CSE674 Project 2: Writer Verification, Explainable AI

Nikhil Srihari
UB Person ID : **50291966**
Department of Computer Science,
SUNY – UB, Buffalo, NY
nikhilsr@buffalo.edu

Abstract

In this report, I document the output and analysis of Writer verification with 4 different AI models - Bayesian Model, Siamese Network, Autoencoder network and Multitask Trainable Encoder Network – on 3 different 'and' data sets. I also briefly touch upon the theory behind these models. This project contains 4 tasks. Let us discuss them by one by one.

1 Introduction

The purpose of this project is to document the output and analysis of Writer verification with 4 different AI models - Bayesian Model, Siamese Network, Autoencoder network and Multitask Trainable Encoder Network – on 3 different data sets. Our primary task, i.e., the Writer Verification task is simple. Given 2 images, determine if they are from the same writer or from different writers.

Task 1 was to annotate the training data. Task 2 involved creating and training a Bayesian network for each of the 3 data sets and noting down the different metrics. Task 3 involves creating and training Deep Learning models for each of the 3 data sets and noting down the different metrics. Task 4 consists of the same task as Task 2, but with 'and' dataset. Finally, Task 4 involves creating and training Explainable AI models for each of the 3 data sets and noting down the different metrics. Let us begin our discussion with Task 2

2 Task 2 – Bayesian Network

Here we build a Bayesian network to determine if two images are from the same writer or from a different writer.

Method 1 - We start by creating a Bayesian Network on the 15 features of images using Hill Climb Search. We then create 2 copies of this network and add the 'label' node to the network and connect it intuitively to some nodes. We add the 'label' node in such a way that the resulting final network is symmetric and that the final network connects the 2 copies of the initial network through the 'label' nodes. We train the final network and estimate the CPDs using our training data. On this trained network, we use map query to predict label given the 2 images. We calculate the training and validation accuracy. I have implemented this method here.

```
BayesianModel([('fpen_pressure', 'fis_lowercase'), ('fpen_pressure', 'fletter_spacing'), ('fsize', 'fslantness'), ('fsize', 'fpen_pressure'),
               ('fsize', 'fstaff_of_d'), ('fsize', 'fletter_spacing'), ('fsize', 'fexit_stroke_d'), ('fsize', 'fentry_stroke_a'),
               ('fdimension', 'fsize'), ('fdimension', 'fis_continuous'), ('fdimension', 'fslantness'), ('fdimension', 'fpen_pressure'),
               ('fis_lowercase', 'fstaff_of_a'), ('fis_lowercase', 'fexit_stroke_d'), ('fis_continuous', 'fexit_stroke_d'), ('fis_continuous', 'fletter_spacing'),
               ('fis_continuous', 'fentry_stroke_a'), ('fis_continuous', 'fstaff_of_a'), ('fis_continuous', 'fis_lowercase'), ('fslantness', 'fis_continuous'),
               ('fslantness', 'ftilt'), ('fentry_stroke_a', 'fpen_pressure'), ('fformation_n', 'fconstancy'), ('fformation_n', 'fword_formation'), ('fformation_n', 'fdimension'),
               ('fformation_n', 'fstaff_of_d'), ('fformation_n', 'fis_continuous'), ('fformation_n', 'fsize'), ('fformation_n', 'fstaff_of_a'), ('fstaff_of_d', 'fis_continuous'),
               ('fstaff_of_d', 'fexit_stroke_d'), ('fstaff_of_d', 'fis_lowercase'), ('fstaff_of_d', 'fslantness'), ('fstaff_of_d', 'fentry_stroke_a'),
               ('fword_formation', 'fdimension'), ('fword_formation', 'fstaff_of_a'), ('fword_formation', 'fsize'), ('fword_formation', 'fstaff_of_d'),
               ('fword_formation', 'fconstancy'), ('fconstancy', 'fstaff_of_a'), ('fconstancy', 'fletter_spacing'), ('fconstancy', 'fdimension'),
               ('gpen_pressure', 'gis_lowercase'), ('gpen_pressure', 'gletter_spacing'), ('gsize', 'gslantness'), ('gsize', 'gpen_pressure'),
               ('gsize', 'gstaff_of_d'), ('gsize', 'gletter_spacing'), ('gsize', 'gexit_stroke_d'), ('gsize', 'gentry_stroke_a'), ('gdimension', 'gsize'),
               ('gdimension', 'gis_continuous'), ('gdimension', 'gslantness'), ('gdimension', 'gpen_pressure'), ('gis_lowercase', 'gstaff_of_a'),
               ('gis_lowercase', 'gexit_stroke_d'), ('gis_continuous', 'gexit_stroke_d'), ('gis_continuous', 'gletter_spacing'), ('gis_continuous', 'gentry_stroke_a'),
               ('gis_continuous', 'gstaff_of_a'), ('gis_continuous', 'gis_lowercase'), ('gslantness', 'gis_continuous'), ('gslantness', 'gtilt'),
               ('gentry_stroke_a', 'gpen_pressure'), ('gformation_n', 'gconstancy'), ('gformation_n', 'gword_formation'), ('gformation_n', 'gdimension'),
               ('gformation_n', 'gstaff_of_d'), ('gformation_n', 'gis_continuous'), ('gformation_n', 'gsize'), ('gformation_n', 'gstaff_of_a'), ('gstaff_of_d', 'gis_continuous'),
               ('gstaff_of_d', 'gexit_stroke_d'), ('gstaff_of_d', 'gis_lowercase'), ('gstaff_of_d', 'gslantness'), ('gstaff_of_d', 'gentry_stroke_a'),
               ('gword_formation', 'gdimension'), ('gword_formation', 'gstaff_of_a'), ('gword_formation', 'gsize'), ('gword_formation', 'gstaff_of_d'),
               ('gword_formation', 'gconstancy'), ('gconstancy', 'gstaff_of_a'), ('gconstancy', 'gletter_spacing'), ('gconstancy', 'gdimension'),
               ('fis_continuous', 'label'), ('fword_formation','label'),
               ('gis_continuous', 'label'), ('gword_formation','label')])
```

Fig 2.1: Bayesian Network

Method 2 – In this method, we take the corresponding features of each image and calculate the CPDs of a similarity node (for each pair of corresponding features). We build a network from these similarity nodes, connecting finally to our true dependent variable, i.e., the 'label' node. We estimate all CPDs and that finishes our training. On this trained network, we use map query to predict label given the 2 images. We calculate the training and validation accuracy. I have not implemented this method here.

```
Seen Data
------------------------------------------------
training accuracy: 60.31392604248623
validation accuracy: 53.29139072847682
training time: 4.154043245633443 min
validation time: 54.366538472970326 min
------------------------------------------------


Shuffled Data
------------------------------------------------
training accuracy: 58.742897106390416
validation accuracy: 52.05077262693157
training time: 4.92474322890323443 min
validation time: 56.3667290364354332 min
------------------------------------------------


Unseen Data
------------------------------------------------
training accuracy: 55.7605559926567
validation accuracy: 48.441384736428
training time: 5.12073745330175 min
validation time: 59.7432259767760786 min
------------------------------------------------
```

3 Task 3 – Deep Learning

Here we build a Deep Learning model to determine if two images are from the same writer

or from a different writer. There are 2 different networks I have built here:

3.1 Method 1 - Siamese Networks

Let us first start by explaining by what a Siamese Network is. In a Siamese network, we pass the 2 images as inputs and get a similarity score as an output. Each image passed through a Deep Network, which typically consists of a chained up convolution layers, with a dense layer at the end. Both images pass through the same network. We then find the distance between 2 output vectors produced by the dense layer for the 2 images. This distance vector is then passed another Dense layer, with sigmoid actvation function producing one output in the range of 0 to 1. This is our final similarity score.

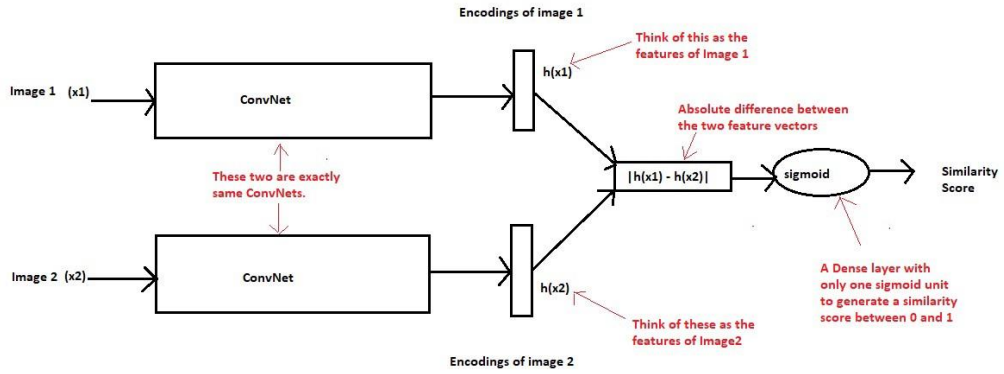
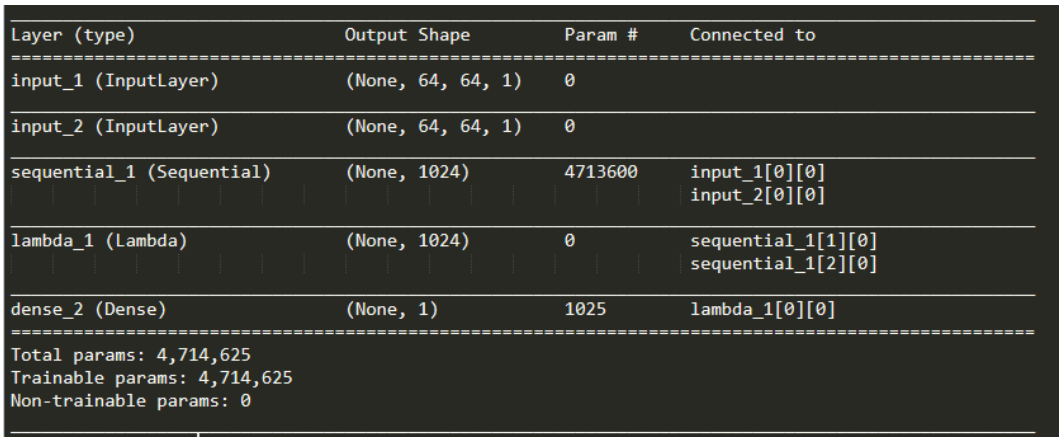


Fig 3.1.1: Siamese Network

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 60, 60, 64)	1664
max_pooling2d_1 (MaxPooling2	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 14, 14, 128)	0
conv2d_3 (Conv2D)	(None, 12, 12, 128)	147584
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 128)	0
conv2d_4 (Conv2D)	(None, 4, 4, 256)	295168
flatten_1 (Flatten)	(None, 4096)	0
dense_1 (Dense)	(None, 1024)	4195328

Total params: 4,713,600
Trainable params: 4,713,600
Non-trainable params: 0

Fig 3.1.2: The 1st part of the Siamese Network



```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_1 (InputLayer)         (None, 64, 64, 1)         0
__________________________________________________________________________________________________
input_2 (InputLayer)         (None, 64, 64, 1)         0
__________________________________________________________________________________________________
sequential_1 (Sequential)    (None, 1024)              4713600     input_1[0][0]
                                                                   input_2[0][0]
__________________________________________________________________________________________________
lambda_1 (Lambda)            (None, 1024)              0           sequential_1[1][0]
                                                                   sequential_1[2][0]
__________________________________________________________________________________________________
dense_2 (Dense)              (None, 1)                 1025        lambda_1[0][0]
==================================================================================================
Total params: 4,714,625
Trainable params: 4,714,625
Non-trainable params: 0
```

Fig 3.1.3: The 2nd part of the Siamese Network, which has Input 1 and Input 2 as output from the network above

3 Siamese networks were implemented here and trained on each of the 3 training data sets. We calculate and observe the different metrics produced during the training and testing(using validation data set) for these 3 networks.

On a total of 725 epochs run for the seen data, where each training dataset size is equal to the batch size, the best metric values we got were as follows:

Training Loss for the best validation loss: 0.62101

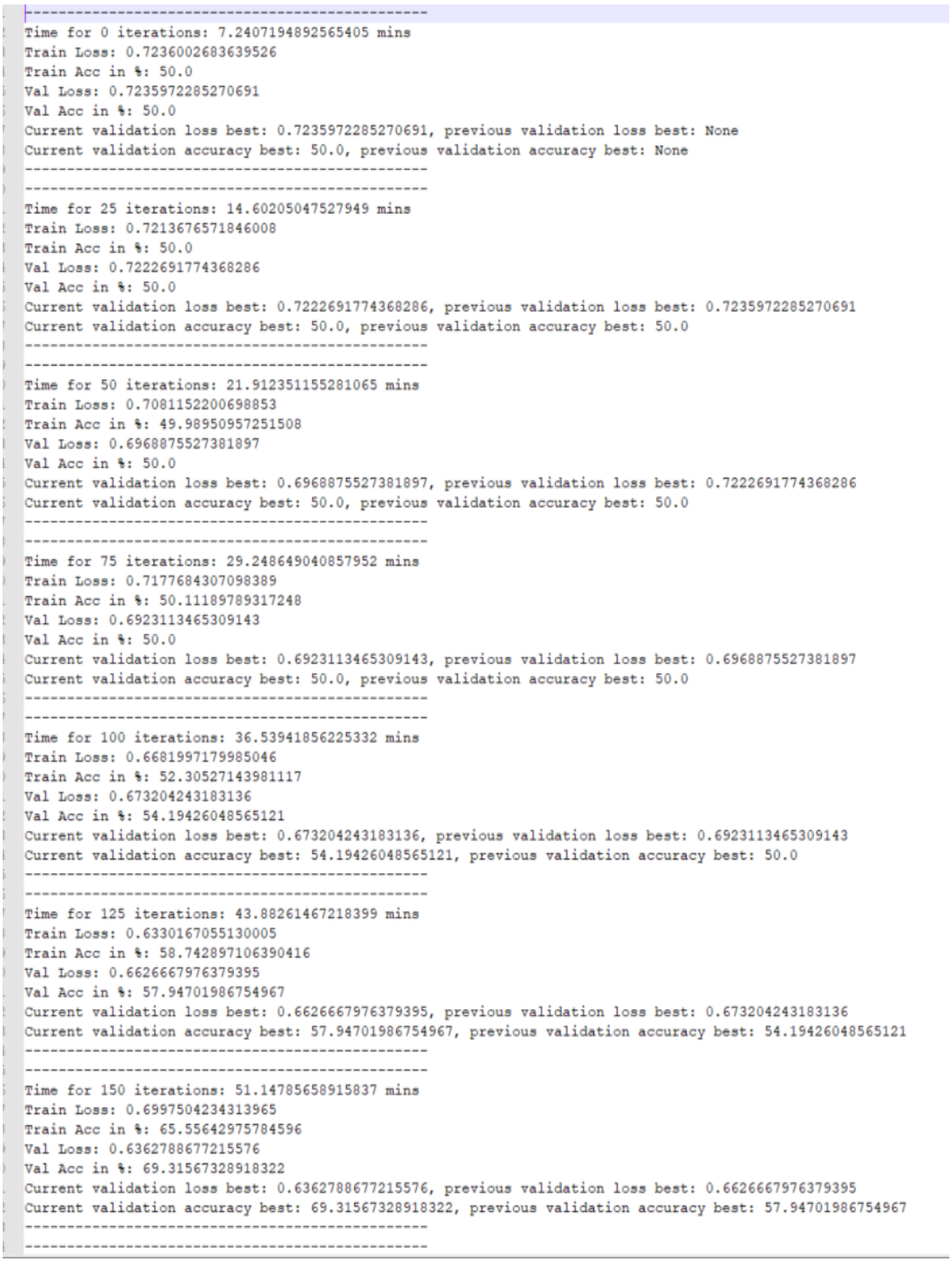
Training Accuracy for the best validation loss: 66.34%

