CAB420 - Assignment 1B

Callum McNeilage - n10482652

Connor Smith - n9991051

Queensland University of Technology

Problem 1: Training and Adapting Deep Networks

Network Design

The design of the network we chose for question 1, matches that used in the week 4 example 2 code for training a CNN. The only difference is that the input layer and output layers have been changed to match the data we've been given. The design is small consisting of three 2d convolutions with a max pooling in between. It ends with a flatten layer and 2 dense layers to classify the image. The full structure can be seen in appendix 1. Due to issues with struggling to get the code to work in time, for the pre-trained network we used the vgg_2stage_CIFAR_bigger model from blackboard and trained it. The model is quite deep consisting of 24 layers and has been pre-trained on CIFAR dataset. CIFAR is a dataset also used for classification but used for different object types like automobile, airplane, etc. The model had the last layer removed so that a dense layer could be added for classification. The full layout of the network can be seen in appendix 2.

Type of Augmentation

The data was augmented before being passed through to the rest of the model. The types of augmentation used were random rotations and random zoom. Both amounts of change were kept relatively small, as to avoid confusing the model with similar numbers i.e. rotate a 5 too much and it could look like a 2. The pictures are also small, only being 32x32 so a small value of zoom was used. Random translations were not used due to the small image size and random flips were not used as it would cause confusion. Examples of the augmented data are shown below

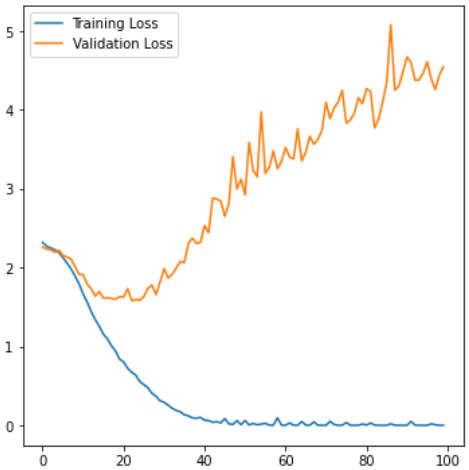
Figure 1

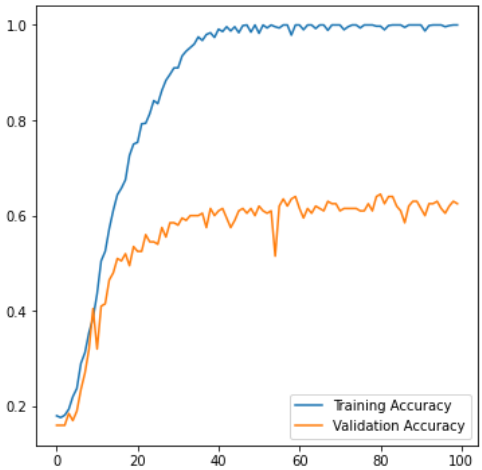
Computational Constraints

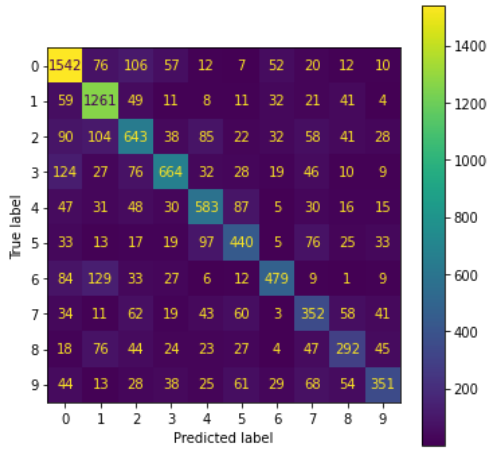
Due to having a decently powerful PC to train the models, all models were trained for 100 epochs and with a batch size of 32. Despite having sufficient power, a smaller model was chosen for the first two sections as it allowed for faster training and testing of the model.

Comparisons of Models

After training the basic model with non-augmented data, it produced accuracy of 100% on the training set







and 62% with validation. The training performance can be seen below:

The basic model with augmented data produced an accuracy of 98% on the training set and 75% on validation. The training performance can be seen below:

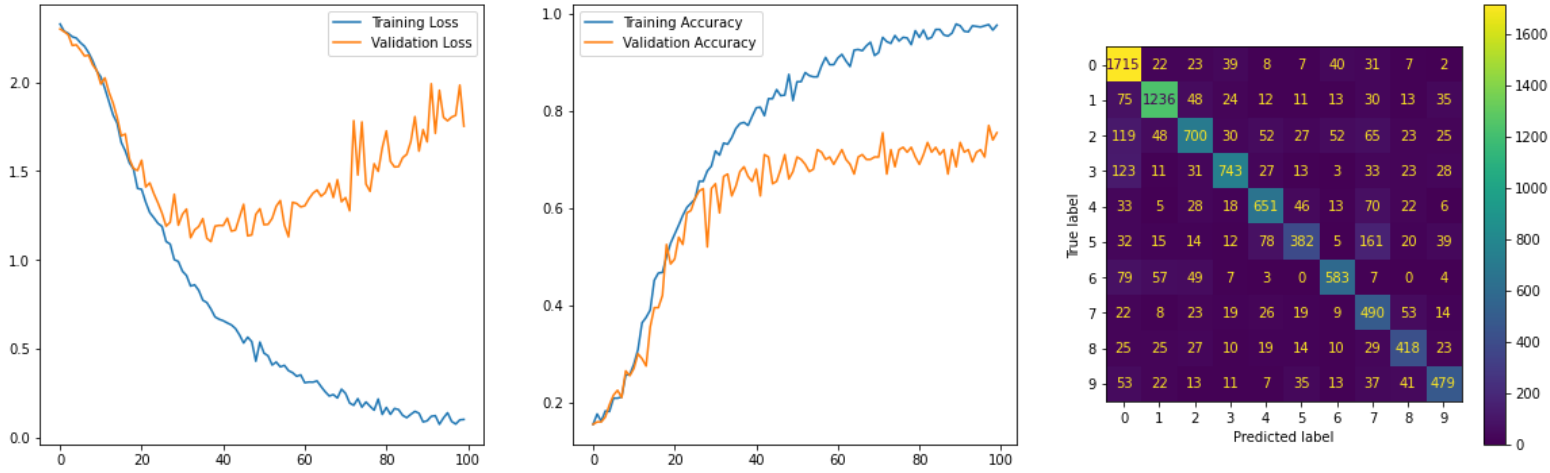


Figure 3

Comparing the non augmented data vs the augmented data, the model with the augmented data performed better resulting in 10% more accuracy in validation. The model without augmented data seemed to have overfit to the dataset with the last few epochs showing 100% accuracy on training with no increase to validation.

```
Epoch 95/100
25/25 [==============================] - 0s 7ms/step - loss: 6.5398e-05 - accuracy: 1.0000 -
val_loss: 4.4679 - val_accuracy: 0.6300
Epoch 96/100
25/25 [==============================] - 0s 6ms/step - loss: 3.6226e-05 - accuracy: 1.0000 -
val_loss: 4.6105 - val_accuracy: 0.6150
Epoch 97/100
25/25 [==============================] - 0s 6ms/step - loss: 0.0027 - accuracy: 0.9995 -
val_loss: 4.3929 - val_accuracy: 0.6050
Epoch 98/100
25/25 [==============================] - 0s 6ms/step - loss: 0.0089 - accuracy: 0.9981 -
val_loss: 4.2590 - val_accuracy: 0.6200
Epoch 99/100
25/25 [==============================] - 0s 6ms/step - loss: 2.1359e-04 - accuracy: 1.0000 -
val_loss: 4.4443 - val_accuracy: 0.6300
Epoch 100/100
25/25 [==============================] - 0s 6ms/step - loss: 5.5059e-05 - accuracy: 1.0000 -
val_loss: 4.5467 - val_accuracy: 0.6250
```

The augmented data prevented the overfitting. The model never hit 100% accuracy and the increased validation accuracy shows this.

```
Epoch 95/100
25/25 [==============================] - 0s 12ms/step - loss: 0.1128 - accuracy: 0.9688 -
val_loss: 1.8018 - val_accuracy: 0.7150
Epoch 96/100
25/25 [==============================] - 0s 12ms/step - loss: 0.1132 - accuracy: 0.9748 -
val_loss: 1.7821 - val_accuracy: 0.7200
Epoch 97/100
25/25 [==============================] - 0s 13ms/step - loss: 0.0605 - accuracy: 0.9845 -
val_loss: 1.8031 - val_accuracy: 0.7050
Epoch 98/100
25/25 [==============================] - 0s 13ms/step - loss: 0.1006 - accuracy: 0.9696 -
val_loss: 1.8132 - val_accuracy: 0.7700
Epoch 99/100
25/25 [==============================] - 0s 13ms/step - loss: 0.0673 - accuracy: 0.9723 -
val_loss: 1.9831 - val_accuracy: 0.7400
Epoch 100/100
25/25 [==============================] - 0s 13ms/step - loss: 0.0806 - accuracy: 0.9814 -
val_loss: 1.7520 - val_accuracy: 0.7550
```

Augmenting the data has produced better results, allowing for the model to learn from ‘new’ data. Whilst actual new data would produce increased results, the small dataset made this not possible, thus augmenting the data is the better option. In terms of speed, both models were reasonably quick with the non augmented model being slightly faster at 7ms/step vs 17ms/step for the augmented.

