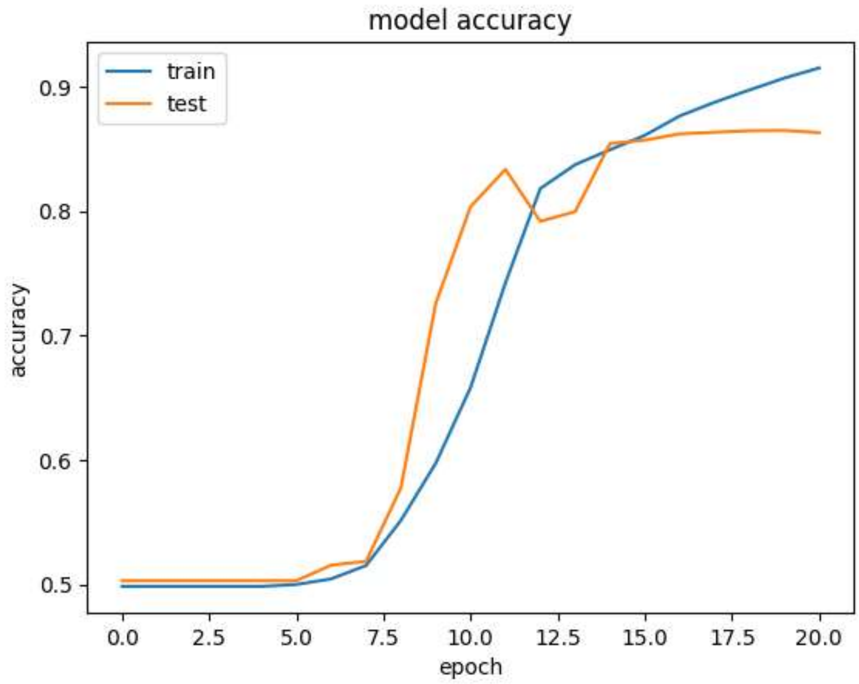
Confusion Matrix

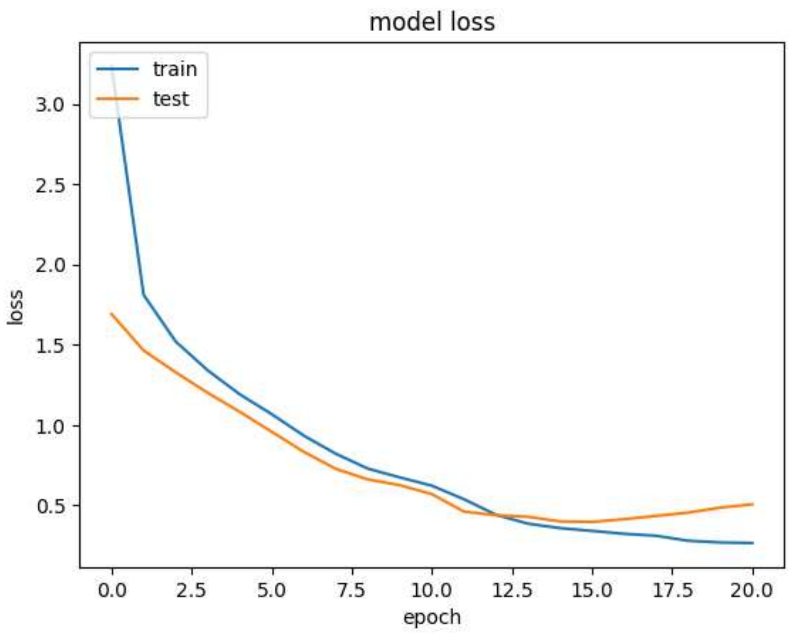
4.d. 1-layer Bi-LSTM model

Model: "SingleBiLSTMModel"

```
_________________________________________________________________
 Layer (type)              Output Shape              Param #
=================================================================
 embedding_17 (Embedding)   (None, 64, 256)          35600128

 bidirectional_1 (Bidirectio  (None, 128)            164352
 nal)

 dense_26 (Dense)           (None, 32)               4128

 dropout_10 (Dropout)       (None, 32)               0

 dense_27 (Dense)           (None, 1)                33

=================================================================
Total params: 35,768,641
Trainable params: 35,768,641
Non-trainable params: 0
_________________________________________________________________
None
```

Hidden Layers of Bi LSTM

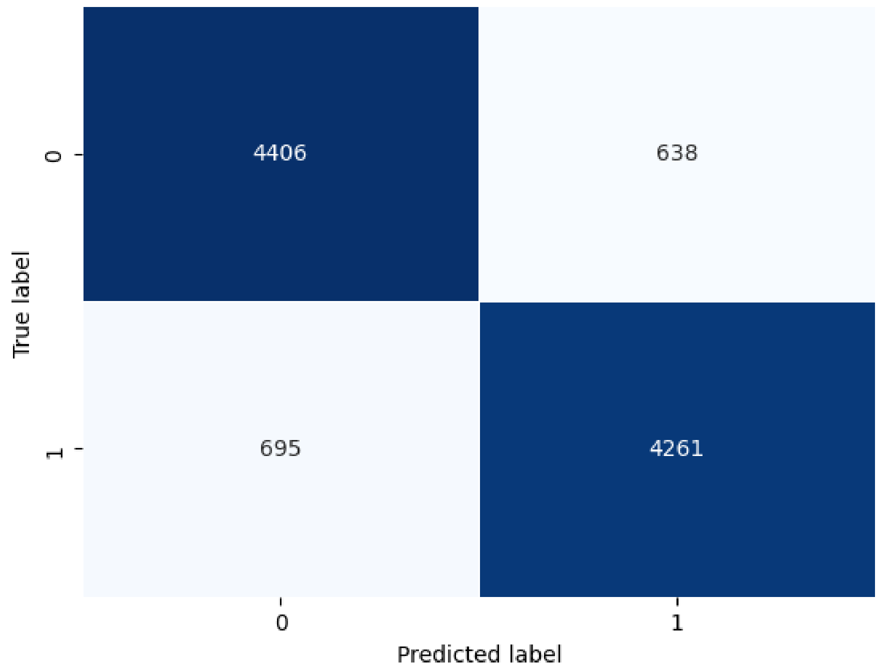




Classification Report

	precision	recall	f1-score	support
0	0.86	0.87	0.87	5044
1	0.87	0.86	0.86	4956
accuracy			0.87	10000
macro avg	0.87	0.87	0.87	10000
weighted avg	0.87	0.87	0.87	10000

Accuracy : 0.8667
Accuracy 86.67 %



Confusion Matrix

4b **1-layer LSTM model.** vs 4d **1-layer Bi-LSTM model.**

1-layer LSTM model Accuracy : 84.27 %

1-layer Bi-LSTM model Accuracy 86.67 %

The performance of Bi_LSTM is better than 1 Layer LSTM because of thefollowing reasons:

1. Bi directional LSTM processes the input sequence in twodirections both forward and backward.

2. Bi-directional LSTM solves vanishing gradient problem in 1 layer LSTM.

3. Bi-directional LSTM has multiple parameters .

4.e Use Google word2vec embeddings as input embedding to model in 4.d.

Model: "SingleBiLSTMModel"

```
________________________________________________________________
_
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 64, 256)           1792

 bidirectional_3 (Bidirectio  (None, 128)              164352
 nal)

 dense_3 (Dense)             (None, 32)                4128

 dropout (Dropout)           (None, 32)                0

 dense_4 (Dense)             (None, 1)                 33

=================================================================
Total params: 170,305
Trainable params: 170,305
Non-trainable params: 0
________________________________________________________________
_
Epoch 1/100
1125/1125 [==============================] - 162s 140ms/step - loss: 3.8885 - accuracy: 0.4994 - val_loss: 2.1759 - val_accuracy: 0.5005
Epoch 2/100
1125/1125 [==============================] - 140s 125ms/step - loss: 2.3539 - accuracy: 0.4985 - val_loss: 1.3062 - val_accuracy: 0.4975
Epoch 3/100
```

1125/1125 [==============================] - 141s 125ms/step - loss: 1.7592 - accuracy: 0.4958 - val_loss: 1.0192 - val_accuracy: 0.4990

...
Epoch 60/100

1125/1125 [==============================] - 309s 274ms/step - loss: 0.4742 - accuracy: 0.8022 - val_loss: 0.4066 - val_accuracy: 0.8487