Best Validation Loss: 0.63478

Best Validation Accuracy: 67.99%

Time Taken to reach this stage during Training: 72.7 min

Here are the console logs during training, which detail the different metrics for every 25 epochs:



```
------------------------------------------------
Time for 0 iterations: 7.2407194892565405 mins
Train Loss: 0.7236002683639526
Train Acc in %: 50.0
Val Loss: 0.7235972285270691
Val Acc in %: 50.0
Current validation loss best: 0.7235972285270691, previous validation loss best: None
Current validation accuracy best: 50.0, previous validation accuracy best: None
------------------------------------------------
------------------------------------------------
Time for 25 iterations: 14.60205047527949 mins
Train Loss: 0.7213676571846008
Train Acc in %: 50.0
Val Loss: 0.7222691774368286
Val Acc in %: 50.0
Current validation loss best: 0.7222691774368286, previous validation loss best: 0.7235972285270691
Current validation accuracy best: 50.0, previous validation accuracy best: 50.0
------------------------------------------------
------------------------------------------------
Time for 50 iterations: 21.912351155281065 mins
Train Loss: 0.7081152200698853
Train Acc in %: 49.98950957251508
Val Loss: 0.6968875527381897
Val Acc in %: 50.0
Current validation loss best: 0.6968875527381897, previous validation loss best: 0.7222691774368286
Current validation accuracy best: 50.0, previous validation accuracy best: 50.0
------------------------------------------------
------------------------------------------------
Time for 75 iterations: 29.248649040857952 mins
Train Loss: 0.7177684307098389
Train Acc in %: 50.11189789317248
Val Loss: 0.6923113465309143
Val Acc in %: 50.0
Current validation loss best: 0.6923113465309143, previous validation loss best: 0.6968875527381897
Current validation accuracy best: 50.0, previous validation accuracy best: 50.0
------------------------------------------------
------------------------------------------------
Time for 100 iterations: 36.53941856225332 mins
Train Loss: 0.6681997179985046
Train Acc in %: 52.30527143981117
Val Loss: 0.673204243183136
Val Acc in %: 54.19426048565121
Current validation loss best: 0.673204243183136, previous validation loss best: 0.6923113465309143
Current validation accuracy best: 54.19426048565121, previous validation accuracy best: 50.0
------------------------------------------------
------------------------------------------------
Time for 125 iterations: 43.88261467218399 mins
Train Loss: 0.6330167055130005
Train Acc in %: 58.742897106390416
Val Loss: 0.6626667976379395
Val Acc in %: 57.94701986754967
Current validation loss best: 0.6626667976379395, previous validation loss best: 0.673204243183136
Current validation accuracy best: 57.94701986754967, previous validation accuracy best: 54.19426048565121
------------------------------------------------
------------------------------------------------
Time for 150 iterations: 51.14785658915837 mins
Train Loss: 0.6997504234313965
Train Acc in %: 65.55642975784596
Val Loss: 0.6362788677215576
Val Acc in %: 69.31567328918322
Current validation loss best: 0.6362788677215576, previous validation loss best: 0.6626667976379395
Current validation accuracy best: 69.31567328918322, previous validation accuracy best: 57.94701986754967
------------------------------------------------
------------------------------------------------
```

Fig 2.2: Training log of the Seen Data Siamese Network Part 1 of 4

```
64  ------------------------------------------------
65  Time for 175 iterations: 58.44356293280919 mins
66  Train Loss: 0.6844720244407654
67  Train Acc in %: 60.23953142757234
68  Val Loss: 0.637333869934082
69  Val Acc in %: 67.54966887417218
70  ------------------------------------------------
71  ------------------------------------------------
72  Time for 200 iterations: 65.68188925186793 mins
73  Train Loss: 0.6357039213180542
74  Train Acc in %: 64.17868694815981
75  Val Loss: 0.6515439748764038
76  Val Acc in %: 64.79028697571744
77  ------------------------------------------------
78  ------------------------------------------------
79  Time for 225 iterations: 72.97666472991308 mins
80  Train Loss: 0.6210104823112488
81  Train Acc in %: 66.34321181921497
82  Val Loss: 0.6347841620445251
83  Val Acc in %: 67.99116997792494
84  Current validation loss best: 0.6347841620445251, previous validation loss best: 0.6362788677215576
85  Current validation accuracy best: 67.99116997792494, previous validation accuracy best: 69.31567328918322
86  ------------------------------------------------
87  ------------------------------------------------
88  Time for 250 iterations: 80.27571214437485 mins
89  Train Loss: 0.6759020090103149
90  Train Acc in %: 64.64026575749628
91  Val Loss: 0.6362654566764832
92  Val Acc in %: 65.45253863134658
93  ------------------------------------------------
94  ------------------------------------------------
95  Time for 275 iterations: 87.55951694250106 mins
96  Train Loss: 0.567981481552124
97  Train Acc in %: 69.28490252644463
98  Val Loss: 0.6482798457145691
99  Val Acc in %: 65.67328918322296
100 ------------------------------------------------
101 ------------------------------------------------
102 Time for 300 iterations: 94.83163629372915 mins
103 Train Loss: 0.5921710729598999
104 Train Acc in %: 69.63895445406067
105 Val Loss: 0.6531269550323486
106 Val Acc in %: 63.90728476821192
107 ------------------------------------------------
108 ------------------------------------------------
109 Time for 325 iterations: 102.11166784763336 mins
110 Train Loss: 0.4360547959804535
111 Train Acc in %: 67.93163738088994
112 Val Loss: 0.6682141423225403
113 Val Acc in %: 61.58940397350993
114 ------------------------------------------------
115 ------------------------------------------------
116 Time for 350 iterations: 109.41746389468511 mins
117 Train Loss: 0.41407638788223267
118 Train Acc in %: 71.25273188215753
119 Val Loss: 0.6699085235595703
120 Val Acc in %: 59.05077262693157
121 ------------------------------------------------
122 ------------------------------------------------
123 Time for 375 iterations: 116.68054074048996 mins
124 Train Loss: 0.4474616050720215
125 Train Acc in %: 70.99134539732493
126 Val Loss: 0.6832395792007446
127 Val Acc in %: 58.27814569536424
```

Fig 2.3: Training log of the Seen Data Siamese Network Part 2 of 4

```
-----------------------------------------------
-----------------------------------------------
Time for 400 iterations: 123.9464932401975 mins
Train Loss: 0.4551668167114258
Train Acc in %: 74.83084185680566
Val Loss: 0.6699594259262085
Val Acc in %: 60.264900662251655
-----------------------------------------------
-----------------------------------------------
Time for 425 iterations: 131.2167509039243 mins
Train Loss: 0.5800811648368835
Train Acc in %: 70.02972287787394
Val Loss: 0.6943979263305664
Val Acc in %: 57.94701986754967
-----------------------------------------------
-----------------------------------------------
Time for 450 iterations: 138.64575871626536 mins
Train Loss: 0.4314533770084381
Train Acc in %: 77.36165748754262
Val Loss: 0.6757692098617554
Val Acc in %: 57.28476821192053
-----------------------------------------------
-----------------------------------------------
Time for 475 iterations: 145.96187717517216 mins
Train Loss: 0.3136719763278961
Train Acc in %: 77.51988810210683
Val Loss: 0.6887243390083313
Val Acc in %: 57.17439293598234
-----------------------------------------------
-----------------------------------------------
Time for 500 iterations: 153.23256701628367 mins
Train Loss: 0.44717344641685486
Train Acc in %: 78.40195821313051
Val Loss: 0.673672080039978
Val Acc in %: 59.93377483443709
-----------------------------------------------
-----------------------------------------------
Time for 525 iterations: 160.52341966231663 mins
Train Loss: 0.2874472439289093
Train Acc in %: 83.00463327213917
Val Loss: 0.6963350176811218
Val Acc in %: 59.492273730684325
-----------------------------------------------
-----------------------------------------------
Time for 550 iterations: 167.8474503159523 mins
Train Loss: 0.31847524642944336
Train Acc in %: 82.77559227205175
Val Loss: 0.7010727524757385
Val Acc in %: 58.498896247240616
-----------------------------------------------
-----------------------------------------------
Time for 575 iterations: 175.11895464658738 mins
Train Loss: 0.21756882965564728
Train Acc in %: 83.39365329137162
Val Loss: 0.6933422088623047
Val Acc in %: 60.154525386313466
-----------------------------------------------
-----------------------------------------------
Time for 600 iterations: 182.42657912572224 mins
Train Loss: 0.3676767349243164
Train Acc in %: 83.55363231051665
Val Loss: 0.7170710563659668
Val Acc in %: 58.05739514348786
-----------------------------------------------
```

Fig 2.4: Training log of the Seen Data Siamese Network Part 3 of 4

```
------------------------------------------------
Time for 625 iterations: 189.6938447992007 mins
Train Loss: 0.2660367488861084
Train Acc in %: 86.06608969315499
Val Loss: 0.7315398454666138
Val Acc in %: 57.94701986754967
------------------------------------------------
------------------------------------------------
Time for 650 iterations: 196.99144606192905 mins
Train Loss: 0.23682314157485962
Train Acc in %: 87.1326164874552
Val Loss: 0.7478304505348206
Val Acc in %: 56.73289183222958
------------------------------------------------
------------------------------------------------
Time for 675 iterations: 204.24468253850938 mins
Train Loss: 0.140179842710495
Train Acc in %: 90.72733630562112
Val Loss: 0.820590615272522
Val Acc in %: 50.99337748344371
------------------------------------------------
------------------------------------------------
Time for 700 iterations: 211.55308827559153 mins
Train Loss: 0.2864168882369995
Train Acc in %: 91.29207098522598
Val Loss: 0.7956681251525879
Val Acc in %: 50.66225165562914
------------------------------------------------
------------------------------------------------
Time for 725 iterations: 218.81584930419922 mins
Train Loss: 0.13054966926574707
Train Acc in %: 91.16793425998776
Val Loss: 0.834297239780426
Val Acc in %: 49.66887417218543
------------------------------------------------
```

Fig 2.5: Training log of the Seen Data Siamese Network Part 3 of 4

3.2 Method 2 - Autoencoder Networks

Let us first start by explaining by what an Autoencoder Network is. The aim of the Autoencoder network is to create a encoded feature vector with an image as an input. This vector, essentially, captures all the significant information about the image and can be used to represent the image. This encoded vector is of a smaller size than the original image, so this vector can then be used by other networks, instead of them directly working on an image. This leads to fewer parameters and is very advantageous. Autoencoder training is an unsupervised task. While training, we take the image as input and create the encoded feature vector. This is the Encoder part of the network. We then take this encoded vector and recreate the image. This is the decoded part. Thus for the whole network, input and output are both the image itself. Once the training is complete, we don't care about decoding and only use the encoded feature vector.

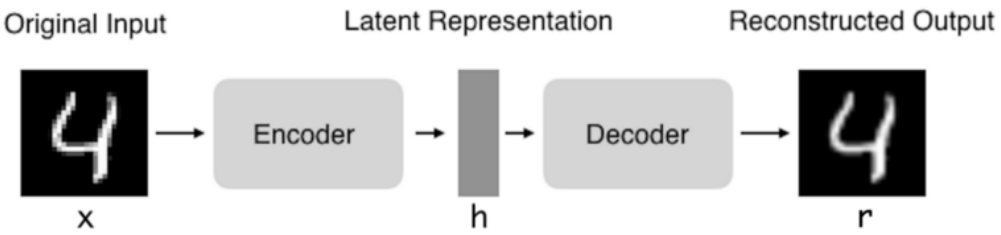


Fig 3.2.1: Autoencoder

AUTOENCODER NETWORKS:
Seen Data Set:

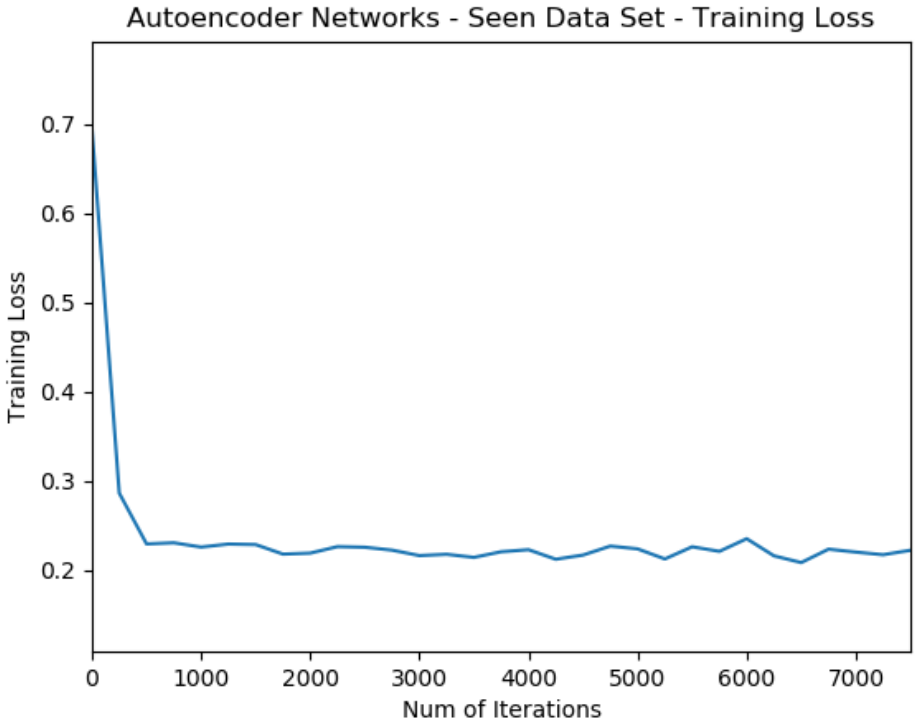
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 64, 64, 1)	0
conv2d_1 (Conv2D)	(None, 64, 64, 16)	160
max_pooling2d_1 (MaxPooling2	(None, 32, 32, 16)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	4640
max_pooling2d_2 (MaxPooling2	(None, 16, 16, 32)	0
conv2d_3 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_3 (MaxPooling2	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73856
max_pooling2d_4 (MaxPooling2	(None, 4, 4, 128)	0
conv2d_5 (Conv2D)	(None, 4, 4, 256)	295168
max_pooling2d_5 (MaxPooling2	(None, 2, 2, 256)	0
conv2d_6 (Conv2D)	(None, 2, 2, 512)	1180160
encoded_feature_vec (MaxPool	(None, 1, 1, 512)	0
conv2d_7 (Conv2D)	(None, 1, 1, 512)	2359808
up_sampling2d_1 (UpSampling2	(None, 2, 2, 512)	0
conv2d_8 (Conv2D)	(None, 2, 2, 256)	1179904
up_sampling2d_2 (UpSampling2	(None, 4, 4, 256)	0
conv2d_9 (Conv2D)	(None, 4, 4, 128)	295040
up_sampling2d_3 (UpSampling2	(None, 8, 8, 128)	0
conv2d_10 (Conv2D)	(None, 8, 8, 64)	73792
up_sampling2d_4 (UpSampling2	(None, 16, 16, 64)	0
conv2d_11 (Conv2D)	(None, 16, 16, 32)	18464
up_sampling2d_5 (UpSampling2	(None, 32, 32, 32)	0
conv2d_12 (Conv2D)	(None, 32, 32, 16)	4624
up_sampling2d_6 (UpSampling2	(None, 64, 64, 16)	0
output (Conv2D)	(None, 64, 64, 1)	145