Looking at the pre-trained model, it produced a disappointing 14% on the training set and 9.5% on validation. The training performance can be seen below:

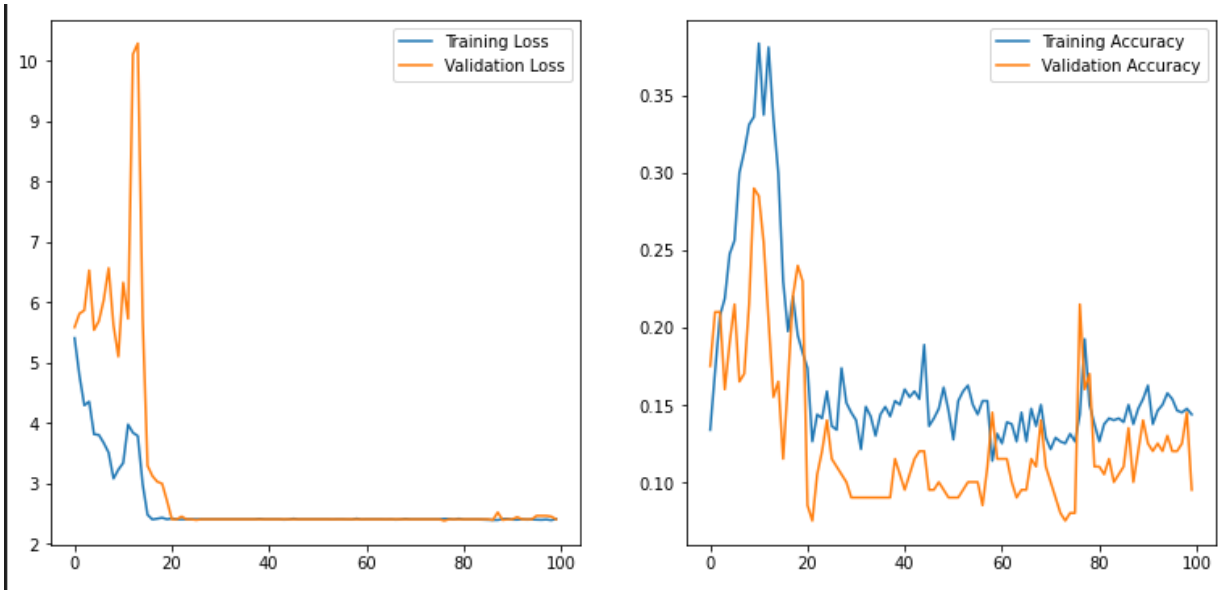


Figure 4

The bad accuracy could be due to the previous model being used to classify objects rather than numbers. The lack of data could also indicate the bad performance. For around 15 epochs performance was as expected, slowly rising before both training and validation accuracy sharply dropped.

```
Epoch 15/100
25/25 [==============================] - 2s 88ms/step - loss: 3.1901 - accuracy: 0.3148 -
val_loss: 5.7288 - val_accuracy: 0.1650
Epoch 16/100
25/25 [==============================] - 2s 86ms/step - loss: 2.5232 - accuracy: 0.2516 -
val_loss: 3.2861 - val_accuracy: 0.1150
Epoch 17/100
25/25 [==============================] - 2s 85ms/step - loss: 2.3934 - accuracy: 0.2013 -
val_loss: 3.1159 - val_accuracy: 0.1650
Epoch 18/100
25/25 [==============================] - 2s 86ms/step - loss: 2.3929 - accuracy: 0.2358 -
val_loss: 3.0206 - val_accuracy: 0.2200
Epoch 19/100
25/25 [==============================] - 2s 88ms/step - loss: 2.4584 - accuracy: 0.2222 -
val_loss: 2.9889 - val_accuracy: 0.2400
Epoch 20/100
25/25 [==============================] - 2s 87ms/step - loss: 2.3979 - accuracy: 0.1897 -
val_loss: 2.7053 - val_accuracy: 0.2300
```

Augmented data was not used so the lack of new samples may not have been enough to sufficiently retrain the model. There is also the possibility that human error is involved. Compared to the other models, this model performed significantly worse in both accuracy and time. The time is due to the increased depth of the model though and cannot be compared to the other simple models.

Overall, the simple model with augmented data performed the best on this limited dataset. It allowed for more ‘new’ data for the model to learn off of and helped to prevent overfitting.

Problem 2: Person Re-Identification

This task required us to match a detected person to a gallery of previously seen people, and determine their identity. For this task we have developed a non-deep learning model and a deep learning model for the provided portion of the Market-1501 dataset. As we were provided with a folder of images rather than a .mat file for this task we used a combination of glob and cv2.imread functions in order to append each image into training, gallery and probe arrays.

```
train = []
gnd = []
files = glob.glob('Data/Q2/Q2/Q2/Training/*.jpg')
for myfile in files:
    image = cv2.imread(myfile, 0)
    gnd.append(myfile[23:27])
    train.append(image)

train = np.array(train)
gnd = np.array(gnd)

print('Training shape: ', train.shape)
print('Training gnd: ', gnd[1:10])
```

Evaluation of Non-Deep Learning method

To develop a non-deep learning method to solve this problem, we decided to follow the Week 6 PCA examples. We opted to use the Gallery dataset as our test dataset so that we could use the Probe dataset for validation later.

We reshaped our feature sets for the three datasets into a 2D dataset by combining the x and y values to allow us to use the decomposition.PCA() function to fit our data on the training set. We then used the developed model as a pca.transform on each of our three datasets.

```
nsamples, nx, ny = train_fea.shape
d2_train_dataset = train_fea.reshape((nsamples,nx*ny))

nsamples, nx, ny = test_fea.shape
d2_test_dataset = test_fea.reshape((nsamples,nx*ny))
```

```
nsamples, nx, ny = test_pro.shape
d2_probe_dataset = test_pro.reshape((nsamples,nx*ny))

pca = decomposition.PCA()
pca.fit(d2_train_dataset)
transformed = pca.transform(d2_train_dataset)
transformed_test = pca.transform(d2_test_dataset)
transformed_probe = pca.transform(d2_probe_dataset)
```

As a sanity check at this stage, we completed a 90%, 95% and 99% reconstruction of image in position pca.components_[0] as well as generating a cumulative sum graph of the explained variance.