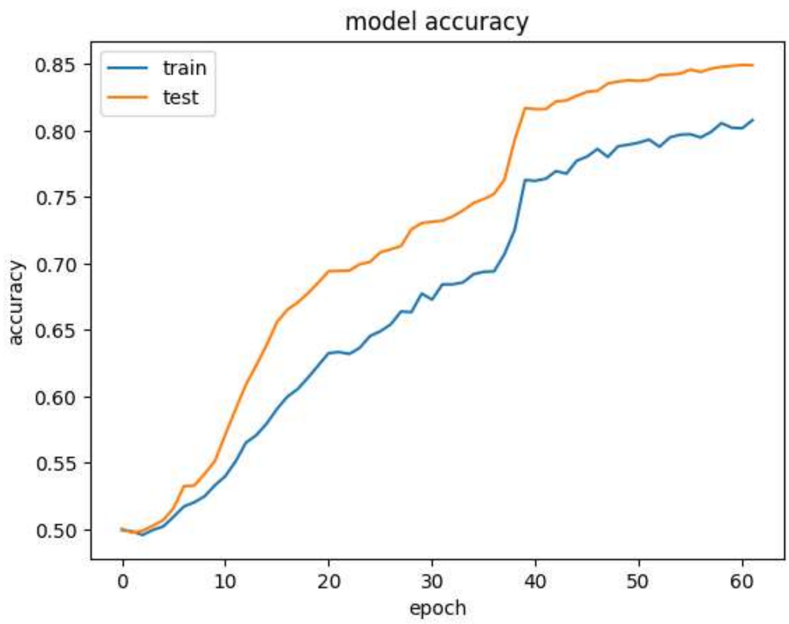
Epoch 61/100

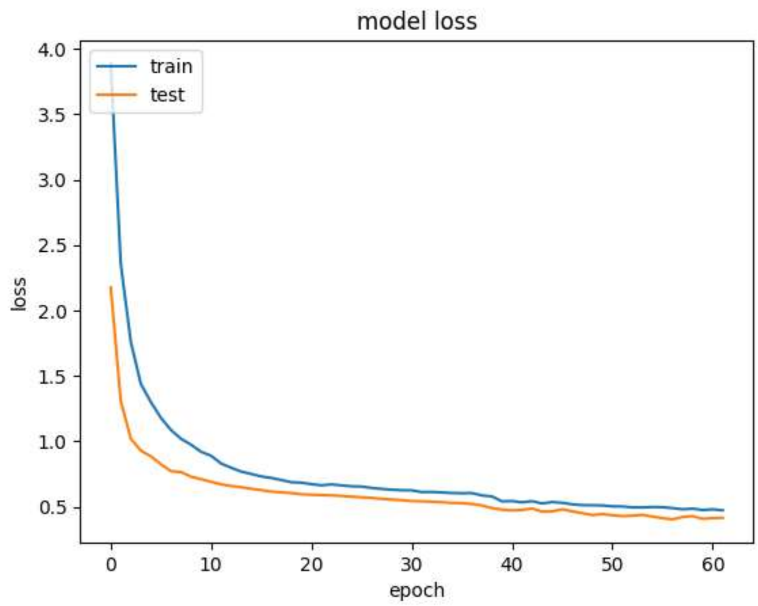
1125/1125 [==============================] - 308s 274ms/step - loss: 0.4792 - accuracy: 0.8019 - val_loss: 0.4123 - val_accuracy: 0.8495

Epoch 62/100

1125/1125 [==============================] - 311s 276ms/step - loss: 0.4721 - accuracy: 0.8079 - val_loss: 0.4143 - val_accuracy: 0.8493

Epoch 62: early stopping

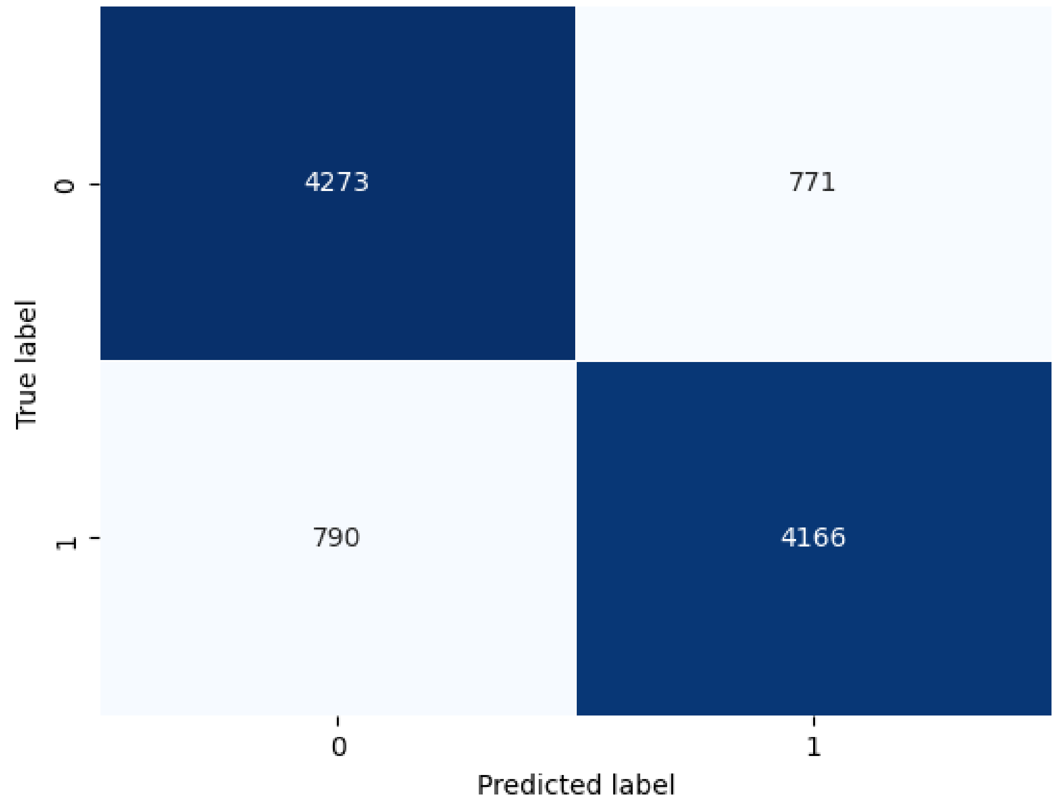




Classification Report

	precision	recall	f1-score	support
0	0.84	0.85	0.85	5044
1	0.84	0.84	0.84	4956
accuracy			0.84	10000
macro avg	0.84	0.84	0.84	10000
weighted avg	0.84	0.84	0.84	10000

Accuracy : 0.8439 (84.39 %)



<u>word2vec embeddings vs 1 Layer Bi LSTM</u>

word2vec accuracy 84.39%

1 layer Bi LSTM 86.6 %

It could be preferable to use a 1-layer Bi-LSTM model rather than word2vec embeddings. This may occur if the task for the model to recognise intricate connections and patterns in the data that are challenging to represent using embeddings alone. By processing the input sequence in both forward and backward directions and updating its hidden states correspondingly, a Bi-LSTM model can learn these patterns and dependencies.

4.f. Use Glove embeddings as input embedding to model in 4.d. [Compare the performance 4.f vs 4.d]

Epoch 1/10 1125/1125 [==============================] - 62s 53ms/step - loss: 0.5142 - accuracy: 0.7524

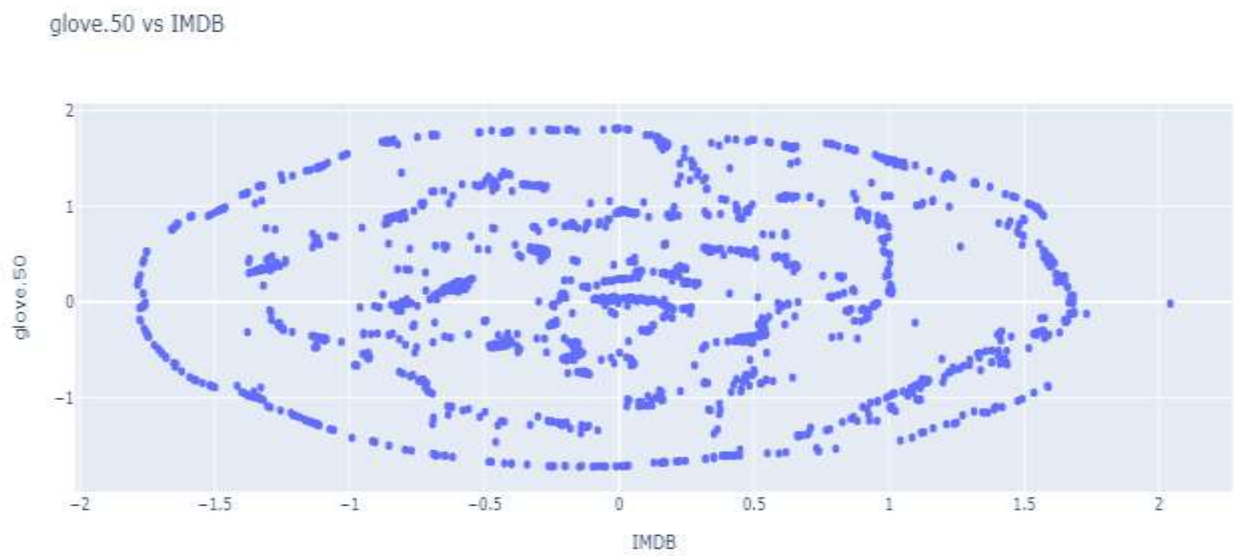
Epoch 2/10 1125/1125 [=================================] - 63s 56ms/step - loss: 0.4158 - accuracy: 0.8123

Epoch 3/10 1125/1125 [=================================] - 58s 52ms/step - loss: 0.3757 - accuracy: 0.8344

Epoch 8/10 1125/1125 [=================================] - 52s 46ms/step - loss: 0.2656 - accuracy: 0.8892

Epoch 9/10 1125/1125 [=================================] - 51s 46ms/step - loss: 0.2448 - accuracy: 0.8985