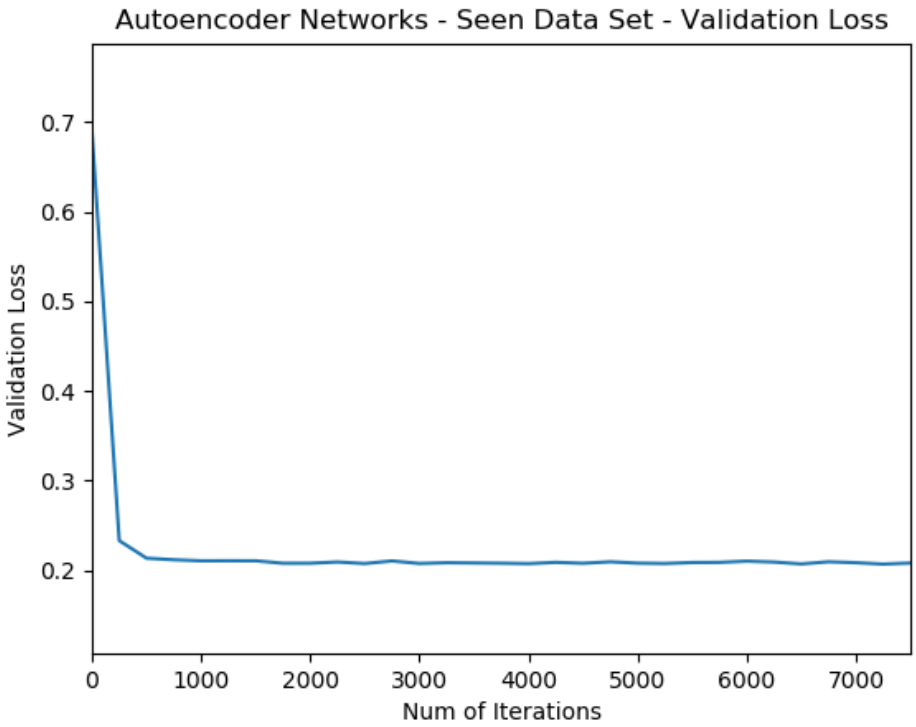
Total params: 5,504,257
Trainable params: 5,504,257
Non-trainable params: 0

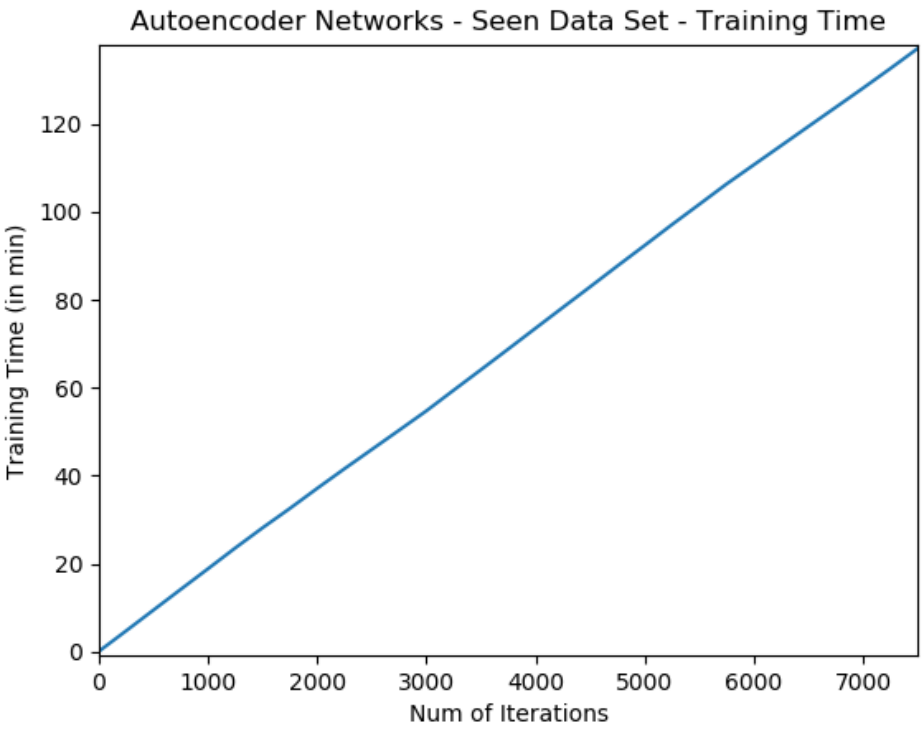
Fig 3.2.2: The Autoencoder network Created for our project

Once we have trained the autoencoder, we take the 2 images and find their encoded feature vectors. We can now proceed 2 2o calculate the similarity using 2 approaches. We can either just pass these 2 vectors as input to a new Deep learning model and train it to predict the similarity. OR we can just take the cosine similarity of these 2 vectors to predict whether the 2 images are from the same writer or not. Here, we do the latter. We calculate and observe the different metrics produced during the training and testing (using validation data set) for

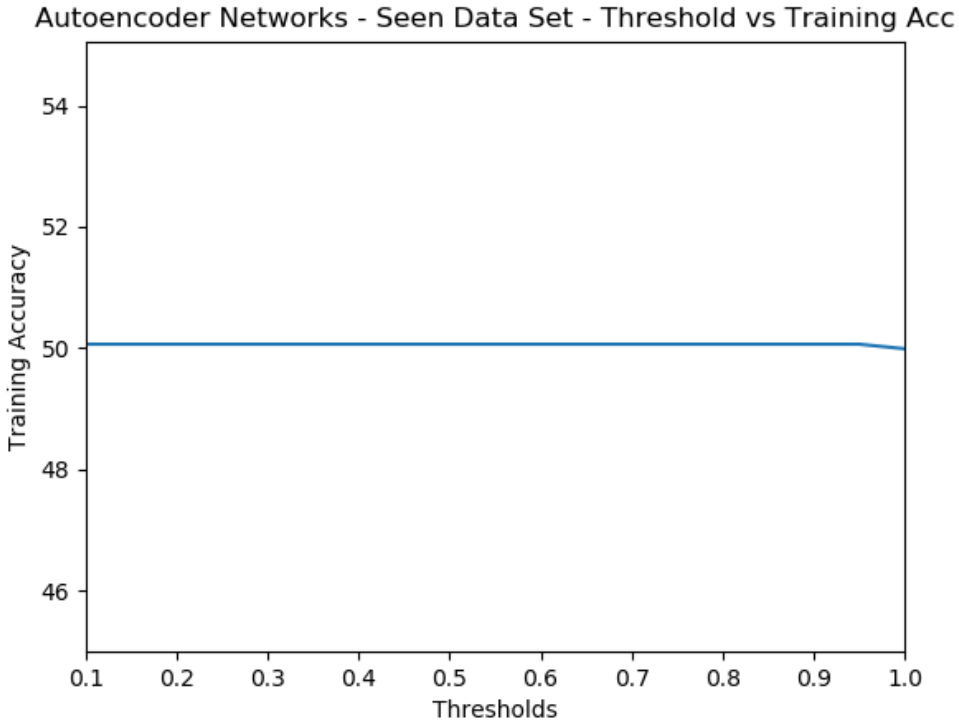
these 3 networks. The results are attached here:



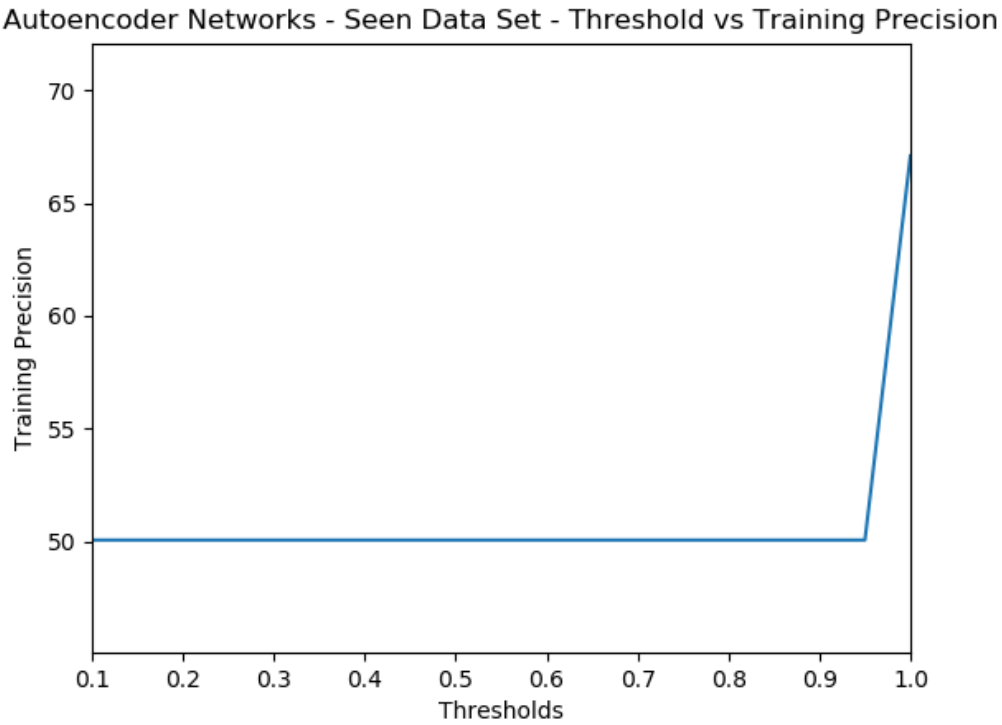



Autoencoder Networks - Seen Data Set - Training Time
Training Time (in min)
0
20
40
60
80
100
120
0
1000
2000
3000
4000
5000
6000
7000
Num of Iterations

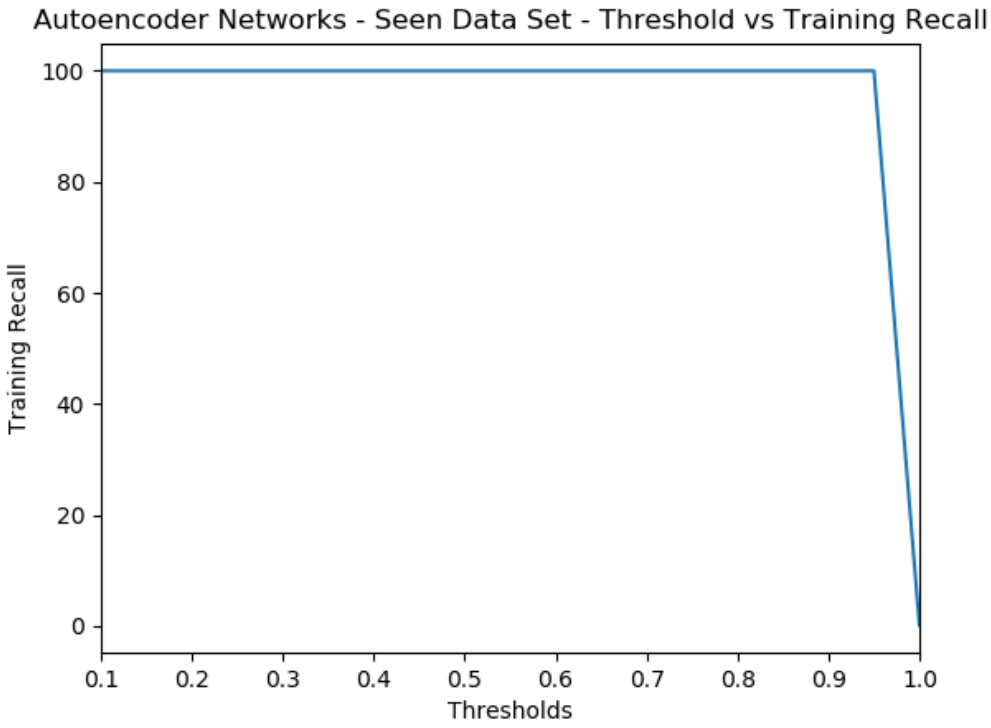
121


Autoencoder Networks - Seen Data Set - Threshold vs Training Acc
Training Accuracy
46
48
50
52
54
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

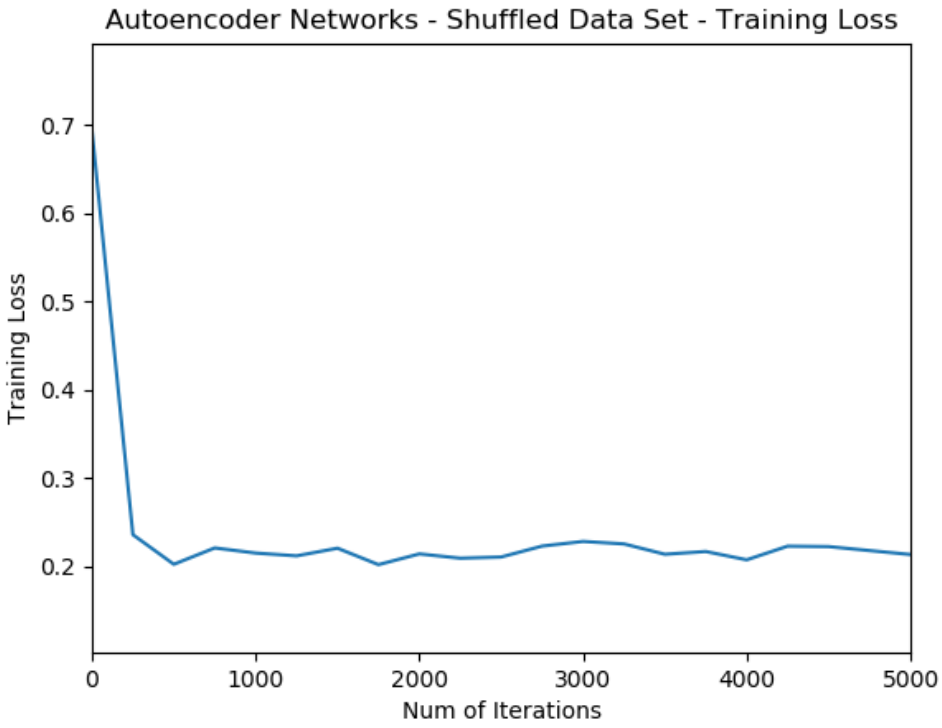
122


Autoencoder Networks - Seen Data Set - Threshold vs Training Precision
Training Precision
70
65
60
55
50
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