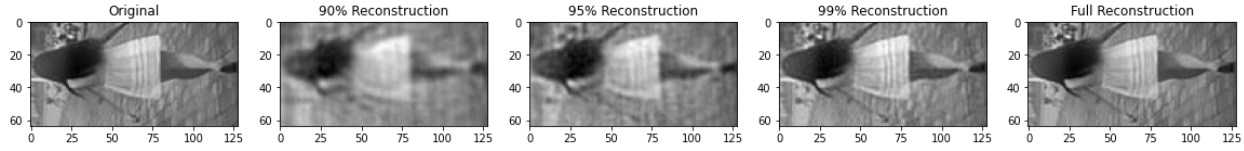


Figure 5

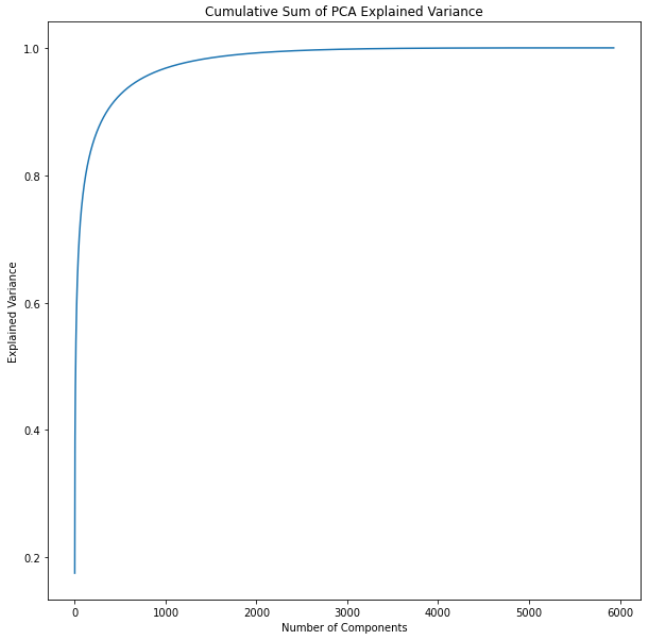


Figure 6

Finally, we graphed a CMC Curve for the 90%, 95% and 99% datasets respectively in order to compare performance of the model given a slightly reduced dataset.