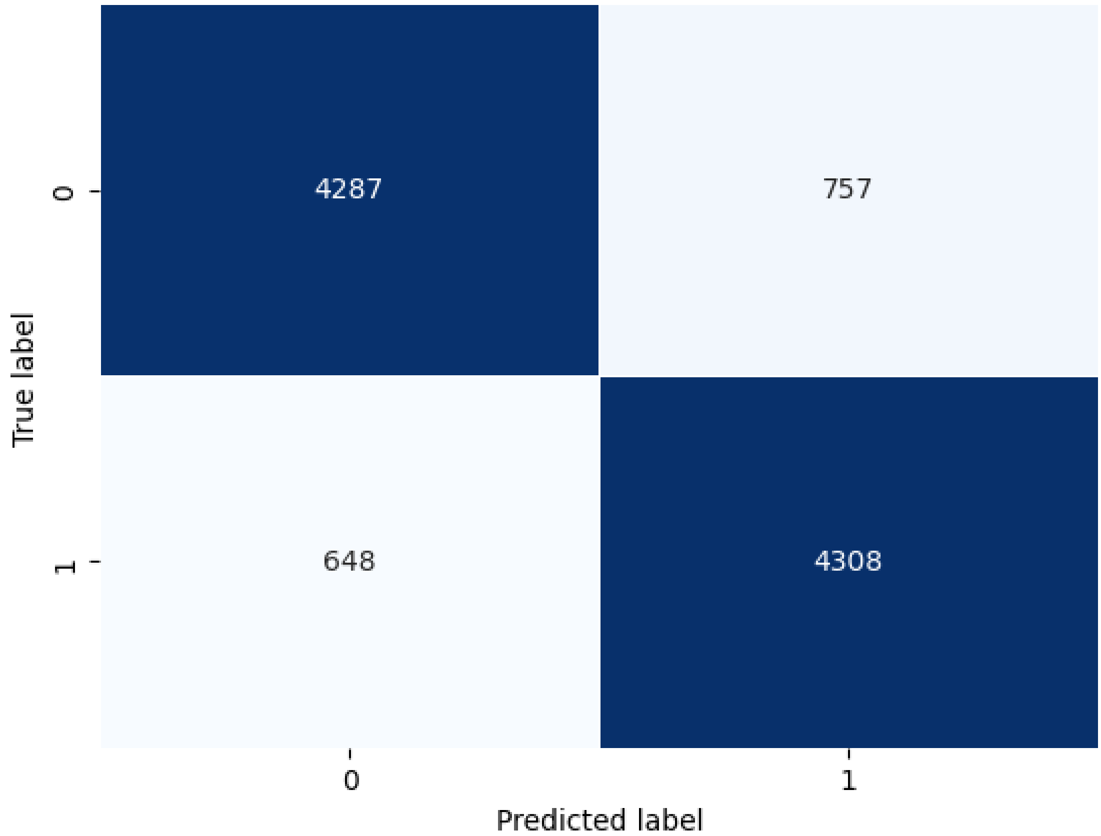
Epoch 10/10 1125/1125 [=================================] - 52s 46ms/step - loss: 0.2282 - accuracy: 0.9069



Classification Report

	precision	recall	f1-score	support
0	0.87	0.85	0.86	5044
1	0.85	0.87	0.86	4956
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

Accuracy : 0.8595 (85.95%)



<u>Comparing Glove Embeddings vs 1 Layer Bi-LSTM</u>

Glove Embeddings Accuracy: **0.8595 (85.95%)**

1 Layer Bi-LSTM Accuracy: **0.8676 (86.76 %)**

Bi directional LSTM processes the input sequence in two directions both forward and backward. Thus, in certain cases Bi-LSTM performs better than Glove Embeddings.

<u>Comparing Glove Embeddings vs Word2vec</u>

Glove Embeddings Accuracy: **0.8595 (85.95%)**

Word2vec Accuracy: 0.8439 **(84.39 %)**

Bi directional LSTM processes the input sequence in two directions both forward and backward. Thus, in certain cases Bi-LSTM performs better than Glove Embeddings.

Contribution:

1. The dataset was mutually decided.
2. The coding of each question was discussed and built by all the members together. And the best performing code was chosen.
3. Knowledge sharing was done on regular basis.
4. Report was prepared part by part by all three members.

Model Architecture

For Auto encoder part

```
autoencoder(
  (encoder): Sequential(
    (0): Linear(in_features=9216, out_features=1024, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1024, out_features=1200, bias=True)
    (3): ReLU()
    (4): Linear(in_features=1200, out_features=728, bias=True)
    (5): ReLU()
    (6): Linear(in_features=728, out_features=512, bias=True)
    (7): ReLU()
    (8): Linear(in_features=512, out_features=128, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=128, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=728, bias=True)
    (3): ReLU()
    (4): Linear(in_features=728, out_features=1200, bias=True)
    (5): ReLU()
    (6): Linear(in_features=1200, out_features=1024, bias=True)
    (7): ReLU()
    (8): Linear(in_features=1024, out_features=9216, bias=True)
    (9): Sigmoid()
  )
)
```

For Classifier part

```
image_classifier(
  (encoder): Sequential(
    (0): Linear(in_features=9216, out_features=1024, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1024, out_features=1200, bias=True)
    (3): ReLU()
    (4): Linear(in_features=1200, out_features=728, bias=True)
```

```
  (5): ReLU()
  (6): Linear(in_features=728, out_features=512, bias=True)
  (7): ReLU()
  (8): Linear(in_features=512, out_features=128, bias=True)
 )
 (fc): Sequential(
  (0): Linear(in_features=128, out_features=10, bias=True)
  (1): ReLU()
 )
)
```

Hyper Parameters for Auto Encoder:

- *Loss Function: MSE Loss*
- *Optimiser: Adam*
- *Learning Rate: 0.001*
- *Number of epochs: 50*

Hyper Parameters for Classifier:

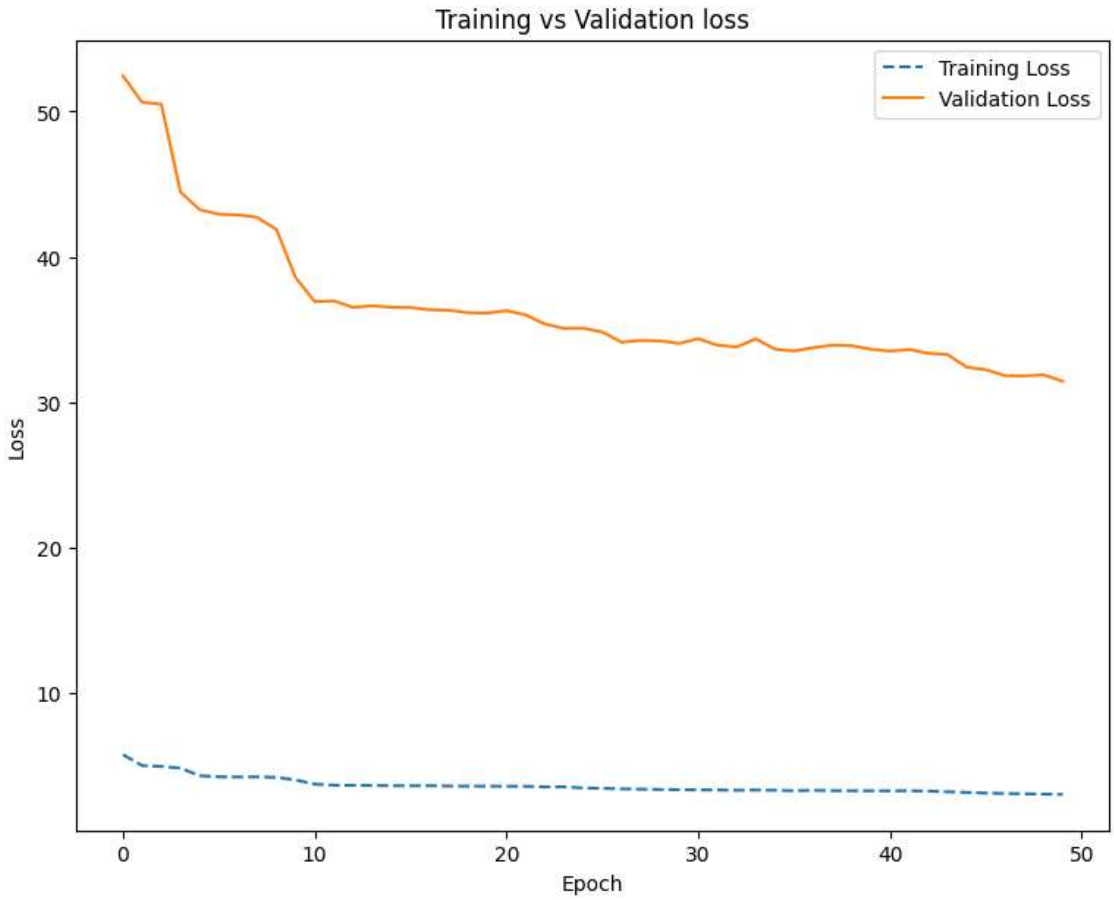
- Loss Function: Cross Entropy Loss
- Optimiser: Adam
- Learning Rate: 0.001
- Number of epochs: 20