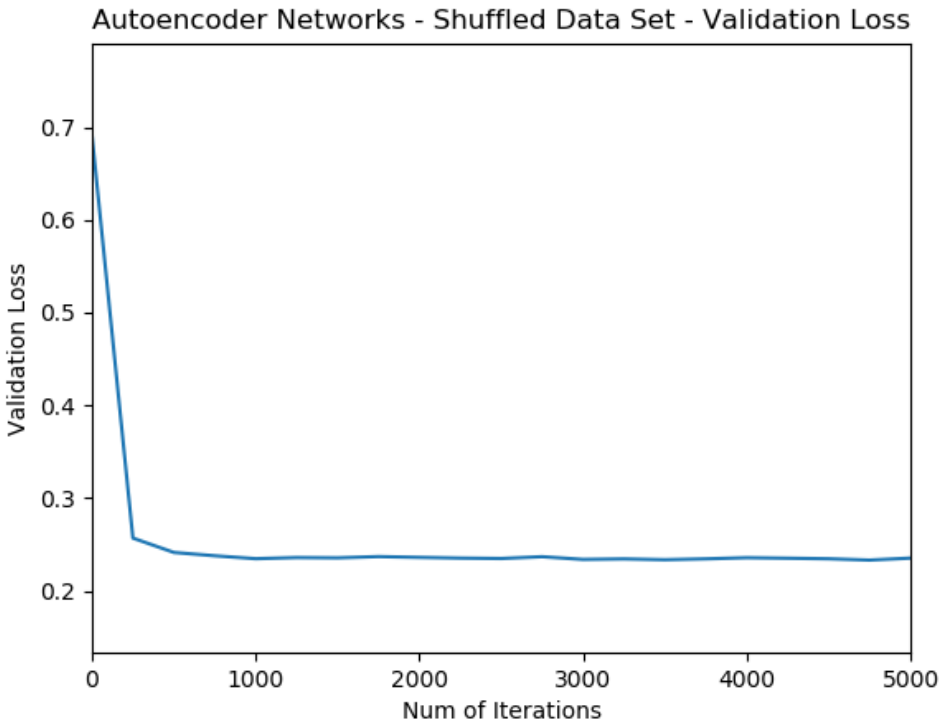
123


Autoencoder Networks - Seen Data Set - Threshold vs Training Recall
Training Recall
100
80
60
40
20
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

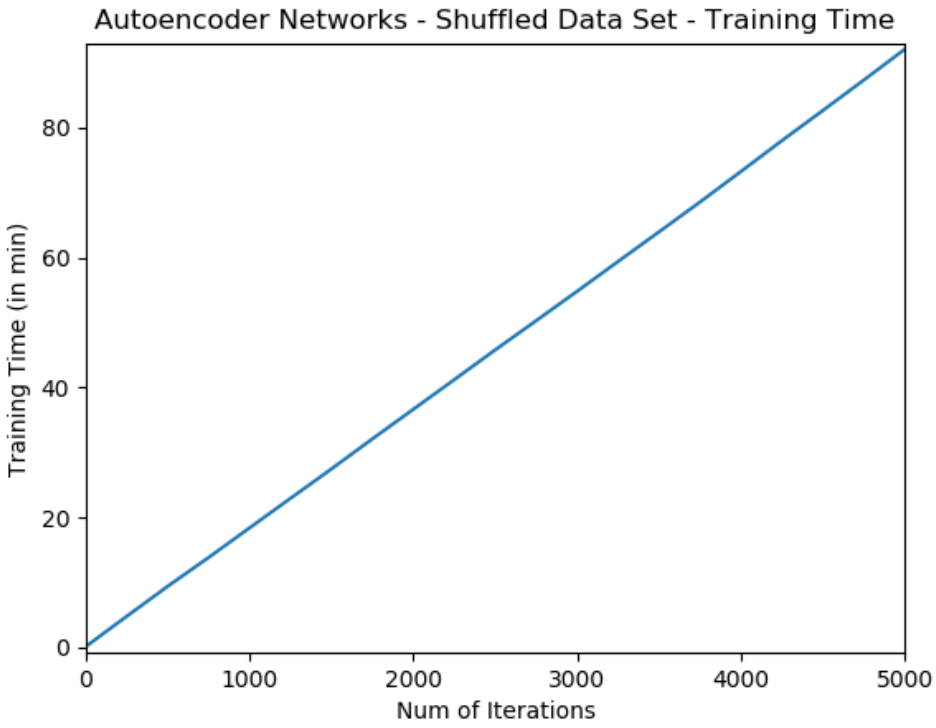
124


Autoencoder Networks - Shuffled Data Set - Training Loss
Training Loss
0.7
0.6
0.5
0.4
0.3
0.2
0
1000
2000
3000
4000
5000
Num of Iterations

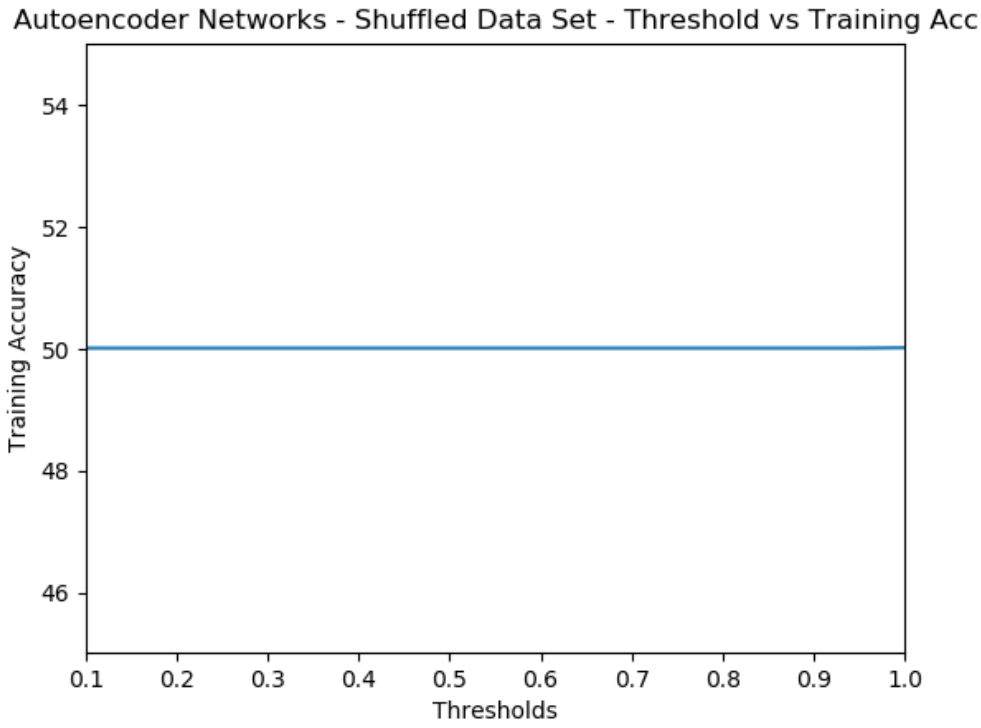
125


Autoencoder Networks - Shuffled Data Set - Validation Loss
Validation Loss
0.7
0.6
0.5
0.4
0.3
0.2
0
1000
2000
3000
4000
5000
Num of Iterations

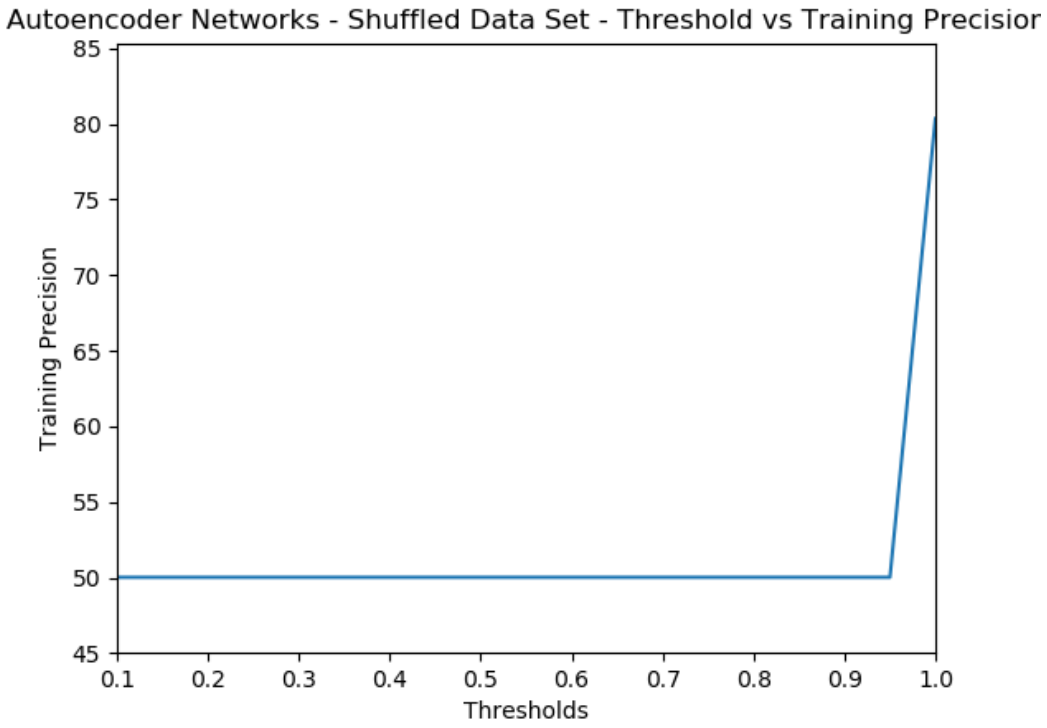
126


Autoencoder Networks - Shuffled Data Set - Training Time
Training Time (in min)
0
20
40
60
80
0
1000
2000
3000
4000
5000
Num of Iterations

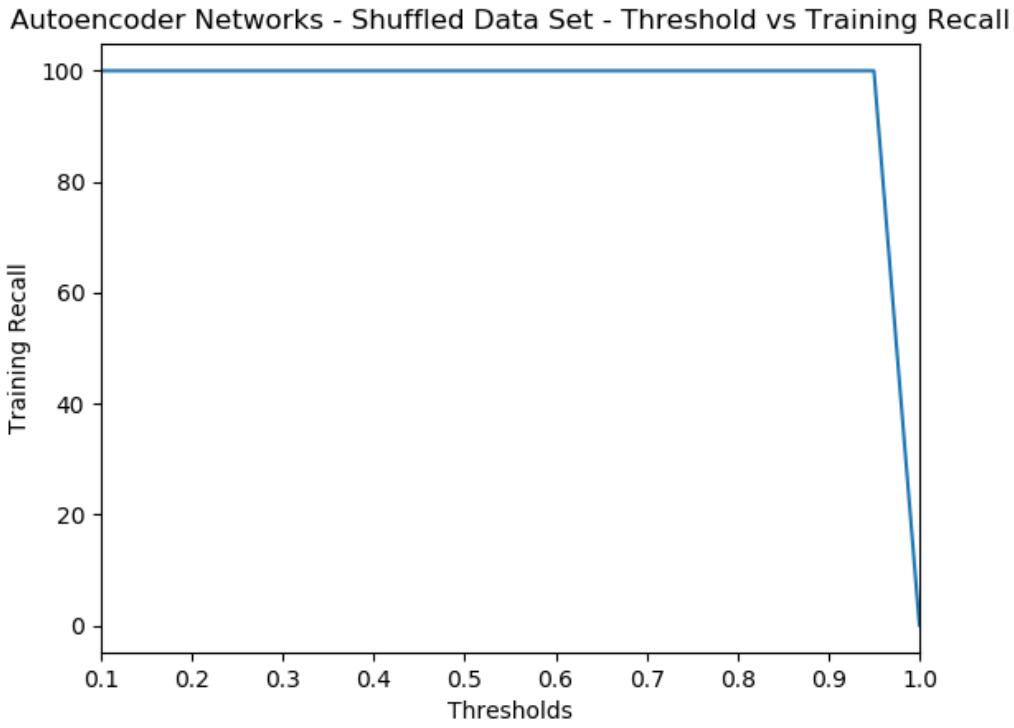
127


Autoencoder Networks - Shuffled Data Set - Threshold vs Training Acc
Training Accuracy
46
48
50
52
54
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

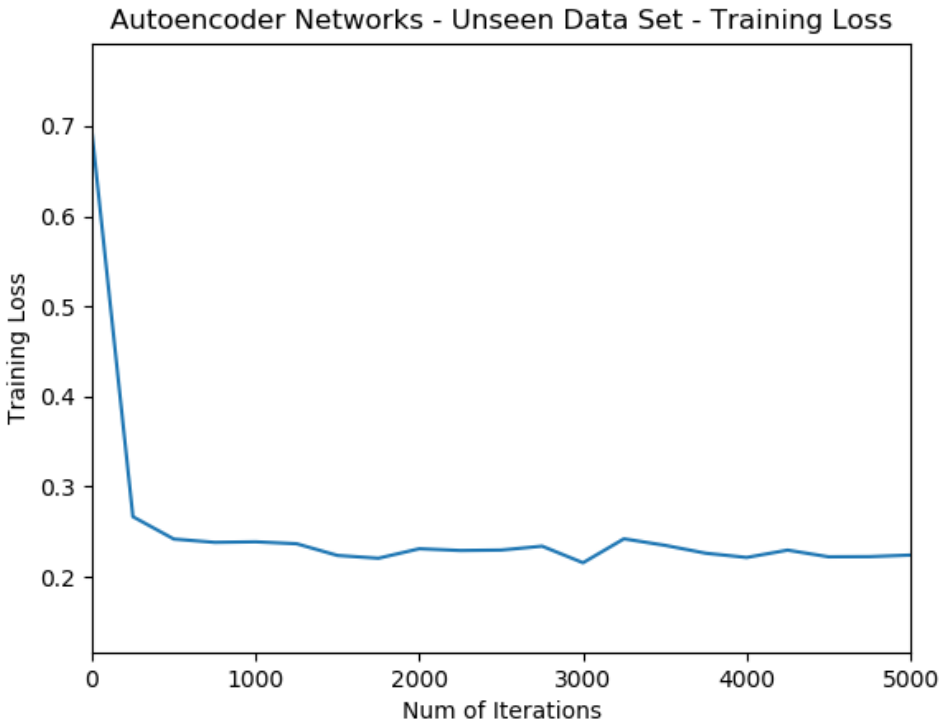
128


Autoencoder Networks - Shuffled Data Set - Threshold vs Training Precisior
Training Precision
85
80
75
70
65
60
55
50
45
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