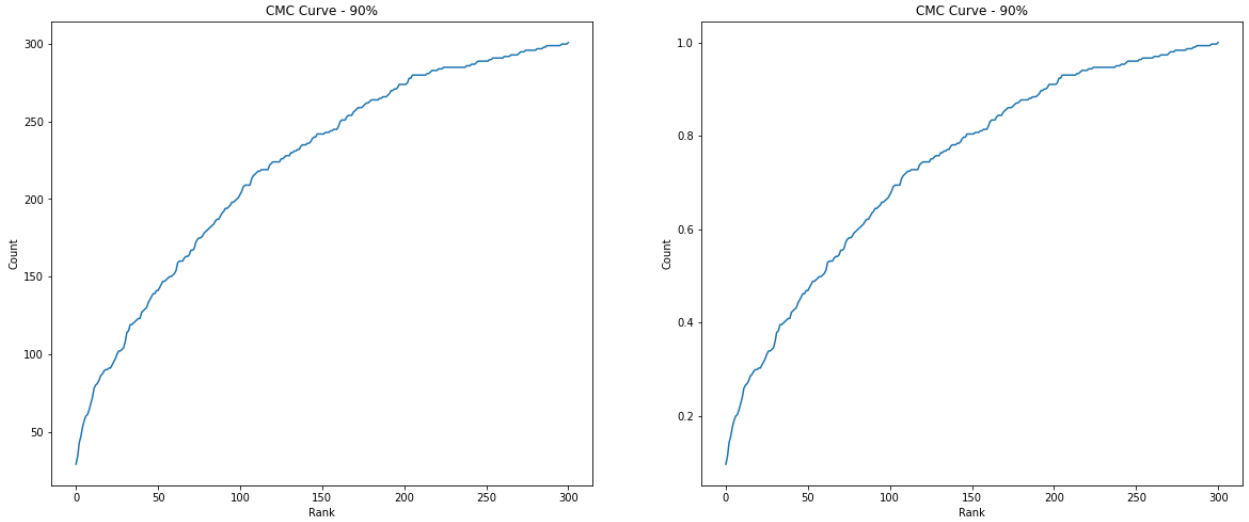


Figure 7

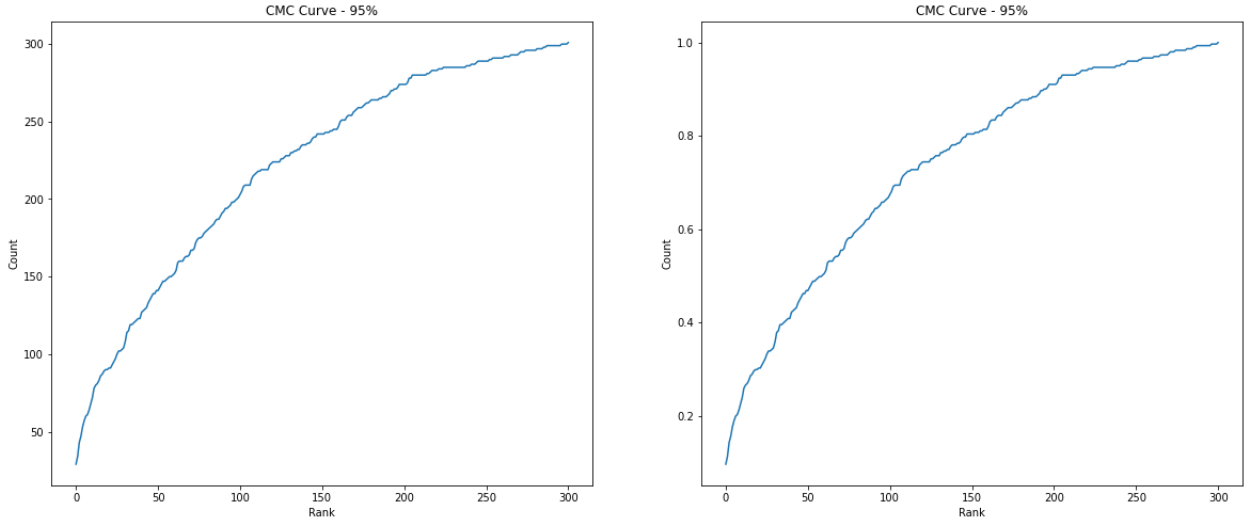


Figure 8

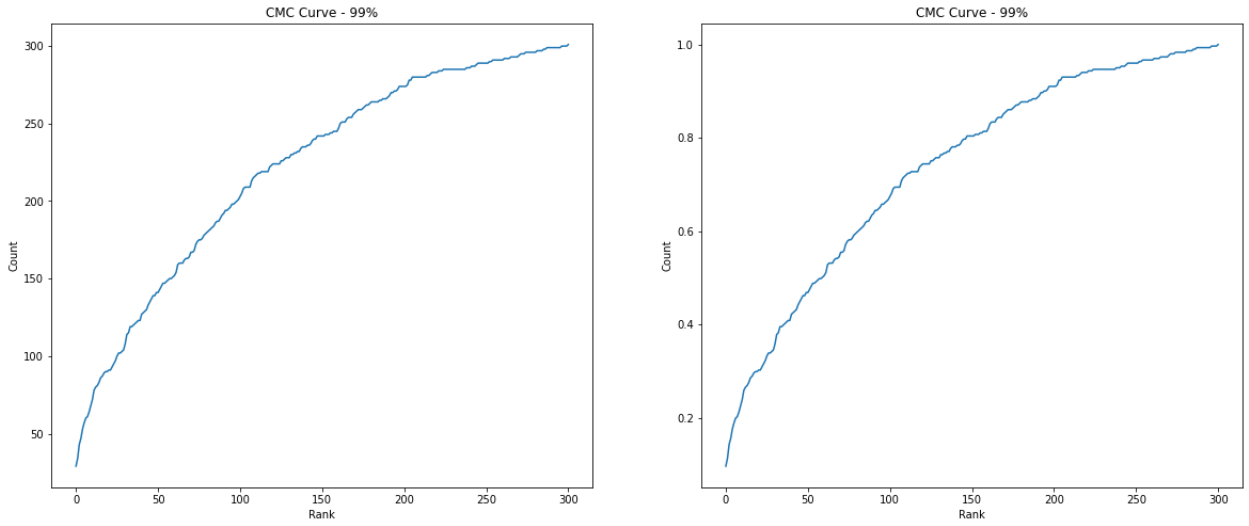


Figure 9

Evaluation of Deep Learning Method

For our Deep Learning method, it was decided to follow the Siamese model completed in Week 7 Example 1 (See Appendix 3 for model and network summary).

We used the PairGenerator code from the example to find pairs of images in the training dataset with the same gnd (ID number) before training the Siamese Network 100 times with a batch size of 32.

```
batch_size = 32
training_gen = PairGenerator(train_fea, gnd, batch_size)

siamese_test_x, siamese_test_y = GetSiameseData(test_fea, test_gnd, 5933)
siamese_network.fit(training_gen, steps_per_epoch = 256 // batch_size, epochs=100, validation_data =
(siamese_test_x, siamese_test_y))
```

We then used the trained network to predict gallery and probe values.

```
gal_x, gal_y = GetSiameseData(test_fea, test_gnd, 301)
gal_predict = siamese_network.predict(gal_x)

pro_x, pro_y = GetSiameseData(test_pro_fea, test_pro_gnd, 301)
pro_predict = siamese_network.predict(pro_x)
```

Finally, we generated a CMC Curve for the network. Unfortunately, due to time constraints we were not able to complete this for Top-1, Top-5 and Top-10 datasets, instead the CMC is for whole generated dataset with no information removed.