Auto Encoder Training

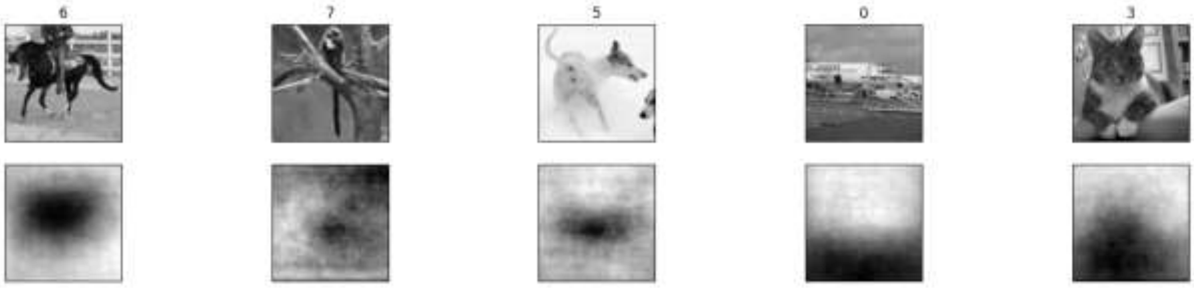
Log:

Epoch: 1 Training Loss: 5.791006, Test Loss: 52.439228, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 2 Training Loss: 5.040079, Test Loss: 50.626852, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 3 Training Loss: 4.983098, Test Loss: 50.500002, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 4 Training Loss: 4.864150, Test Loss: 44.460203, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 5 Training Loss: 4.343365, Test Loss: 43.248069, Training acc: 0.000000, Test acc: 0.000000,

Epoch: 6 Training Loss: 4.266812, Test Loss: 42.935669, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 7 Training Loss: 4.258879, Test Loss: 42.887688, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 8 Training Loss: 4.262150, Test Loss: 42.721353, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 9 Training Loss: 4.228788, Test Loss: 41.894421, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 10 Training Loss: 4.055733, Test Loss: 38.596042, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 11 Training Loss: 3.763021, Test Loss: 36.924113, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 12 Training Loss: 3.691430, Test Loss: 36.973801, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 13 Training Loss: 3.686795, Test Loss: 36.533810, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 14 Training Loss: 3.673056, Test Loss: 36.641978, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 15 Training Loss: 3.655363, Test Loss: 36.534924, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 16 Training Loss: 3.655831, Test Loss: 36.520660, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 17 Training Loss: 3.655898, Test Loss: 36.369693, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 18 Training Loss: 3.633159, Test Loss: 36.328144, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 19 Training Loss: 3.625808, Test Loss: 36.168855, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 20 Training Loss: 3.625555, Test Loss: 36.134638, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 21 Training Loss: 3.617460, Test Loss: 36.310866, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 22 Training Loss: 3.616085, Test Loss: 36.016114, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 23 Training Loss: 3.574844, Test Loss: 35.385124, Training acc: 0.000000, Test acc: 0.000000,
Epoch: 24 Training Loss: 3.577289, Test Loss: 35.084840, Training acc: 0.000000, Test acc: 0.000000,



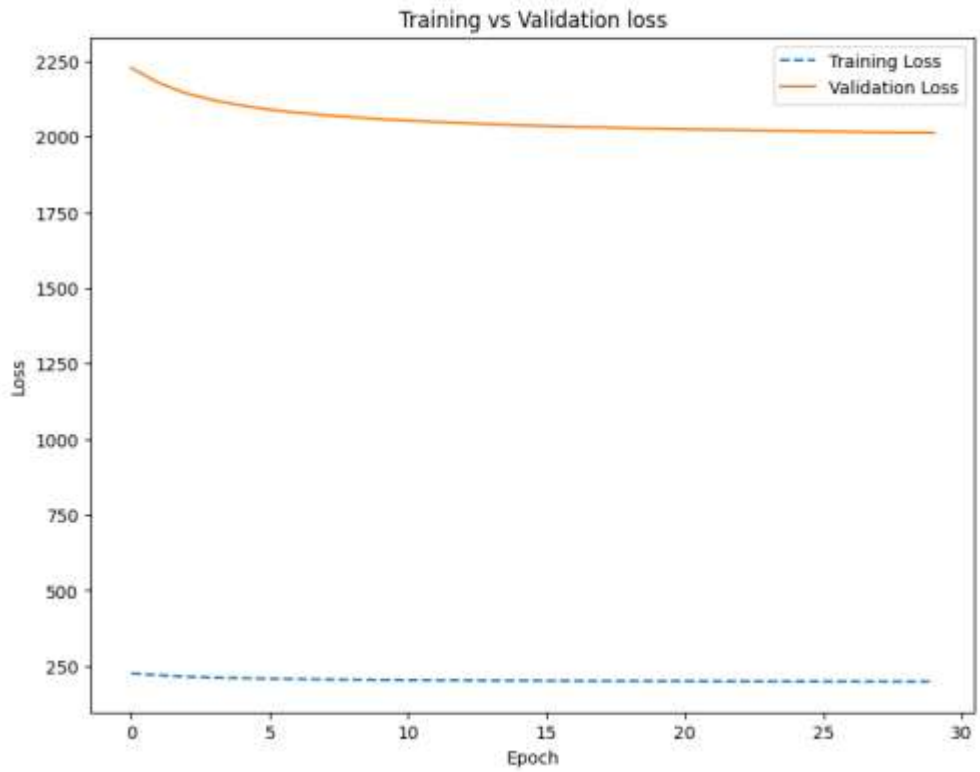
AE Sample prediction after training

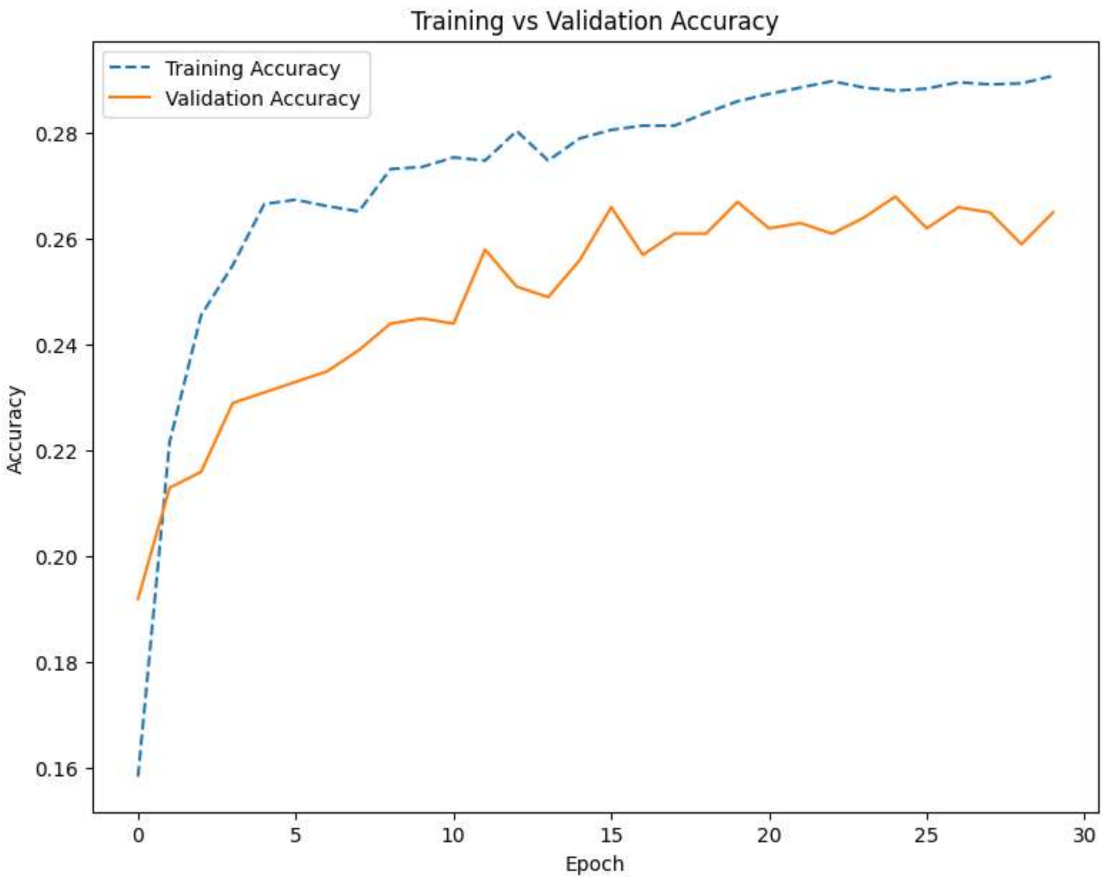


Training Log of Classifier:

Epoch: 1 Training Loss: 226.134165, Test Loss: 2227.391720, Training acc: 0.158400, Test acc: 0.192000,
Epoch: 2 Training Loss: 219.504177, Test Loss: 2177.054644, Training acc: 0.221600, Test acc: 0.213000,
Epoch: 3 Training Loss: 214.973242, Test Loss: 2142.748594, Training acc: 0.245600, Test acc: 0.216000,
Epoch: 4 Training Loss: 211.981044, Test Loss: 2119.860172, Training acc: 0.255000, Test acc: 0.229000,
Epoch: 5 Training Loss: 209.860574, Test Loss: 2102.726221, Training acc: 0.266600, Test acc: 0.231000,
Epoch: 6 Training Loss: 208.242755, Test Loss: 2089.500904, Training acc: 0.267400, Test acc: 0.233000,
Epoch: 7 Training Loss: 207.009069, Test Loss: 2079.362869, Training acc: 0.266200, Test acc: 0.235000,
Epoch: 8 Training Loss: 205.961483, Test Loss: 2070.981264, Training acc: 0.265200, Test acc: 0.239000,
Epoch: 9 Training Loss: 205.147539, Test Loss: 2064.410448, Training acc: 0.273200, Test acc: 0.244000,
Epoch: 10 Training Loss: 204.397602, Test Loss: 2057.743311, Training acc: 0.273600, Test acc: 0.245000,
Epoch: 11 Training Loss: 203.774241, Test Loss: 2053.155184, Training acc: 0.275400, Test acc: 0.244000,
Epoch: 12 Training Loss: 203.218104, Test Loss: 2049.008846, Training acc: 0.274800, Test acc: 0.258000,
Epoch: 13 Training Loss: 202.769477, Test Loss: 2044.994593, Training acc: 0.280400, Test acc: 0.251000,
Epoch: 14 Training Loss: 202.340623, Test Loss: 2041.218996, Training acc: 0.274800, Test acc: 0.249000,
Epoch: 15 Training Loss: 201.971613, Test Loss: 2037.982702, Training acc: 0.279000, Test acc: 0.256000,
Epoch: 16 Training Loss: 201.655735, Test Loss: 2035.289049, Training acc: 0.280600, Test acc: 0.266000,
Epoch: 17 Training Loss: 201.339532, Test Loss: 2032.493591, Training acc: 0.281400, Test acc: 0.257000,
Epoch: 18 Training Loss: 201.044175, Test Loss: 2031.008005, Training acc: 0.281400, Test acc: 0.261000,
Epoch: 19 Training Loss: 200.779714, Test Loss: 2028.291225, Training acc: 0.283800, Test acc: 0.261000,
Epoch: 20 Training Loss: 200.546949, Test Loss: 2026.812077, Training acc: 0.286000, Test acc: 0.267000,
Epoch: 21 Training Loss: 200.314089, Test Loss: 2024.442196, Training acc: 0.287400, Test acc: 0.262000,
Epoch: 22 Training Loss: 200.107319, Test Loss: 2023.317814, Training acc: 0.288600, Test acc: 0.263000,
Epoch: 23 Training Loss: 199.906536, Test Loss: 2021.675825, Training acc: 0.289800, Test acc: 0.261000,
Epoch: 24 Training Loss: 199.729853, Test Loss: 2020.110607, Training acc: 0.288600, Test acc: 0.264000,
Epoch: 25 Training Loss: 199.580452, Test Loss: 2018.836737, Training acc: 0.288000, Test acc: 0.268000,
Epoch: 26 Training Loss: 199.432682, Test Loss: 2017.655134, Training acc: 0.288400, Test acc: 0.262000,

Epoch: 27 Training Loss: 199.256521, Test Loss: 2015.921354, Training acc: 0.289600, Test acc: 0.266000,
Epoch: 28 Training Loss: 199.072124, Test Loss: 2014.232635, Training acc: 0.289200, Test acc: 0.265000,
Epoch: 29 Training Loss: 198.940572, Test Loss: 2013.534307, Training acc: 0.289400, Test acc: 0.259000,
Epoch: 30 Training Loss: 198.803510, Test Loss: 2013.034582, Training acc: 0.290800, Test acc: 0.265000,


Training vs Validation loss
Training Loss
Validation Loss
2250
2000
1750
1500
1250
1000
750
500
250
Loss
0
5
10
15
20
25
30
Epoch


Training vs Validation Accuracy
Training Accuracy
Validation Accuracy
0.28
0.26
0.24
0.22
0.20
0.18
0.16
Accuracy
0
5
10
15
20
25
30
Epoch

Accuracy and Confusion Matrix:

Top 1 Accuracy: 26.500%

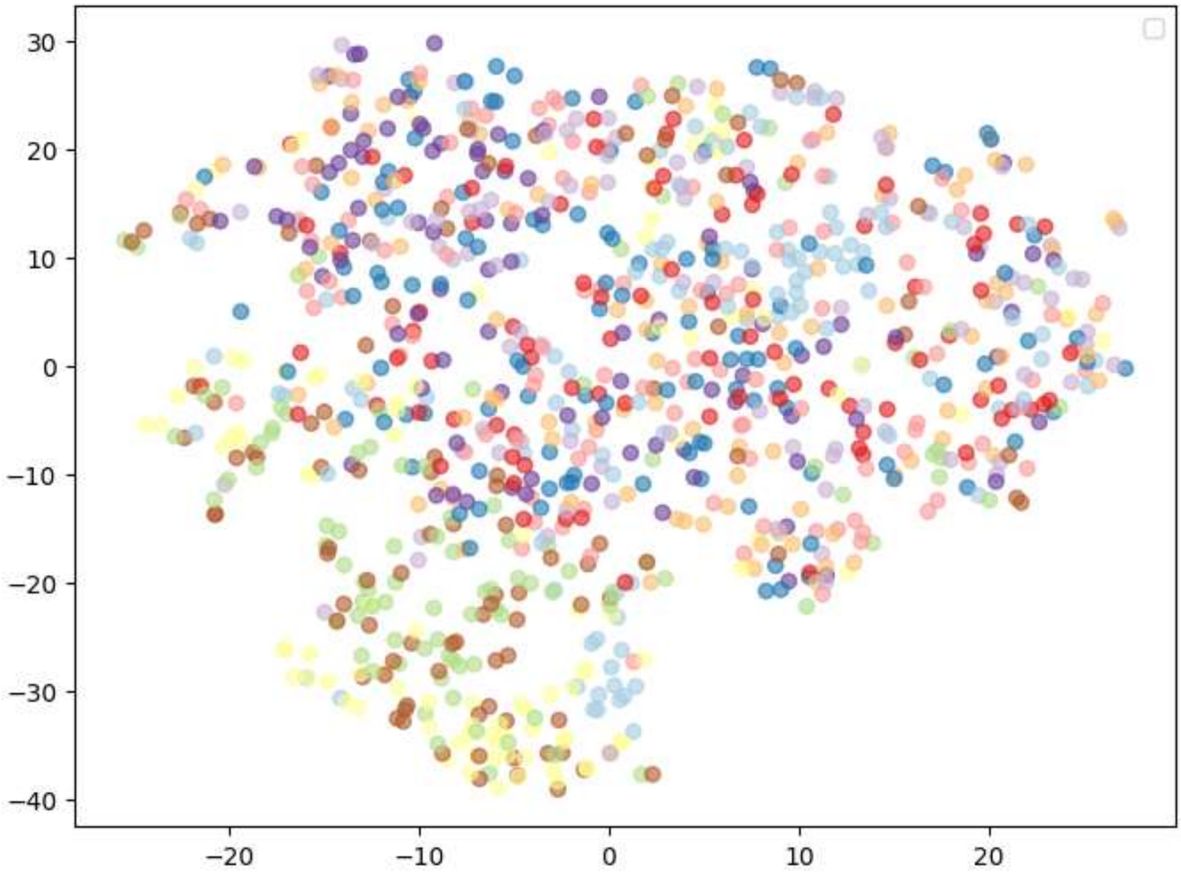
Top 5 Accuracy: 70.5%

Classwise Accuracy Score:

[0.41747573 0.1754386 0.40206186 0.   0.38043478 0.1980198

 0.33684211 0.   0.52427184 0.22916667]

TSNE Plot for the embeddings:



Observation and Conclusion

It can be seen from the results that model did not performed well and reasons for are following:

- Plain Feedforward network unable to extract spatial information from complex colour images. For initial few layer, Convolution Blocks may be used to overcome this issue.
- Lower number of epochs runs due to constrain of computing resources.

Solution of Question 1:

Resnet18

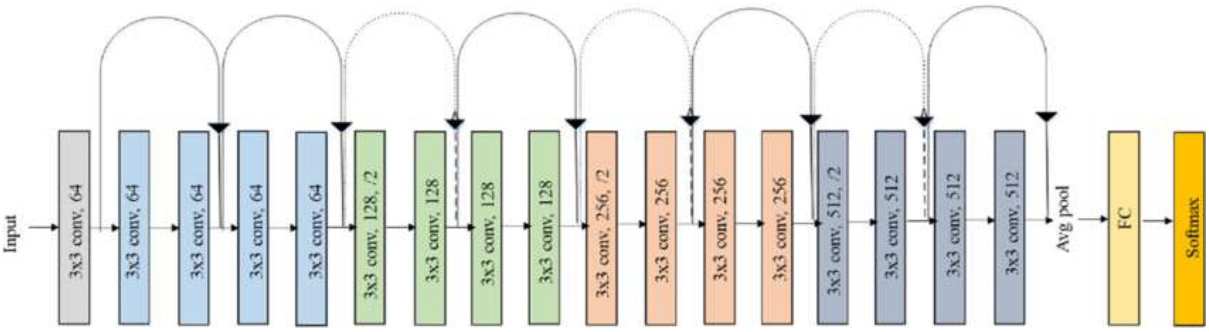
ResNet-18 is a convolutional neural network architecture that was introduced in 2015 as part of the ResNet (Residual Network) family of models. It was developed by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun from Microsoft Research.

The main innovation of ResNet-18 is the use of residual connections, which allow the network to better learn the underlying mapping between the input and output. Instead of trying to learn the mapping directly, the network learns the residual mapping, which is the difference between the input and the desired output. This residual mapping is then added back to the input to obtain the final output.