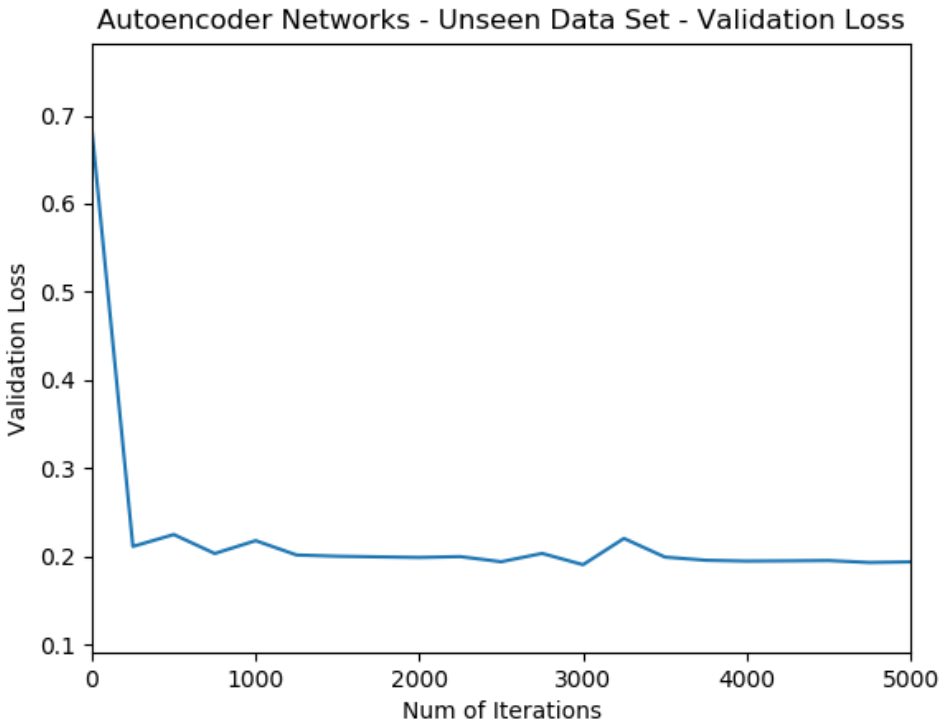
129


Autoencoder Networks - Shuffled Data Set - Threshold vs Training Recall
Training Recall
100
80
60
40
20
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

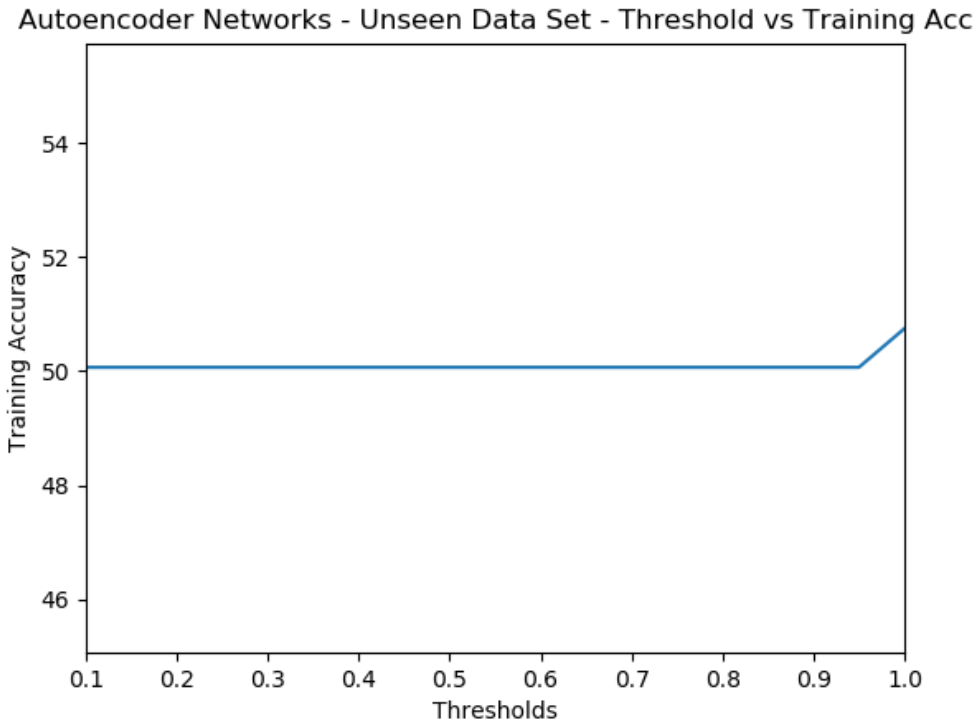
130


Autoencoder Networks - Unseen Data Set - Training Loss
Training Loss
0.7
0.6
0.5
0.4
0.3
0.2
0
1000
2000
3000
4000
5000
Num of Iterations

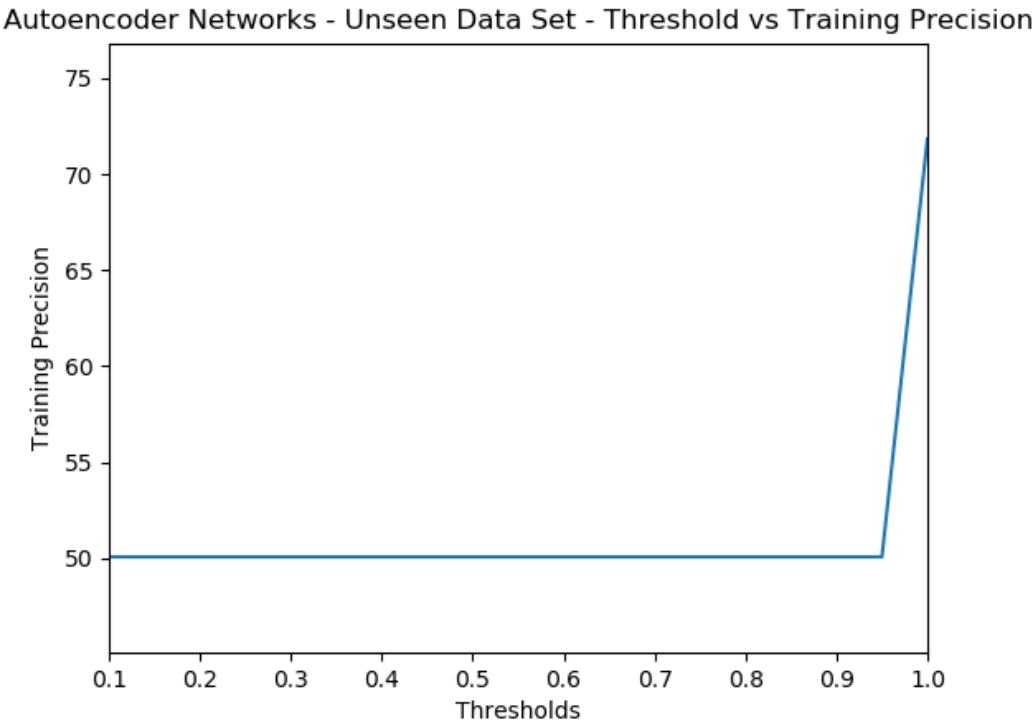
131


Autoencoder Networks - Unseen Data Set - Validation Loss
Validation Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1000
2000
3000
4000
5000
Num of Iterations

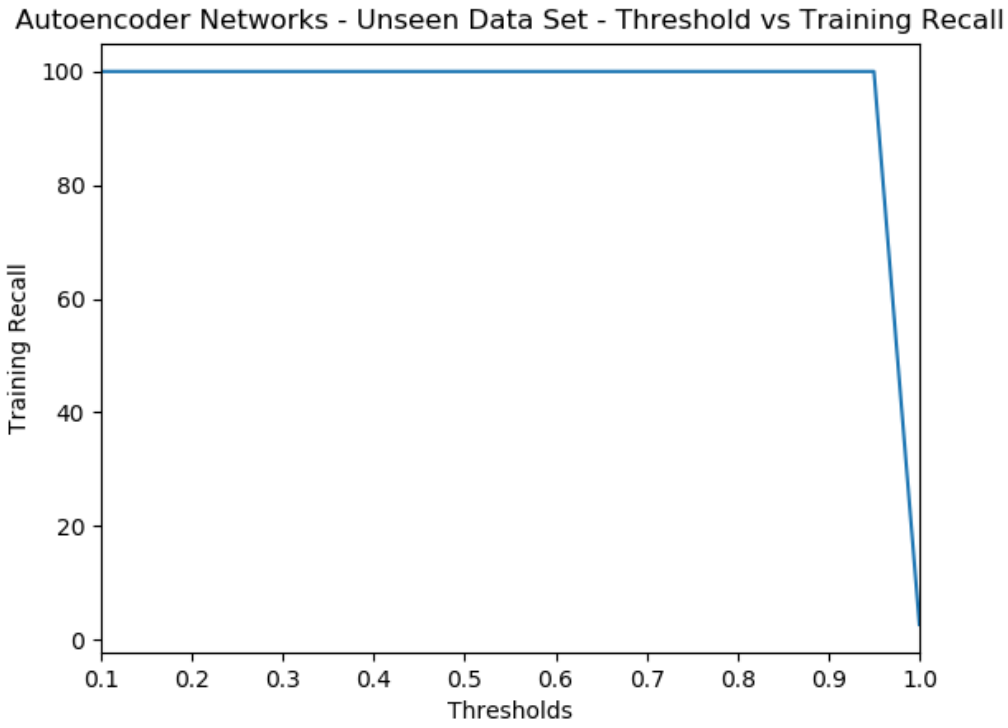
132

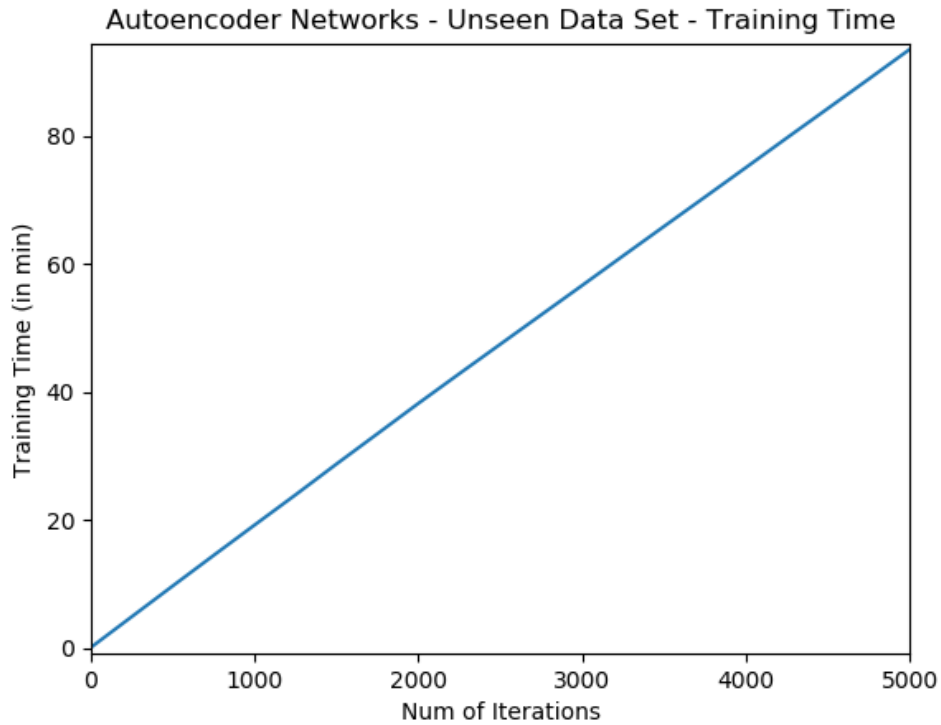

Autoencoder Networks - Unseen Data Set - Threshold vs Training Acc
Training Accuracy
54
52
50
48
46
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

133


Autoencoder Networks - Unseen Data Set - Threshold vs Training Precision
Training Precision
75
70
65
60
55
50
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

134





Here are the final Best Values obtained:

```
Seen Data:
-------------------------------------------------
Model Training Complete. Model Saved.
Best validation loss: 0.20694151520729065
Best validation loss iter number: 7250
For similarity detection:
Best training accuracy: 50.06126914660831
Best training precision: 50.06126914660831
Best training recall: 100.0
Best validation accuracy: 60.4
Best validation precision: 60.4
Best validation recall: 100.0
Best cosine threshold: 0.1
-------------------------------------------------
```

```
Shuffed Datset:
-------------------------------------------------
Model Training Complete. Model Saved.
Best validation loss: 0.23326733708381653
Best validation loss iter number: 4750
For similarity detection:
Best training accuracy: 50.01955990220049
Best training precision: 80.35714285714286
Best training recall: 0.08799546334500087
Best validation accuracy: 49.56190476190476
Best validation precision: 33.33333333333333
Best validation recall: 0.15128593040847202
Best cosine threshold: 1.0
-------------------------------------------------
```

```
Unseen Datset
-------------------------------------------------
Model Training Complete. Model Saved.
Best validation loss: 0.19063013792037964
Best validation loss iter number: 3000
For similarity detection:
Best training accuracy: 50.745279383429676
Best training precision: 71.83098591549296
Best training recall: 2.6692527939410735
Best validation accuracy: 49.72
Best validation precision: 54.460093896713616
Best validation recall: 3.0606860158311346
Best cosine threshold: 1.0
-------------------------------------------------
```

4 Task 4 – Explainable AI: Multitask Trainable Encoder

In this task we try to create an Explainable AI model. An explainable AI model, as the name says aims to build an AI model that can explain its decisions. We need out model to justify

its decisions. If it says that 2 images are from the same writer, then we need it to tell us why it said so, which features influenced our decisions, etc.

In this task, since we need explainability, we try to create the feature vector given an image. This feature vector resembles the human observed feature vector and hence, corresponds to qualities that a human can understand. Thus, we take an image as an input and then calculate the encoded feature vector using the encoder we trained in Task3. This encoded feature vector is then passed to a new Deep Network layer, consisting of Dense nodes, to predict the human observed features. Thus, the output is an array of size 15 (as we give rise to 15 parallel networks from the encoded feature vector – one for each feature), where each element is an array of size equal to the number of distinct values of that feature i.e., each feature vector is the probability of each possible value for that feature.