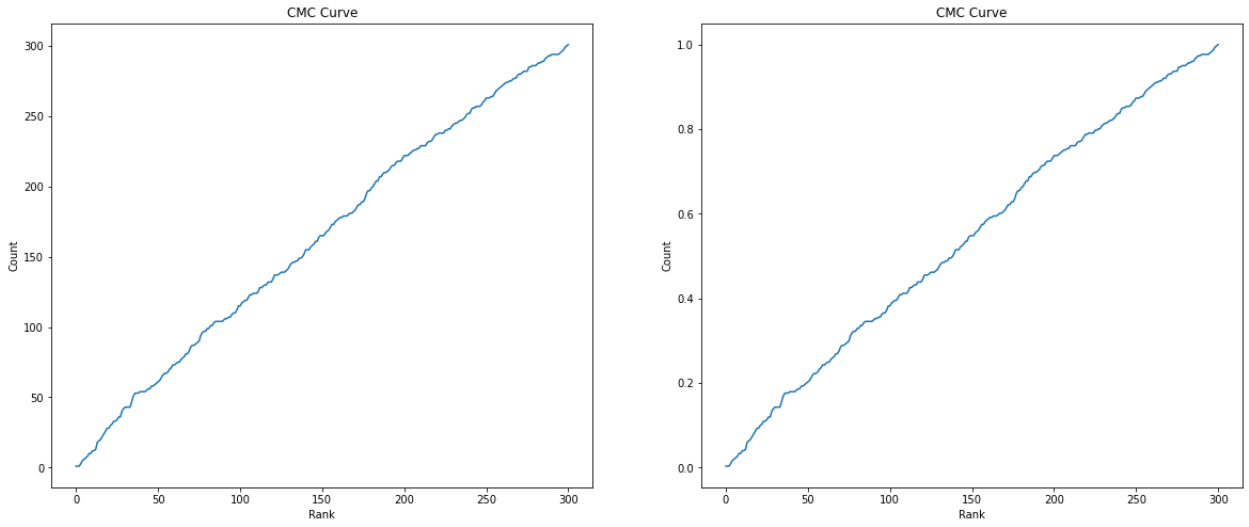


Figure 10

Comparison of Performance

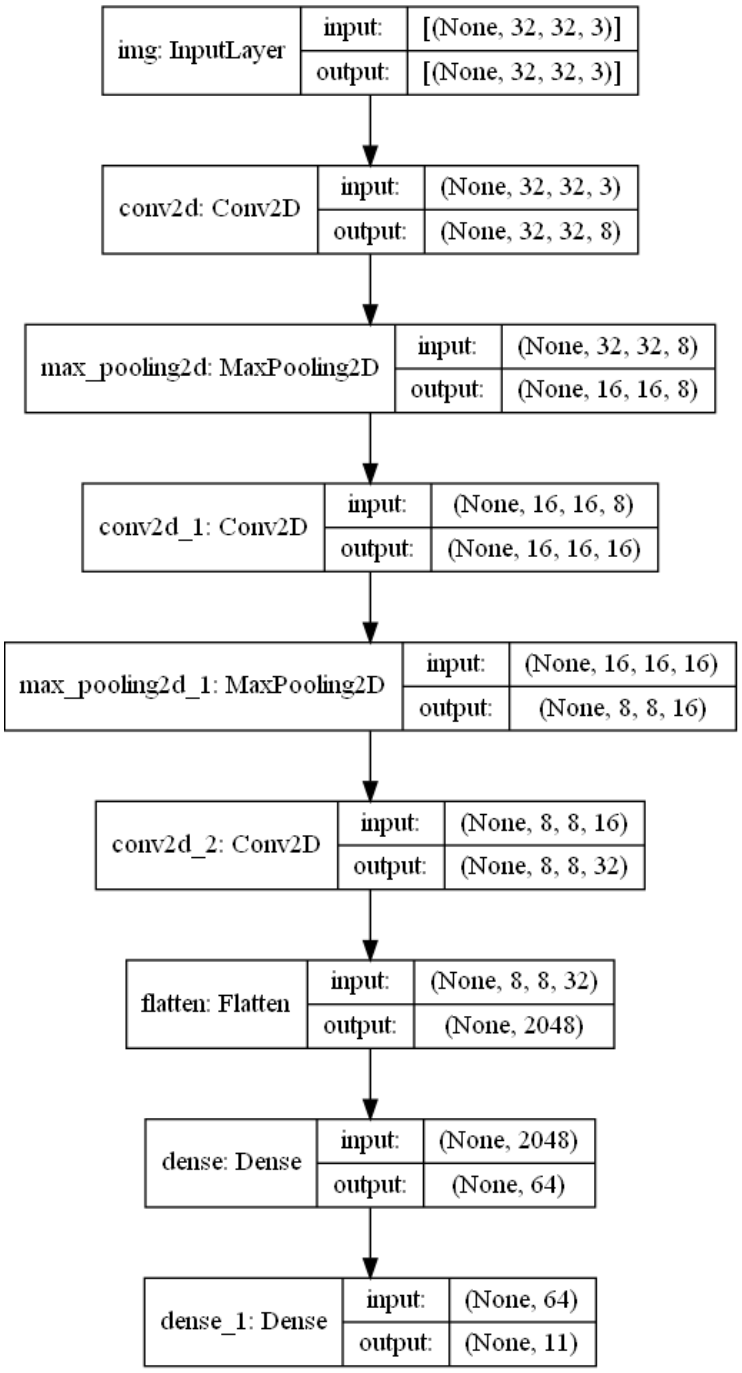
Both Deep Learning and Non-Deep Learning approaches have their strengths and weaknesses. A Deep Learning model is perfect for large datasets of unstructured and unlabeled data, however, if there is not enough data for the model to train on or if there is a lot of similar data, a Deep Learning model will be highly inaccurate or in the worst case will be no different from randomly selecting any ID in the dataset for a given image. For our dataset, our Deep Learning model was only achieving between 50% and 60% accuracy (See Appendix 4). We believe this is because of a combination of only being provided a subset of the entire Market-1501 dataset as well as using greyscale images that, at times, could be particularly low quality. As can be seen when