ResNet-18 consists of 18 layers, including a convolutional layer, four residual blocks, and a fully connected layer. Each residual block contains two convolutional layers with batch normalization and ReLU activation, followed by the addition of the input and the residual mapping. The last layer of the network is a fully connected layer that produces the final output.

ResNet-18 has been widely used for a variety of computer vision tasks, such as image classification, object detection, and semantic segmentation, and has achieved state-of-the-art performance on many benchmark datasets.

Resnet18 Model Architecture:



Original ResNet-18 Architecture

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
```

```
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=10, bias=True)
)
```

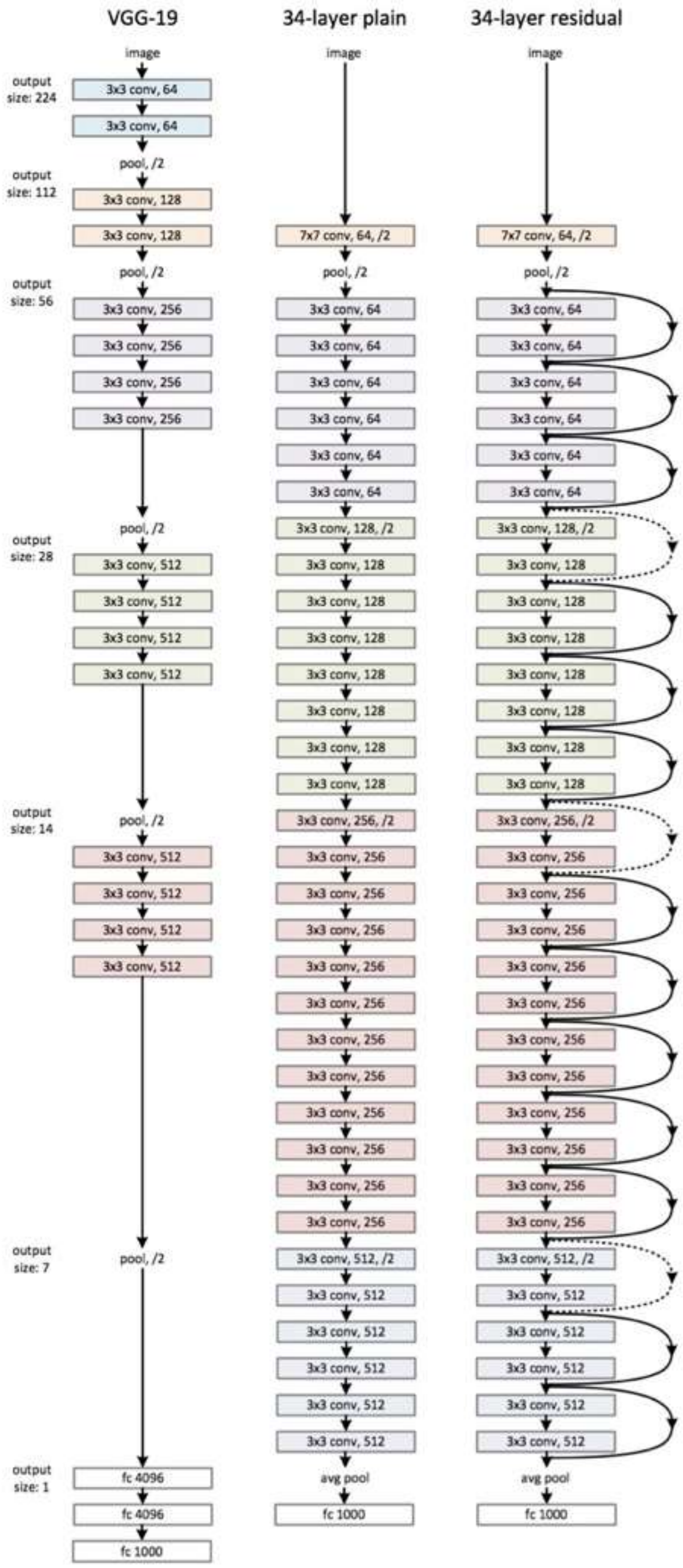


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

ResNet 34 from original paper

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Sizes of outputs and convolutional kernels for ResNet 34

One of the problems ResNets solve is the famous known vanishing gradient. This is because when the network is too deep, the gradients from where the loss function is calculated easily shrink to zero after several applications of the chain rule. This result on the weights never updating its values and therefore, no learning is being performed.

With ResNets, the gradients can flow directly through the skip connections backwards from later layers to initial filters.

Given Dataset:

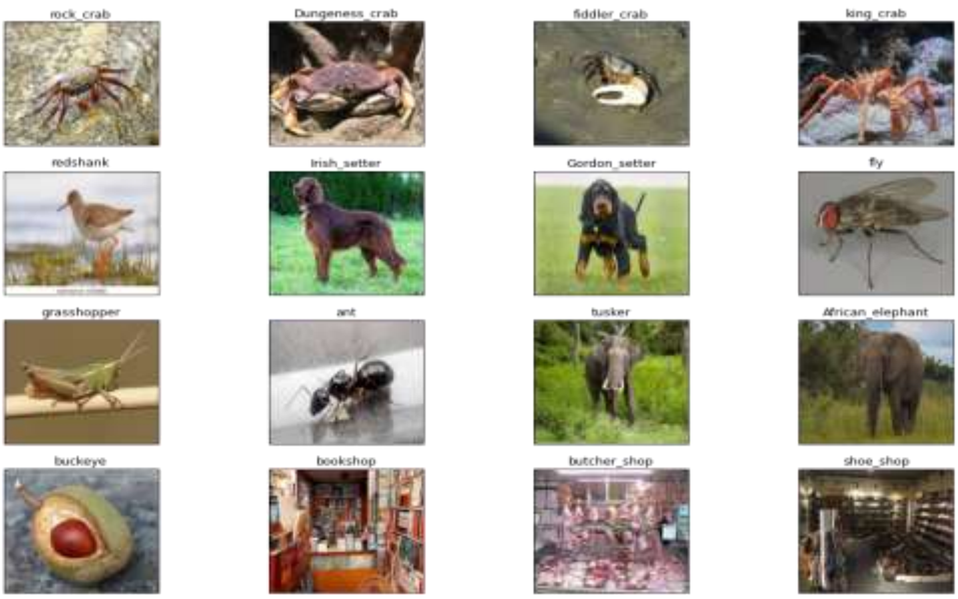
Tiny ImageNet

Training Set: 100000 (500 for each class) Colour (3 channels) 64×64 Images

Testing Set: 100000 (500 for each class) Colour (3 channels) 64×64 Images

Classes: 200 classes

Data Samples:



- ## Optimizer X = Adam, as last digit of my roll no(M21AIE225) is odd

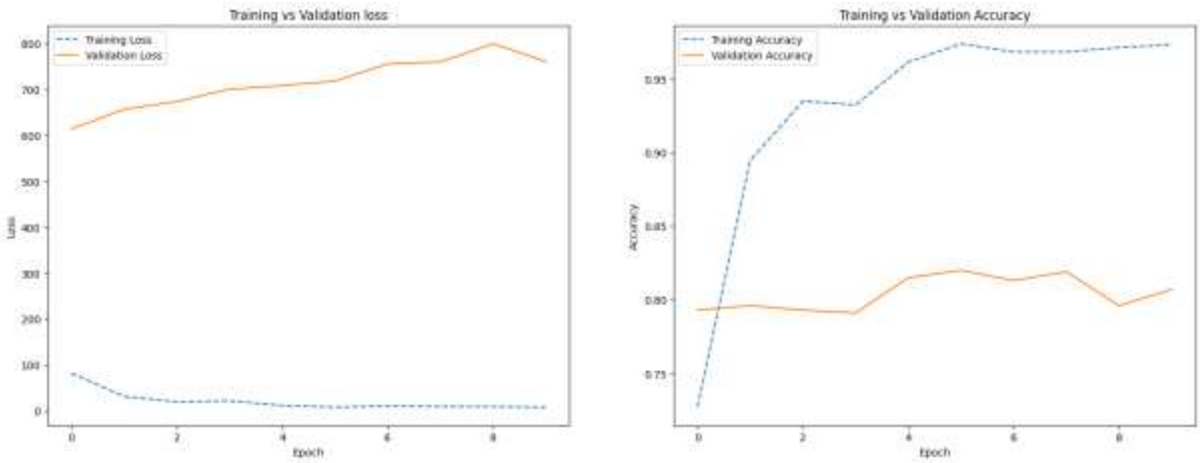
Adam is an optimization algorithm that computes adaptive learning rates for individual weights during training. It maintains a running estimate of the first and second moments of the gradients of the weights, denoted by m and v, respectively. These estimates are initialized as vectors of zeros and updated using a combination of a moving average and bias correction. The optimizer computes bias-corrected estimates of m and v as m_hat = m / (1 - β1^t) and v_hat = v / (1 - β2^t), where β1 and β2 are hyperparameters controlling the exponential decay rates of the estimates, and t is the iteration number. Finally, the optimizer updates the weights using a learning rate α and the bias-corrected estimates of m and v as follows: θ = θ - α * m_hat / (sqrt(v_hat) + ε), where ε is a small constant added to the denominator to avoid division by zero.

- ## Loss function = Cross Entropy

Model Training Log:

Epoch: 1 Training Loss: 82.078650, Test Loss: 614.433289, Training acc: 0.727000, Test acc: 0.793000,
Epoch: 2 Training Loss: 31.350838, Test Loss: 657.145262, Training acc: 0.894600, Test acc: 0.796000,
Epoch: 3 Training Loss: 19.408739, Test Loss: 673.955321, Training acc: 0.934800, Test acc: 0.793000,
Epoch: 4 Training Loss: 21.767490, Test Loss: 700.918734, Training acc: 0.932200, Test acc: 0.791000,
Epoch: 5 Training Loss: 11.681993, Test Loss: 709.099174, Training acc: 0.961400, Test acc: 0.815000,
Epoch: 6 Training Loss: 7.955046, Test Loss: 718.438625, Training acc: 0.973800, Test acc: 0.820000,
Epoch: 7 Training Loss: 10.227830, Test Loss: 755.967617, Training acc: 0.968200, Test acc: 0.813000,
Epoch: 8 Training Loss: 9.303071, Test Loss: 760.519803, Training acc: 0.968200, Test acc: 0.819000,
Epoch: 9 Training Loss: 9.134530, Test Loss: 800.109386, Training acc: 0.971200, Test acc: 0.796000,