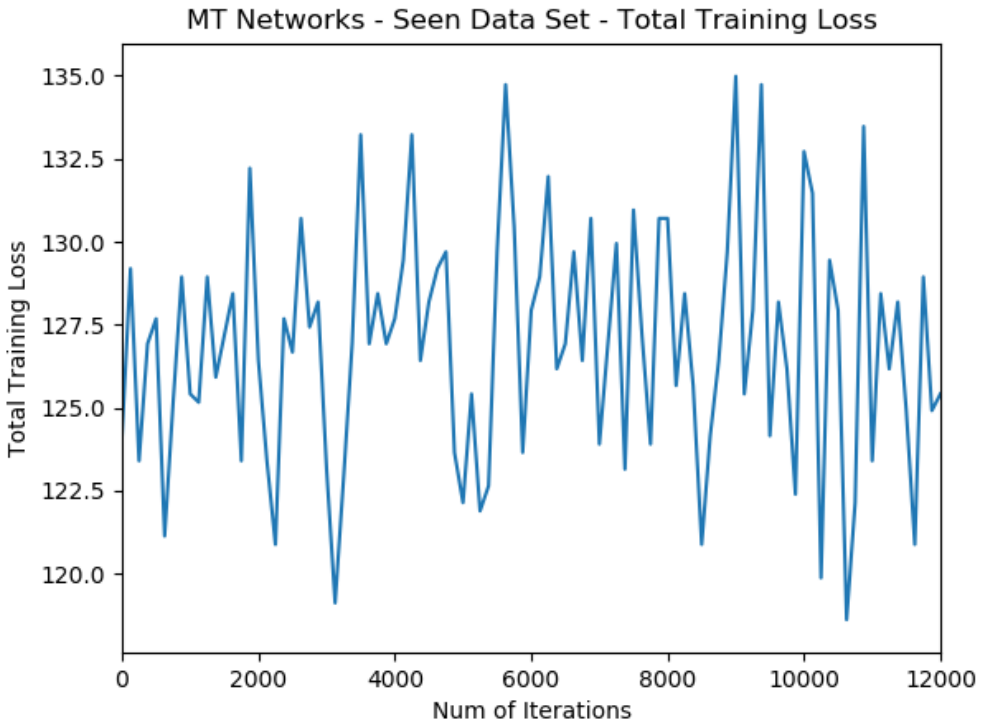
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 64, 64, 1)	0	
conv2d_1 (Conv2D)	(None, 64, 64, 16)	160	input_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 16)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 32, 32, 32)	4640	max_pooling2d_1[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 32)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 16, 16, 64)	18496	max_pooling2d_2[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 64)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73856	max_pooling2d_3[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 128)	0	conv2d_4[0][0]
conv2d_5 (Conv2D)	(None, 4, 4, 256)	295168	max_pooling2d_4[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 2, 2, 256)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 2, 2, 512)	1180160	max_pooling2d_5[0][0]
encoded_feature_vec (MaxPooling	(None, 1, 1, 512)	0	conv2d_6[0][0]
flatten_1 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_2 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_3 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_4 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_5 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_6 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_7 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_8 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_9 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_10 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_11 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_12 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_13 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_14 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
flatten_15 (Flatten)	(None, 512)	0	encoded_feature_vec[0][0]
dense_layer_1 (Dense)	(None, 256)	131328	flatten_1[0][0]

Fig 4.1: Explainable AI model we have built, part 1 of 2

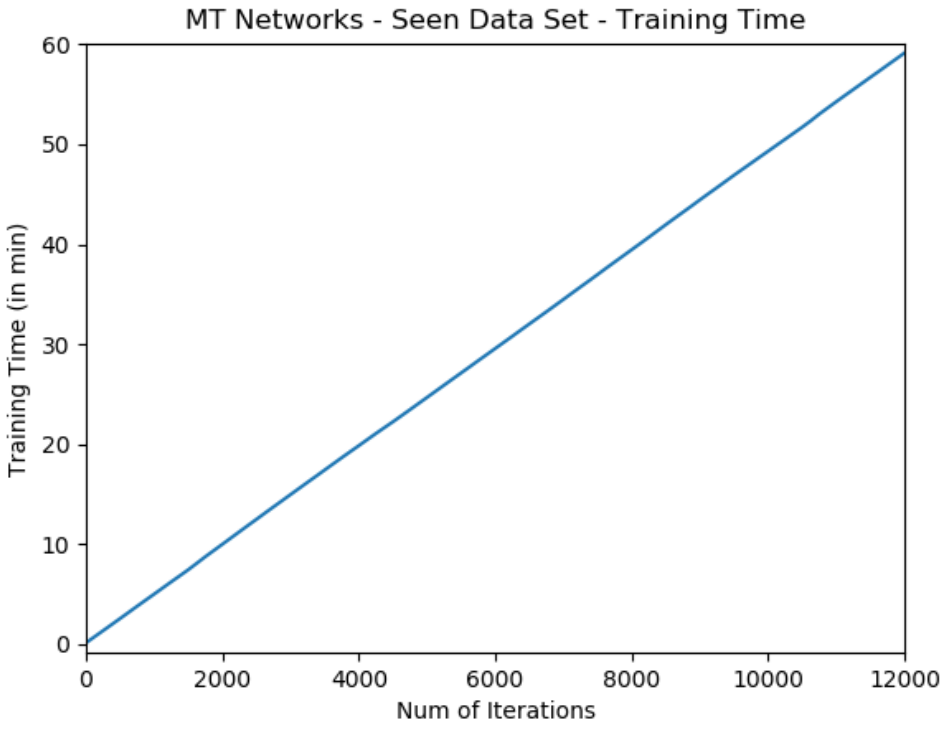
```
dense_layer_1 (Dense)          (None, 256)          131328      flatten_1[0][0]
dense_layer_2 (Dense)          (None, 256)          131328      flatten_2[0][0]
dense_layer_3 (Dense)          (None, 256)          131328      flatten_3[0][0]
dense_layer_4 (Dense)          (None, 256)          131328      flatten_4[0][0]
dense_layer_5 (Dense)          (None, 256)          131328      flatten_5[0][0]
dense_layer_6 (Dense)          (None, 256)          131328      flatten_6[0][0]
dense_layer_7 (Dense)          (None, 256)          131328      flatten_7[0][0]
dense_layer_8 (Dense)          (None, 256)          131328      flatten_8[0][0]
dense_layer_9 (Dense)          (None, 256)          131328      flatten_9[0][0]
dense_layer_10 (Dense)         (None, 256)          131328      flatten_10[0][0]
dense_layer_11 (Dense)         (None, 256)          131328      flatten_11[0][0]
dense_layer_12 (Dense)         (None, 256)          131328      flatten_12[0][0]
dense_layer_13 (Dense)         (None, 256)          131328      flatten_13[0][0]
dense_layer_14 (Dense)         (None, 256)          131328      flatten_14[0][0]
dense_layer_15 (Dense)         (None, 256)          131328      flatten_15[0][0]
out_feature_1 (Dense)          (None, 2)            514         dense_layer_1[0][0]
out_feature_2 (Dense)          (None, 3)            771         dense_layer_2[0][0]
out_feature_3 (Dense)          (None, 3)            771         dense_layer_3[0][0]
out_feature_4 (Dense)          (None, 3)            771         dense_layer_4[0][0]
out_feature_5 (Dense)          (None, 2)            514         dense_layer_5[0][0]
out_feature_6 (Dense)          (None, 2)            514         dense_layer_6[0][0]
out_feature_7 (Dense)          (None, 4)            1028        dense_layer_7[0][0]
out_feature_8 (Dense)          (None, 2)            514         dense_layer_8[0][0]
out_feature_9 (Dense)          (None, 2)            514         dense_layer_9[0][0]
out_feature_10 (Dense)         (None, 4)            1028        dense_layer_10[0][0]
out_feature_11 (Dense)         (None, 2)            514         dense_layer_11[0][0]
out_feature_12 (Dense)         (None, 3)            771         dense_layer_12[0][0]
out_feature_13 (Dense)         (None, 4)            1028        dense_layer_13[0][0]
out_feature_14 (Dense)         (None, 2)            514         dense_layer_14[0][0]
out_feature_15 (Dense)         (None, 2)            514         dense_layer_15[0][0]
==================================================================================================
Total params: 3,552,680
Trainable params: 3,552,680
```

Fig 4.1: Explainable AI model we have built, part 2 of 2

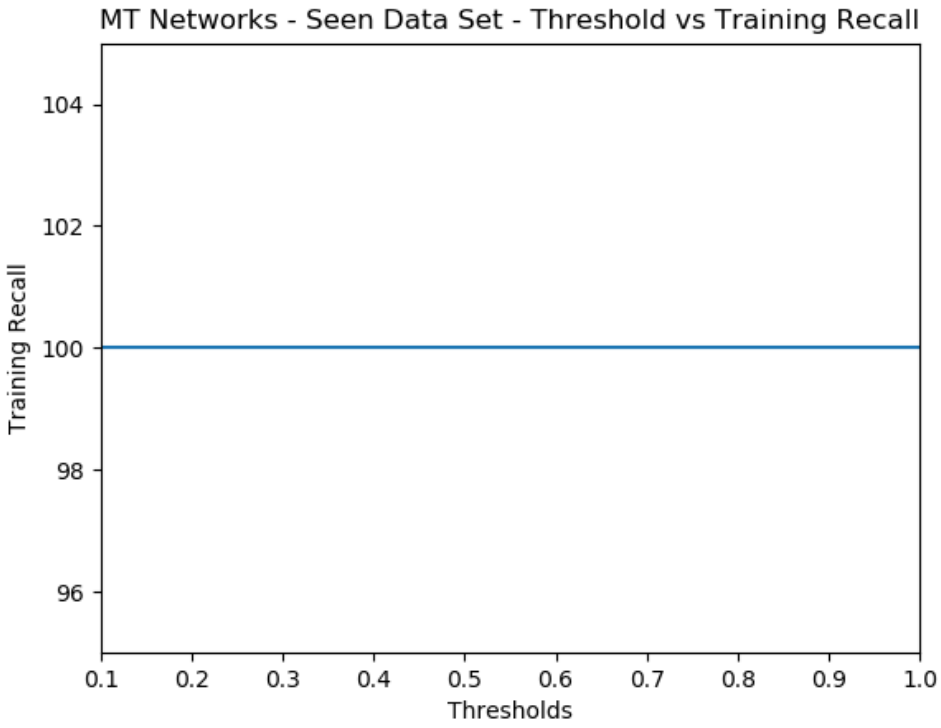
We take these human observed features for 2 images,and find their cosine similarity. We use training data to find the best cosine threshold for these scores. Best cosine threshold finds a high accuracy value but also tries to keep the precision and recall value as close to each other as possible. We calculate and observe the different metrics produced during the training and testing (using validation data set) for these 3 networks. The results are attached here:


MT Networks - Seen Data Set - Total Training Loss
Total Training Loss
135.0
132.5
130.0
127.5
125.0
122.5
120.0
0
2000
4000
6000
8000
10000
12000
Num of Iterations

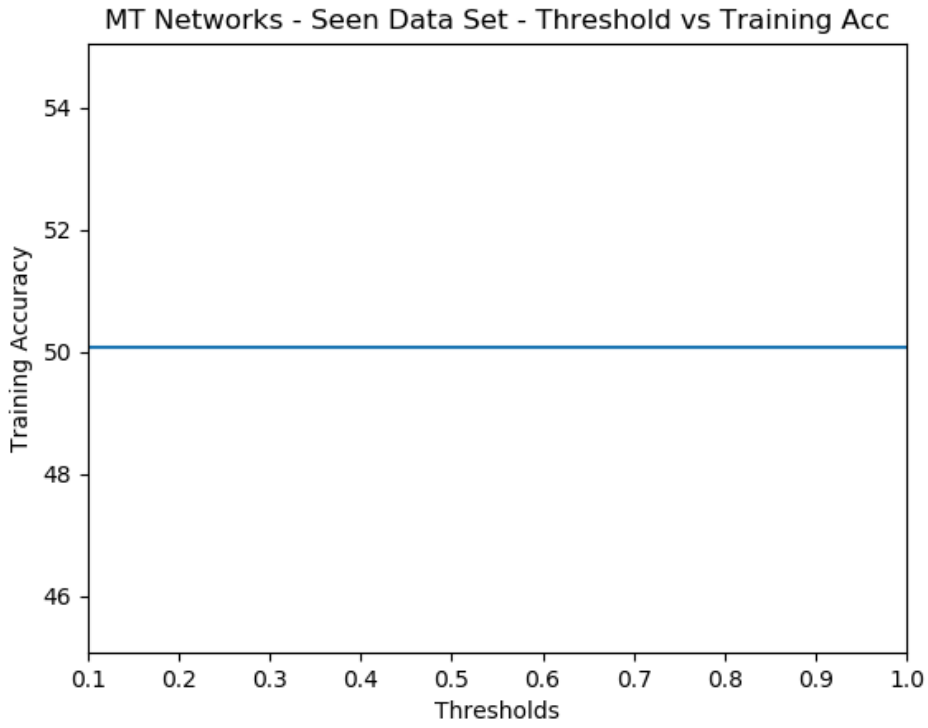
169


MT Networks - Seen Data Set - Training Time
Training Time (in min)
60
50
40
30
20
10
0
0
2000
4000
6000
8000
10000
12000
Num of Iterations

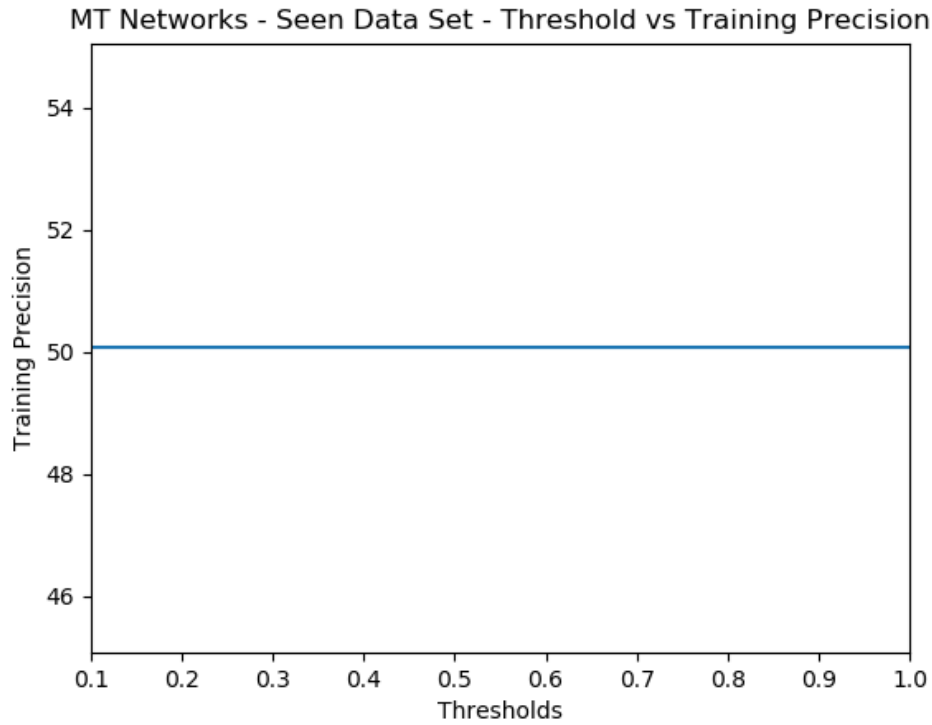
170


MT Networks - Seen Data Set - Threshold vs Training Recall
Training Recall
104
102
100
98
96
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