comparing the CMC Curves of the Deep Learning solution in Figure 10 with our PCA models in Figures 7, 8 & 9, this resulted in significantly less accuracy generated in our Deep Learning model. However, this does not mean our PCA model is the best model possible as the CMC Curves generated are less than ideal for an accurate model, where the optimal curve would approach 1.0 as quickly as possible.

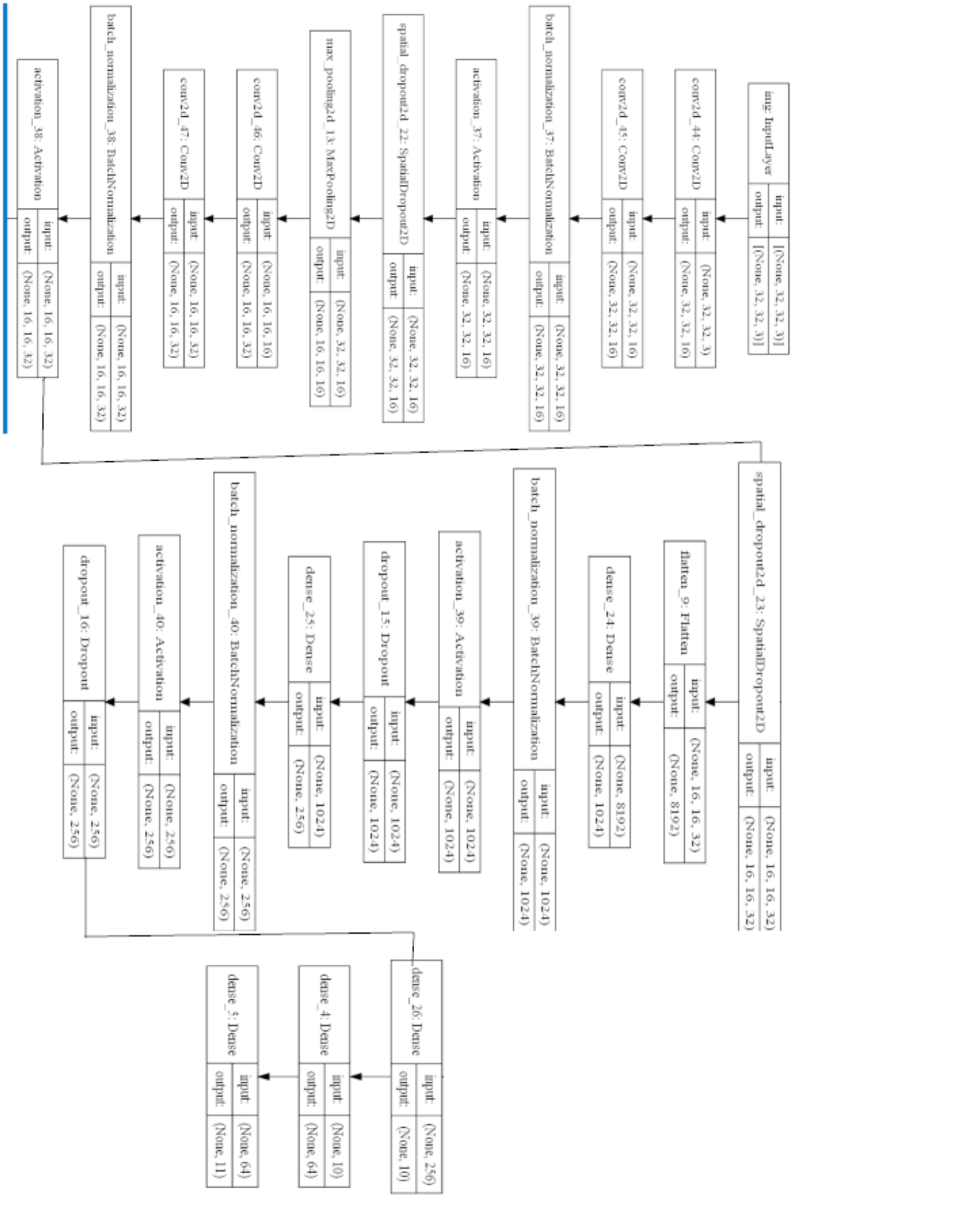
We believe that if we were able to use colour images instead of greyscale, this would provide a wider range of diversity within images that are seen by the model and would lead to less false positive matches and improve overall accuracy of the model. Additionally, having access to the entire dataset may improve accuracy as well.