Epoch: 10 Training Loss: 7.732154, Test Loss: 761.471570, Training acc: 0.973200, Test acc: 0.807000,

Model Training Losses and Accuracies:



Accuracy:

Top 1 Accuracy: 80.700%
Top 5 Accuracy: 98.8%

Class wise Accuracy Score:

[0.89320388 0.78070175 0.90721649 0.69811321 0.82608696 0.58415842

0.82105263 0.80645161 0.9223301 0.84375]

Q. 1.2. Triplet Loss with hard mining as the final classification loss function:

Triplet Loss:

Triplet Loss is a loss function, where the goal is to learn a mapping from the input space to a metric space, such that similar inputs are mapped close to each other and dissimilar inputs are mapped far apart.

The Triplet Loss function involves selecting triplets of examples (anchor, positive, and negative), and then minimizing the distance between the anchor and positive examples while maximizing the distance between the anchor and negative examples. This can be expressed as:

L_triplet = max(0, d(a,p) - d(a,n) + margin)

where d(a,p) is the distance between the anchor (a) and positive (p) examples, d(a,n) is the distance between the anchor (a) and negative (n) examples, and margin is a hyperparameter that determines the minimum difference between the distances.

In order to make the Triplet Loss function more effective for classification tasks, hard mining can be used. Hard mining involves selecting the hardest negative example for each anchor-positive pair, i.e., the negative example that has the smallest distance to the anchor among all negative examples. This makes the loss function more discriminative, as it focuses on the most difficult examples to classify.

Therefore, using Triplet Loss with hard mining as the final classification loss function can be an effective way to learn a metric space that is optimized for classification tasks.

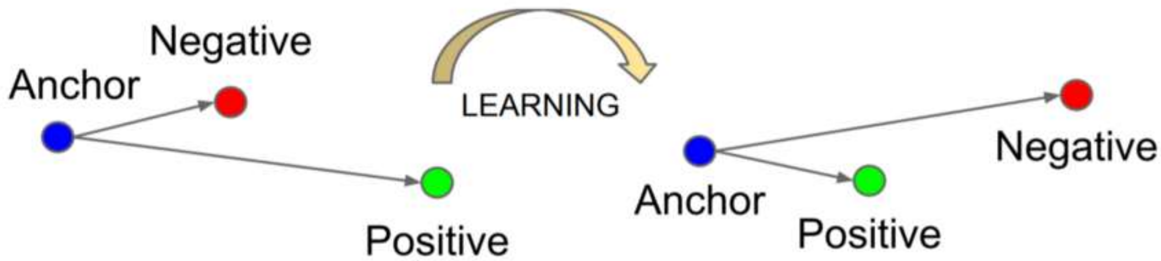


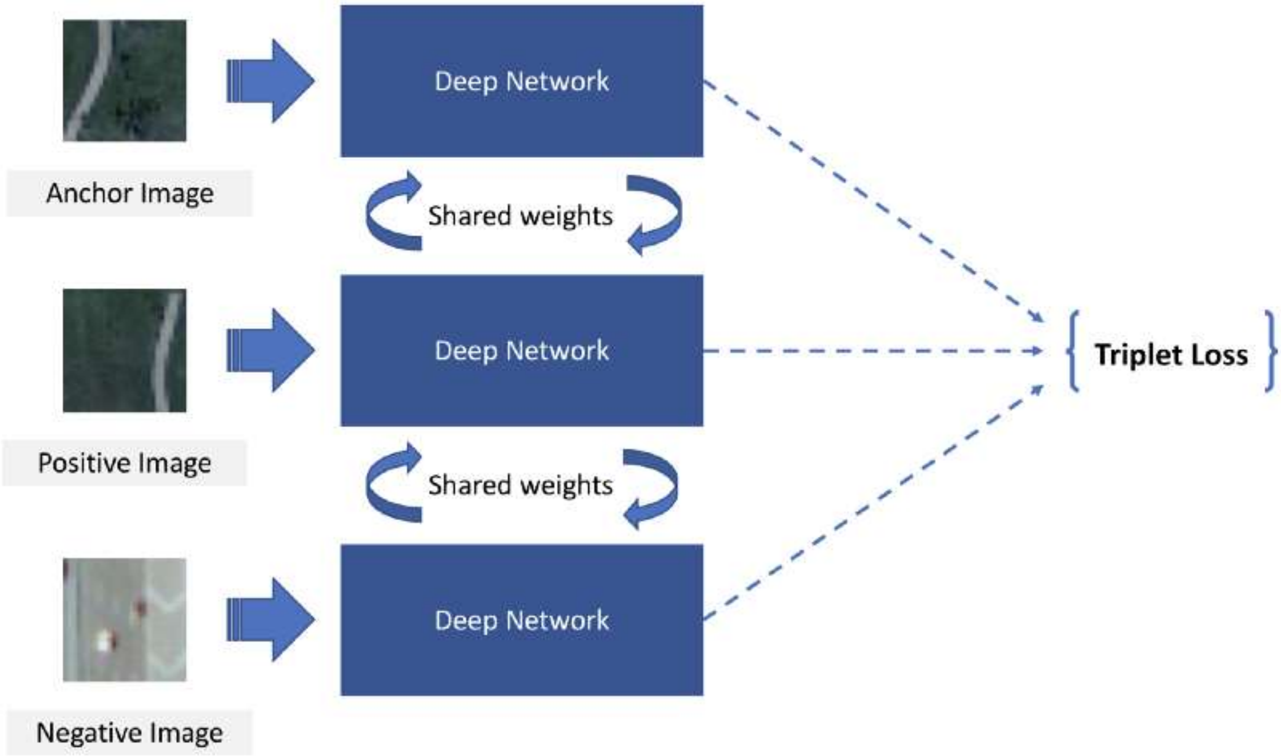
Image Source: medium.com

$$\sum_{i}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]$$

Mathematical Equation of Triplet Loss Function.

- ***f(x)*** takes ***x*** as an input and returns a vector ***w.***
- ***i*** denotes *i'th* input.
- Subscript ***a*** indicates ***Anchor*** image, ***p*** indicates ***Positive*** image, ***n*** indicates ***Negative*** image.

Our objective is to *minimize the above equation*, which implicitly means:-
Minimizing first term → distance between Anchor and Positive image.
Maximizing(since it has negative sign before it) second term → distance between Anchor and Negative image.



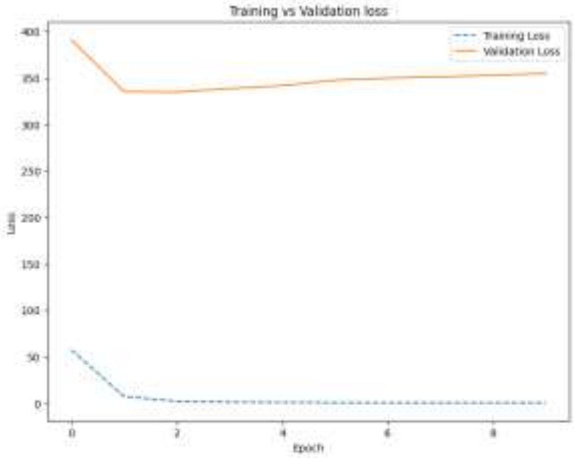
Triplet Loss architecture

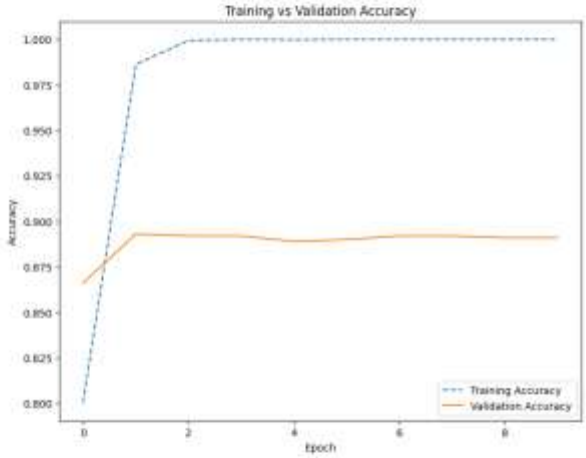
Training and Observations:

Model Training Log:

Epoch: 1 Training Loss: 57.339447, Test Loss: 391.216218, Training acc: 0.800400, Test acc: 0.866000,
Epoch: 2 Training Loss: 7.036367, Test Loss: 335.199594, Training acc: 0.986200, Test acc: 0.893000,
Epoch: 3 Training Loss: 2.061597, Test Loss: 334.692627, Training acc: 0.999400, Test acc: 0.892000,
Epoch: 4 Training Loss: 1.033667, Test Loss: 338.609815, Training acc: 1.000000, Test acc: 0.892000,
Epoch: 5 Training Loss: 0.771824, Test Loss: 341.696292, Training acc: 0.999800, Test acc: 0.889000,
Epoch: 6 Training Loss: 0.554433, Test Loss: 347.637147, Training acc: 1.000000, Test acc: 0.890000,
Epoch: 7 Training Loss: 0.439116, Test Loss: 349.825233, Training acc: 1.000000, Test acc: 0.892000,
Epoch: 8 Training Loss: 0.407432, Test Loss: 351.264626, Training acc: 1.000000, Test acc: 0.892000,
Epoch: 9 Training Loss: 0.375267, Test Loss: 352.931619, Training acc: 1.000000, Test acc: 0.891000,
Epoch: 10 Training Loss: 0.298735, Test Loss: 354.932994, Training acc: 1.000000, Test acc: 0.891000,