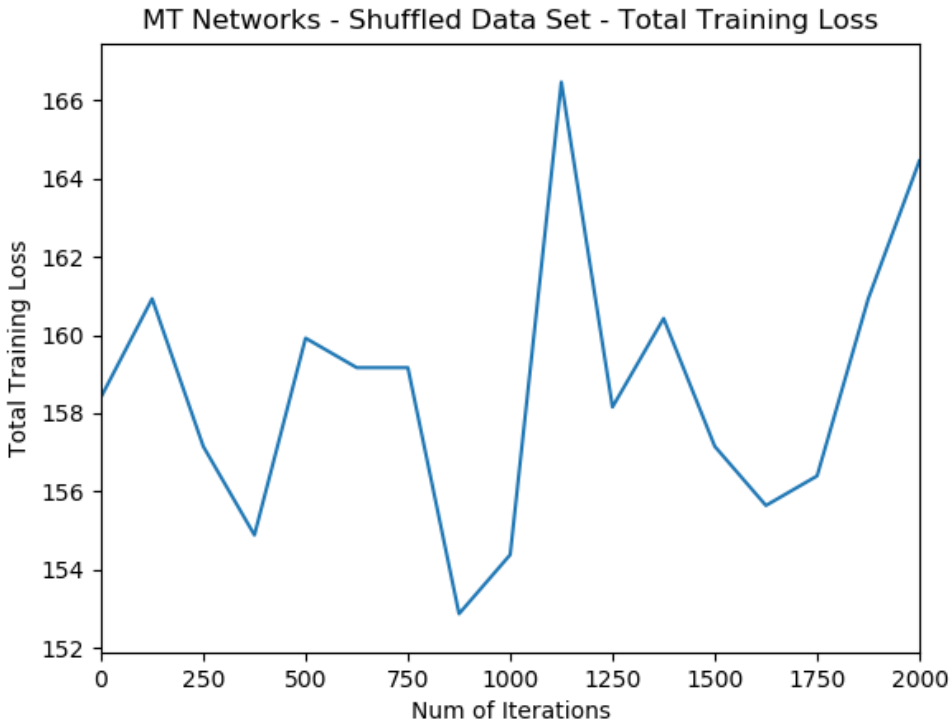
171


MT Networks - Seen Data Set - Threshold vs Training Acc
Training Accuracy
54
52
50
48
46
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

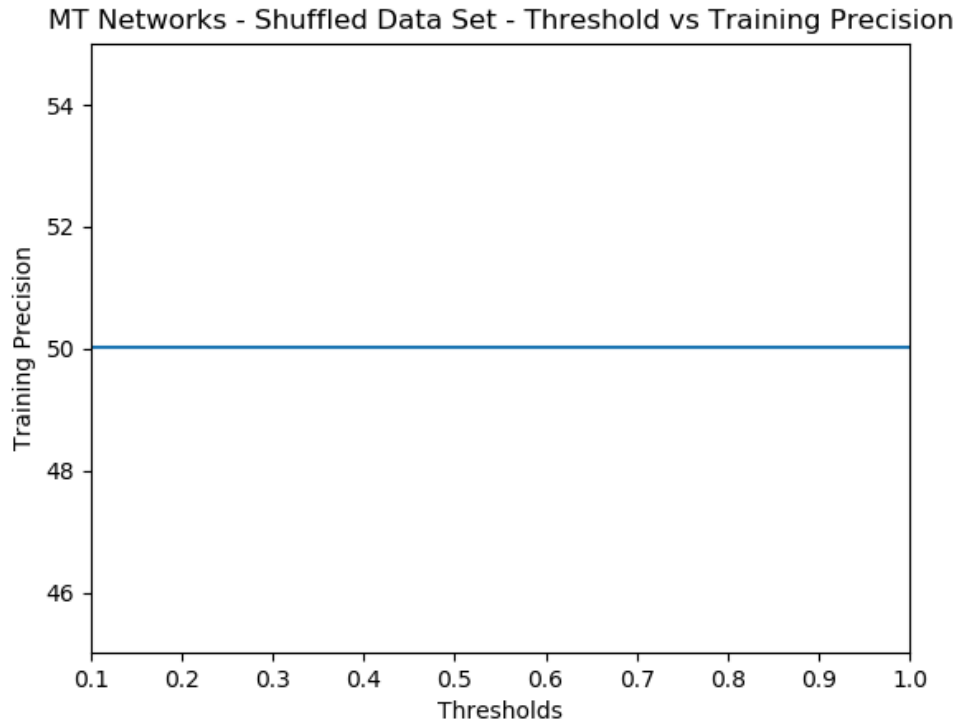
172


MT Networks - Seen Data Set - Threshold vs Training Precision
Training Precision
54
52
50
48
46
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

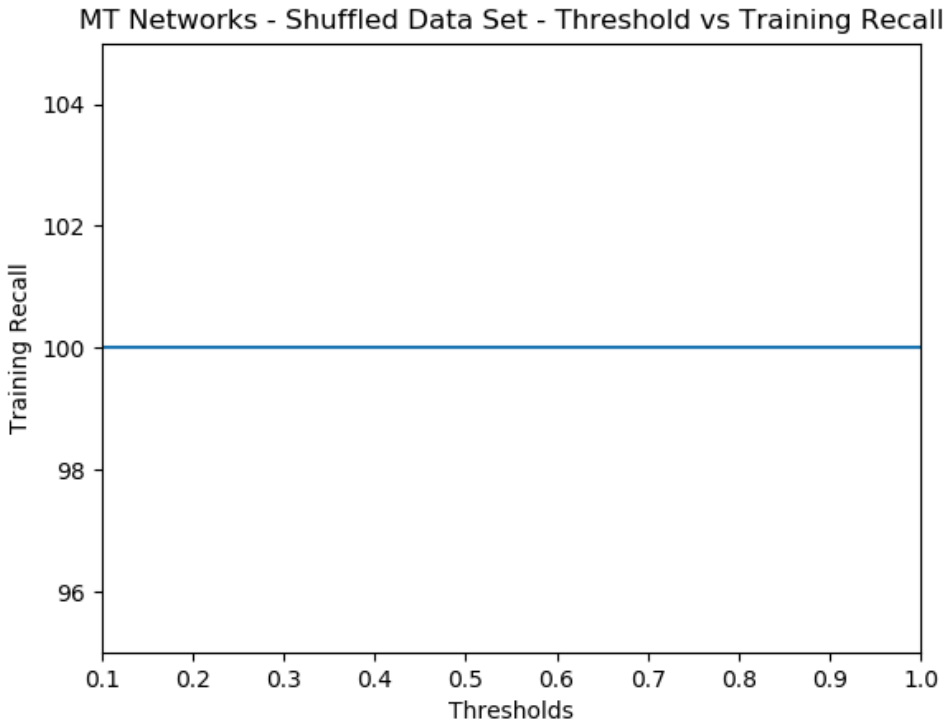
173


MT Networks - Shuffled Data Set - Total Training Loss
Total Training Loss
166
164
162
160
158
156
154
152
0
250
500
750
1000
1250
1500
1750
2000
Num of Iterations

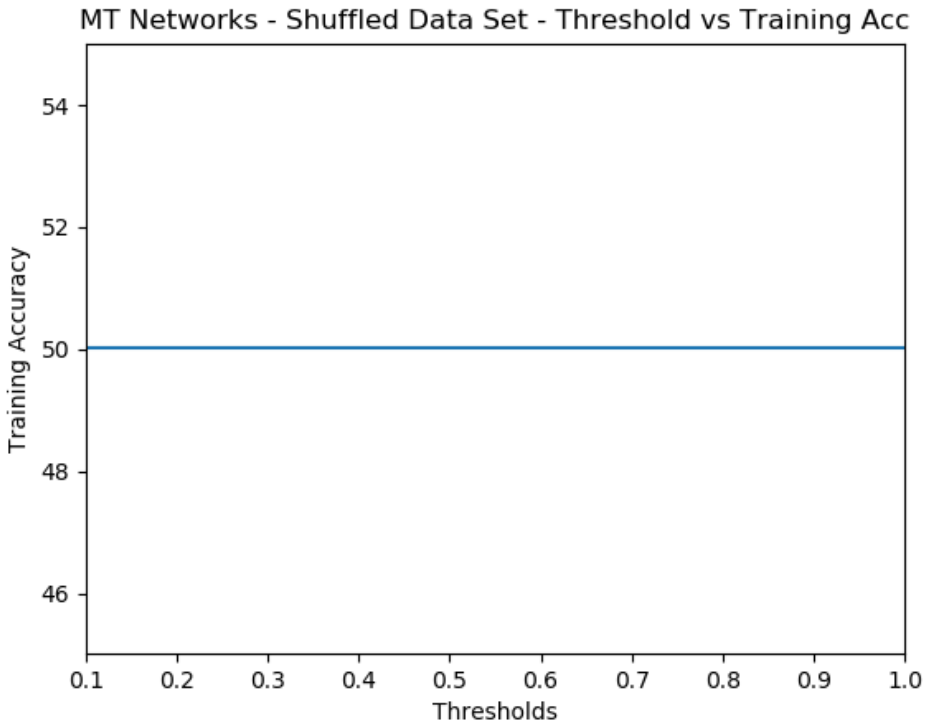
174


MT Networks - Shuffled Data Set - Threshold vs Training Precision
Training Precision
54
52
50
48
46
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

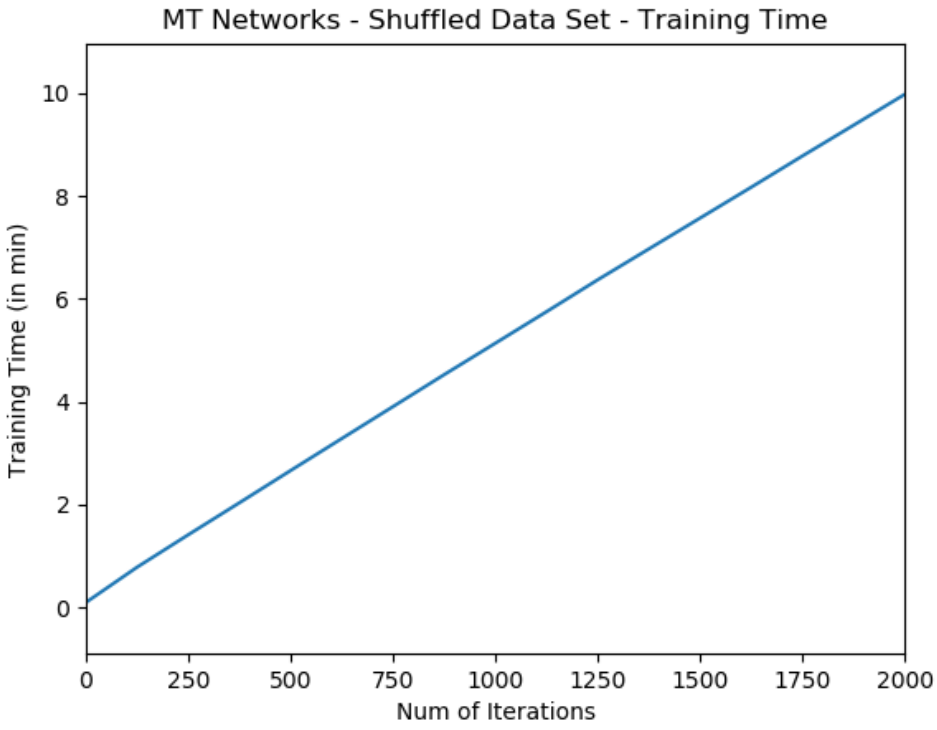
175


MT Networks - Shuffled Data Set - Threshold vs Training Recall
Training Recall
104
102
100
98
96
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

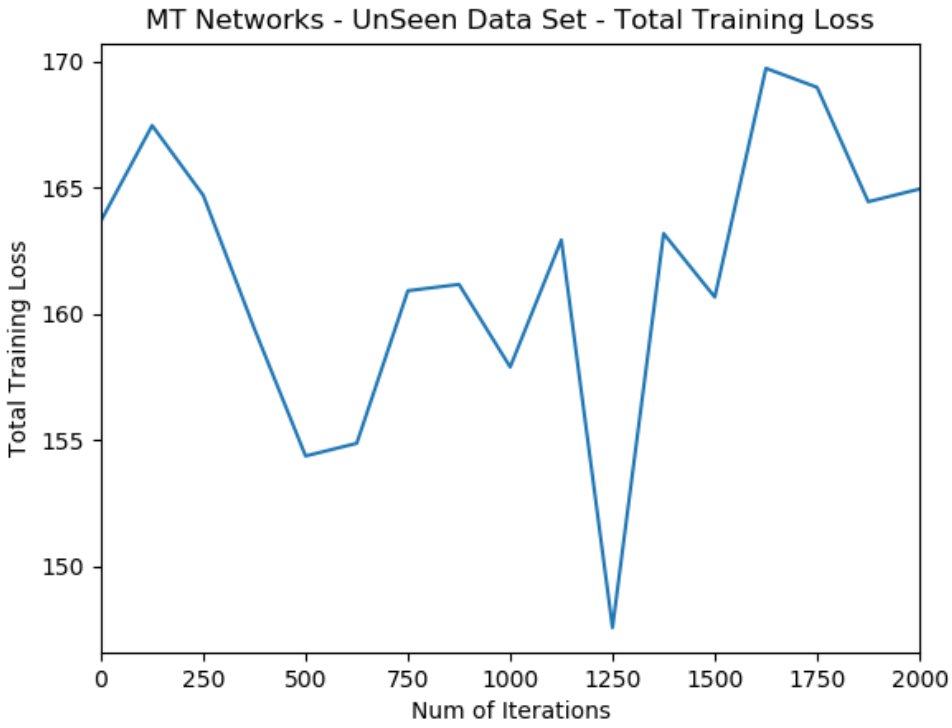
176


MT Networks - Shuffled Data Set - Threshold vs Training Acc
Training Accuracy
54
52
50
48
46
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