Model Training Losses and Accuracies:



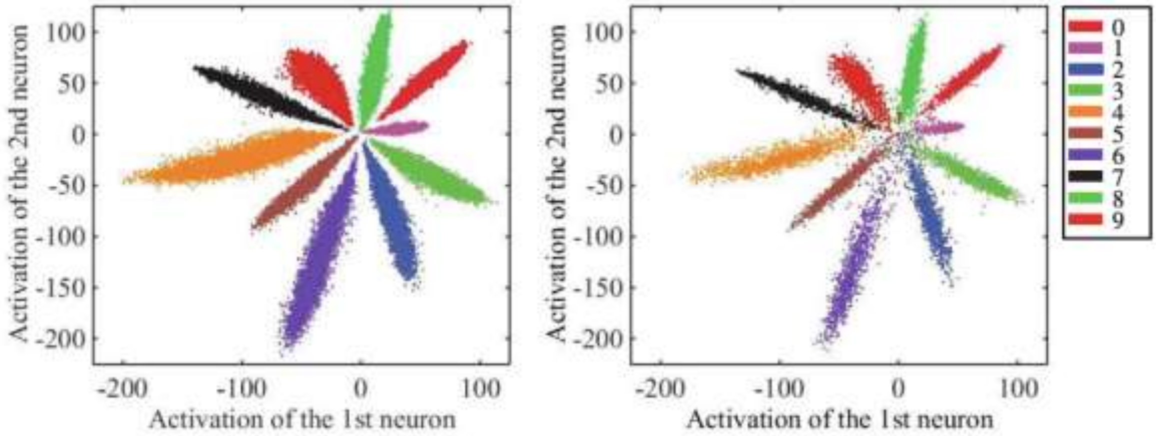


Accuracy:

Top 1 Accuracy: 89.100%
Top 5 Accuracy: 99.6%
Classwise Accuracy Score:
[0.91262136 0.90350877 0.96907216 0.77358491 0.81521739 0.82178218
 0.91578947 0.94623656 0.95145631 0.90625]

Q. 1.3. Center Loss:

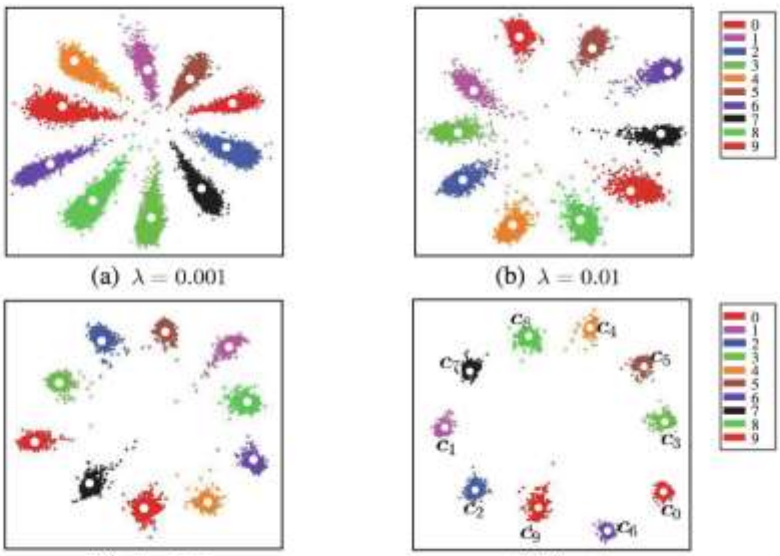
Center loss is a strategy for constructing widely-separated classes.A common problem with ordinary supervised learning is that the latent features for the classes can end up being tightly grouped. This can be undesirable, because a small change in the input can cause an example to shift from one side of the class boundary to the other.



(a) (b)

This plot from the paper shows how the activations of 2 neurons display the (a) training data and (b) testing data. Especially near the coordinate (0,0), there is an undesirable mixing of the 10 classes, which contributes to a larger error on the test set.

By contrast, the center loss is a regularization strategy that encourages the model to learn widely-separated class representations. The center loss augments the standard supervised loss by adding a penalty term proportional to the distance of a class's examples from its center.



(a) $\lambda = 0.001$ (b) $\lambda = 0.01$

(c) $\lambda = 0.1$ (d) $\lambda = 1$

The center loss includes a hyperparameter λ� which controls the strength of the regularization. Increasing λ� increases the separation of the class centers.

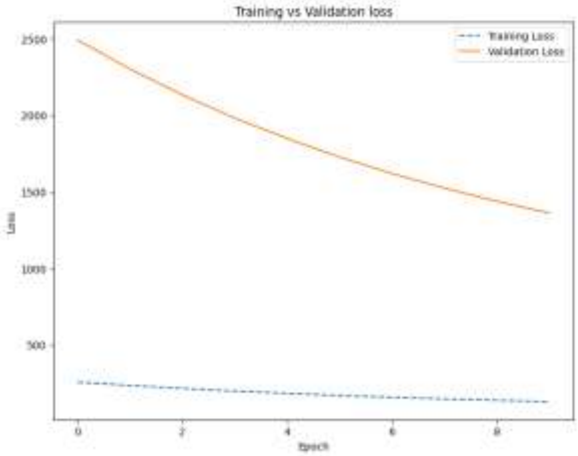
These images were taken from Yandong Wen et al., "A Discriminative Feature Learning Approach for Deep Face Recognition" (2016).

Training and Observations:

Model Training Log:

Epoch: 1 Training Loss: 256.169827, Test Loss: 2491.599083, Training acc: 0.088800, Test acc: 0.106000,
Epoch: 2 Training Loss: 234.733407, Test Loss: 2301.831961, Training acc: 0.150600, Test acc: 0.166000,
Epoch: 3 Training Loss: 215.314672, Test Loss: 2132.591963, Training acc: 0.234600, Test acc: 0.258000,
Epoch: 4 Training Loss: 198.007964, Test Loss: 1981.519461, Training acc: 0.313200, Test acc: 0.327000,
Epoch: 5 Training Loss: 183.040474, Test Loss: 1846.614122, Training acc: 0.390400, Test acc: 0.379000,
Epoch: 6 Training Loss: 169.069120, Test Loss: 1725.814939, Training acc: 0.466200, Test acc: 0.438000,
Epoch: 7 Training Loss: 157.340216, Test Loss: 1618.575931, Training acc: 0.525200, Test acc: 0.484000,
Epoch: 8 Training Loss: 146.545958, Test Loss: 1522.536397, Training acc: 0.575000, Test acc: 0.532000,
Epoch: 9 Training Loss: 137.393952, Test Loss: 1436.814308, Training acc: 0.617800, Test acc: 0.567000,
Epoch: 10 Training Loss: 129.242647, Test Loss: 1360.561848, Training acc: 0.652200, Test acc: 0.601000,

Model Training Losses and Accuracies:



Confusion Matrix and Accuracy:

Top 1 Accuracy: 60.100%
Top 5 Accuracy: 94.6%
Classwise Accuracy Score:
[0.66019417 0.53508772 0.88659794 0.41509434 0.5 0.45544554
0.55789474 0.64516129 0.85436893 0.51041667]

References:

1. Lectures of NLU class
2. Blogs from Internet

    a. https://sabber.medium.com/classifying-yelp-review-comments-using-cnn-lstm-and-pre-trained-glove-word-embeddings-part-3-53fcea9a17fa
    b. https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/
    c. https://medium.com/swlh/sentiment-classification-for-restaurant-reviews-using-tf-idf-42f707bfe44d
    d. https://www.youtube.com/watch?v=D2V1okCEsiE&t=169s
    e. https://www.tensorflow.org/guide/keras/rnn
    f. https://reader.elsevier.com/reader/sd/pii/S187705091831439X?token=C80CEF15EFECC5B9FAE297D64CE50EFA7AAC055038951C8EB82D867A83B1AEF1CE0EE4E3535E057DE4DEA9B033AE1E8B&originRegion=eu-west-1&originCreation=20230301154620