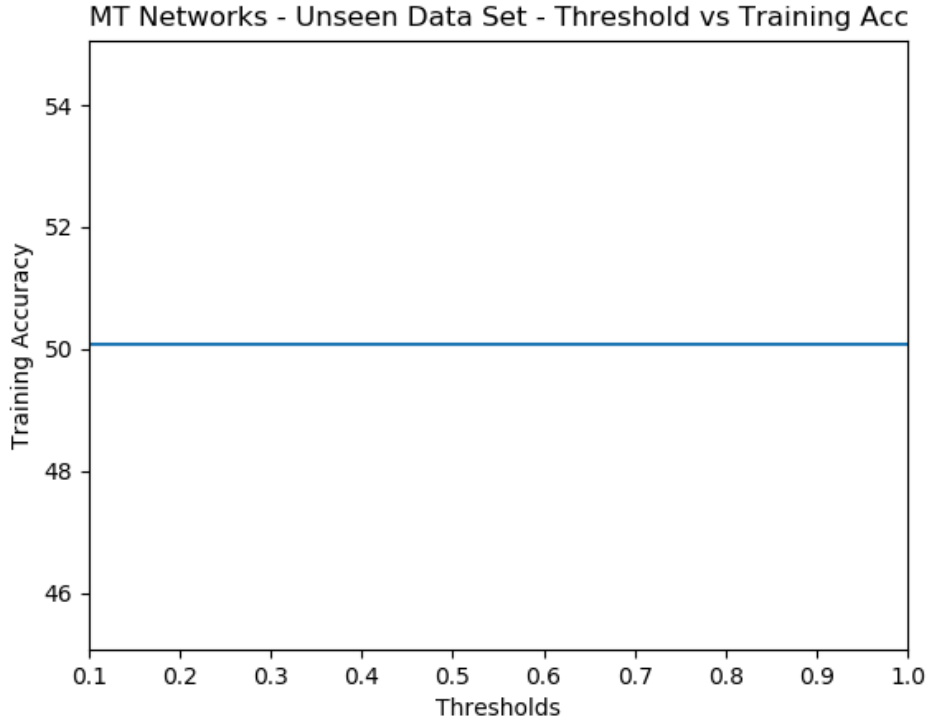
177


MT Networks - Shuffled Data Set - Training Time
Training Time (in min)
10
8
6
4
2
0
0
250
500
750
1000
1250
1500
1750
2000
Num of Iterations

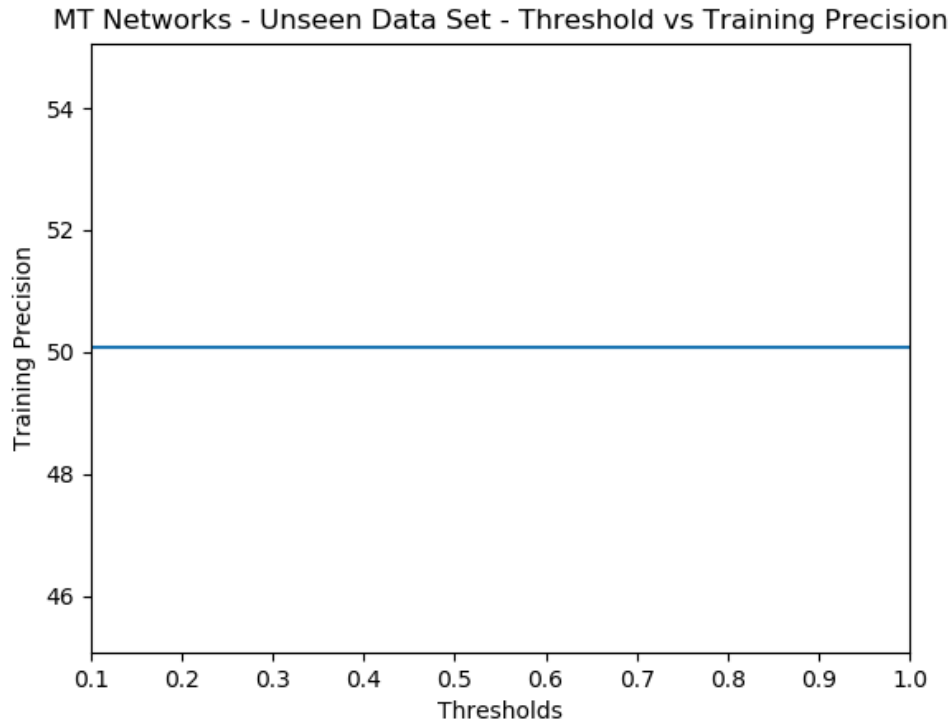
178


MT Networks - UnSeen Data Set - Total Training Loss
Total Training Loss
170
165
160
155
150
0
250
500
750
1000
1250
1500
1750
2000
Num of Iterations

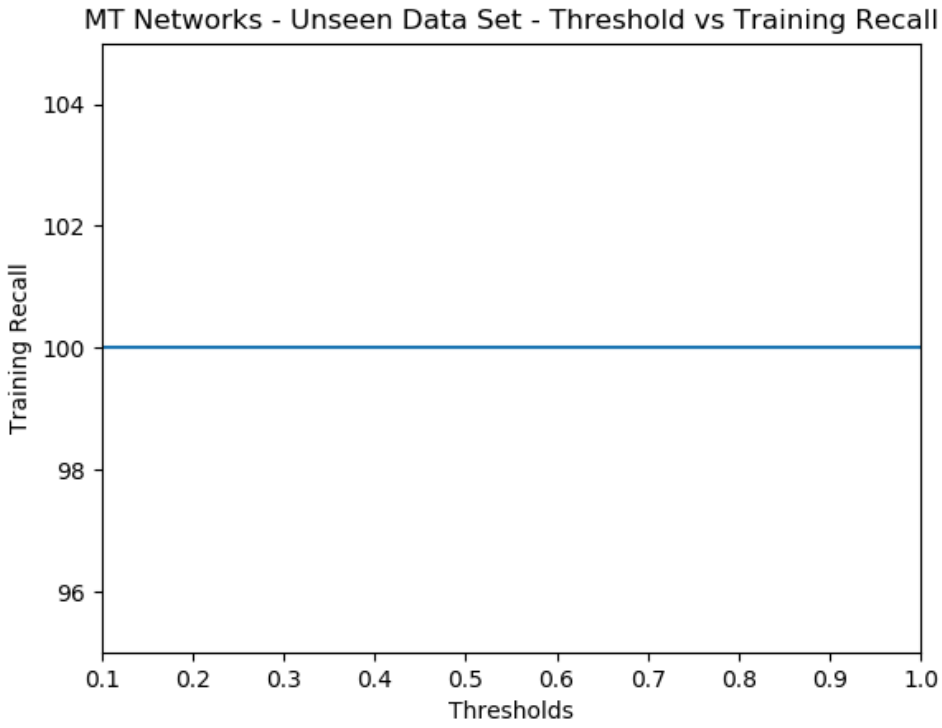
179


MT Networks - Unseen Data Set - Threshold vs Training Acc
Training Accuracy
54
52
50
48
46
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

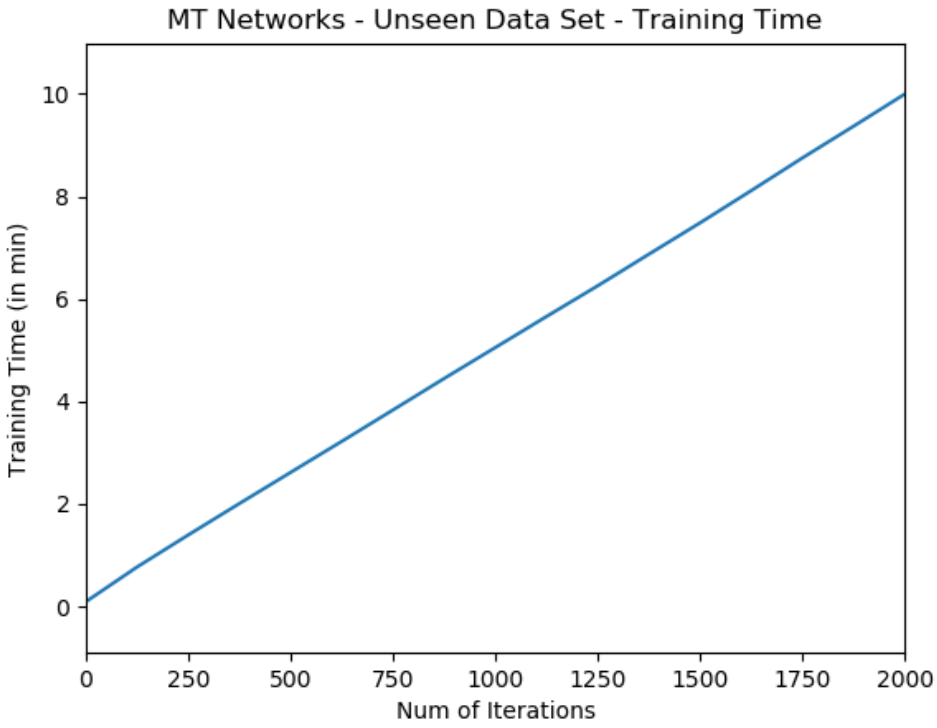
180


MT Networks - Unseen Data Set - Threshold vs Training Precision
Training Precision
54
52
50
48
46
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

181


MT Networks - Unseen Data Set - Threshold vs Training Recall
Training Recall
104
102
100
98
96
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Thresholds

182



As can be noticed from all the graph, not enough iterations were run and more training time was needed to obtain much better metrics. Here are the best values produced for 12000 epochs, where training set size equals the batch size:

```
Seen Data
--------------------------------------------------
Model Training Complete. Model Saved.
Train Loss: [120.88573, 8.562738, 6.7998214, 7.303512, 12.592262, 0.25184533, 6.296131, 11.333036, 12.844108, 1.007381, 14.607023, 3.2739882,
14.103333, 9.57012, 8.059048, 4.281369, 0.46875, 0.578125, 0.546875, 0.21875, 0.984375, 0.609375, 0.296875, 0.203125, 0.9375, 0.09375, 0.796875,
0.125, 0.40625, 0.5, 0.734375]
Best validation loss: [125.96293, 9.388783, 7.285371, 7.341783, 13.72457, 0.11282672, 5.8428054, 12.112754, 13.676215, 0.5318972, 14.965668,
3.1349707, 14.667482, 9.25178, 8.381399, 5.544621, 0.4175, 0.548, 0.5445, 0.1485, 0.993, 0.6375, 0.2485, 0.1515, 0.967, 0.0715, 0.8055, 0.09,
0.426, 0.48, 0.656]
Best training accuracy: 13.683333333333332
Best validation accuracy: 13.611458333333335
Best validation loss iter number: 8500

For similarity detection:
Best training accuracy: 50.06126914660831
Best training precision: 50.06126914660831
Best training recall: 100.0
Best validation accuracy: 60.4
Best validation precision: 60.4
Best validation recall: 100.0
Best cosine threshold: 0.1
--------------------------------------------------
```

```
Shuffled Data
--------------------------------------------------
Model Training Complete. Model Saved.
Train Loss: [152.87007, 10.829346, 11.333035, 8.059048, 9.57012, 15.86625, 4.78506, 13.599644, 3.7776785, 15.110714, 14.858869,
11.081191, 10.325655, 11.584881, 4.5332146, 7.5553575, 0.328125, 0.296875, 0.5, 0.40625, 0.015625, 0.703125, 0.15625, 0.765625,
0.0625, 0.078125, 0.3125, 0.359375, 0.28125, 0.71875, 0.53125]
Best validation loss: [165.71829, 6.479468, 12.257816, 6.9227138, 14.828664, 15.803812, 6.0039854, 15.368623, 2.8851402, 15.489509,
14.868959, 13.603683, 7.6802626, 14.038874, 8.123509, 11.363258, 0.598, 0.2395, 0.5705, 0.08, 0.0195, 0.6275, 0.0465, 0.821, 0.039,
0.0775, 0.156, 0.5235, 0.129, 0.496, 0.295]
Best training accuracy: 36.4625
Best validation accuracy: 38.93333333333333
Best validation loss iter number: 875

For similarity detection:
Best training accuracy: 50.013691931540336
Best training precision: 50.013691931540336
Best training recall: 100.0
Best validation accuracy: 50.36190476190476
Best validation precision: 50.36190476190476
Best validation recall: 100.0
Best cosine threshold: 0.1
--------------------------------------------------


Unseen Data
--------------------------------------------------
Model Training Complete. Model Saved.
Train Loss: [147.5813, 5.28875, 11.333036, 11.08119, 8.814584, 1.1920929e-07, 5.5405955, 13.347798, 12.844108, 15.362559, 11.836727,
11.836726, 14.355179, 12.088572, 6.296131, 7.5553575, 0.671875, 0.296875, 0.3125, 0.453125, 1.0, 0.65625, 0.171875, 0.203125, 0.046875,
0.265625, 0.265625, 0.109375, 0.25, 0.609375, 0.53125]
Best validation loss: [151.76001, 4.1342926, 12.757481, 10.613764, 9.646674, 0.23371242, 4.0778794, 13.184611, 13.19267, 15.312209,
13.007312, 12.66883, 14.578832, 12.660771, 6.1732254, 9.517729, 0.7435, 0.2085, 0.3415, 0.4015, 0.9855, 0.747, 0.182, 0.1815, 0.05,
0.193, 0.214, 0.0955, 0.2145, 0.617, 0.4095]
Best training accuracy: 30.39583333333333
Best validation accuracy: 28.33333333333333
Best validation loss iter number: 1250

For similarity detection:
Best training accuracy: 50.067052023121384
Best training precision: 50.067052023121384
Best training recall: 100.0
Best validation accuracy: 50.53333333333333
Best validation precision: 50.53333333333333
Best validation recall: 100.0
Best cosine threshold: 0.1
--------------------------------------------------
```

5 Conclusion

In conclusion, we have created the different models and compared them based on their metrics. We also noticed that the Deep Learning models could not be completely trained in a reasonable amount of time. Bayesian Networks trained faster but were less accurate than the Deep Net models. The Explainable AI models were better in terms of performance than the Bayesian Nets but were not as good as the Deep Net Models – the reason being that we traded off performance for explainability. Explainable AI models also took the longest to train in our implementation. And Explainable AI models made the basis of its decisions clear enough for a human understand it, instead of being a black box.

References:

1. https://towardsdatascience.com/one-shot-learning-with-siamese-networks-using-keras-17f34e75bb3d
2. https://github.com/hlamba28/One-Shot-Learning-with-Siamese-Networks/blob/master/Siamese%20on%20Omniglot%20Dataset.ipynb
3. https://github.com/akshaysharma096/Siamese-Networks/blob/master/Few%20Shot%20Learning%20-%20V1.ipynb
4. https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f