Appendix

Appendix 1



Appendix 2


ing: InputLayer
input: [(None, 32, 32, 3)]
output: [(None, 32, 32, 3)]
conv2d_44: Conv2D
input: (None, 32, 32, 3)
output: (None, 32, 32, 16)
conv2d_45: Conv2D
input: (None, 32, 32, 16)
output: (None, 32, 32, 16)
batch_normalization_37: BatchNormalization
input: (None, 32, 32, 16)
output: (None, 32, 32, 16)
activation_37: Activation
input: (None, 32, 32, 16)
output: (None, 32, 32, 16)
spatial_dropout2d_22: SpatialDropout2D
input: (None, 32, 32, 16)
output: (None, 32, 32, 16)
max_pooling2d_13: MaxPooling2D
input: (None, 32, 32, 16)
output: (None, 16, 16, 16)
conv2d_46: Conv2D
input: (None, 16, 16, 16)
output: (None, 16, 16, 32)
conv2d_47: Conv2D
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
batch_normalization_38: BatchNormalization
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
activation_38: Activation
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
spatial_dropout2d_23: SpatialDropout2D
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
flatten_9: Flatten
input: (None, 16, 16, 32)
output: (None, 8192)
dense_24: Dense
input: (None, 8192)
output: (None, 1024)
batch_normalization_39: BatchNormalization
input: (None, 1024)
output: (None, 1024)
activation_39: Activation
input: (None, 1024)
output: (None, 1024)
dropout_15: Dropout
input: (None, 1024)
output: (None, 1024)
dense_25: Dense
input: (None, 1024)
output: (None, 256)
batch_normalization_40: BatchNormalization
input: (None, 256)
output: (None, 256)
activation_40: Activation
input: (None, 256)
output: (None, 256)
dropout_16: Dropout
input: (None, 256)
output: (None, 256)
dense_26: Dense
input: (None, 256)
output: (None, 10)
dense_4: Dense
input: (None, 10)
output: (None, 64)
dense_5: Dense
input: (None, 64)
output: (None, 11)

Appendix 3

```
Model: "SiameseBranch"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128, 64, 1)]      0
_________________________________________________________________
conv2d (Conv2D)              (None, 128, 64, 8)        80
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 64, 8)        584
_________________________________________________________________
batch_normalization (BatchNo (None, 128, 64, 8)        32
_________________________________________________________________
activation (Activation)      (None, 128, 64, 8)        0
_________________________________________________________________
spatial_dropout2d (SpatialDr (None, 128, 64, 8)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 32, 8)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 32, 16)        1168
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 32, 16)        2320
_________________________________________________________________
batch_normalization_1 (Batch (None, 64, 32, 16)        64
_________________________________________________________________
activation_1 (Activation)    (None, 64, 32, 16)        0
_________________________________________________________________
spatial_dropout2d_1 (Spatial (None, 64, 32, 16)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 16, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 16, 32)        4640
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 32, 16, 32)        9248
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 16, 32)        128
_________________________________________________________________
activation_2 (Activation)    (None, 32, 16, 32)        0
_________________________________________________________________
spatial_dropout2d_2 (Spatial (None, 32, 16, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 16384)             0
_________________________________________________________________
dense (Dense)                (None, 256)               4194560
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
activation_3 (Activation)    (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                8224
=================================================================
Total params: 4,222,072
Trainable params: 4,221,448
Non-trainable params: 624
_________________________________________________________________
```

```
Model: "SiameseNetwork"
__________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================
InputA (InputLayer)             [(None, 128, 64, 1)] 0
__________________________________________________________________________
InputB (InputLayer)             [(None, 128, 64, 1)] 0
__________________________________________________________________________
SiameseBranch (Functional)      (None, 32)           4222072     InputA[0][0]
                                                                 InputB[0][0]
__________________________________________________________________________
concatenate (Concatenate)       (None, 64)           0           SiameseBranch[0][0]
                                                                 SiameseBranch[1][0]
__________________________________________________________________________
dense_2 (Dense)                 (None, 128)          8320        concatenate[0][0]
__________________________________________________________________________
dense_3 (Dense)                 (None, 1)            129         dense_2[0][0]
==========================================================================
Total params: 4,230,521
Trainable params: 4,229,897
Non-trainable params: 624
__________________________________________________________________________
```

Appendix 4

```
Epoch 91/100
8/8 [==============================] - 24s 3s/step - loss: 0.6609 - accuracy: 0.6099 - val_loss: 0.6991 - val_accuracy: 0.5036
Epoch 92/100
8/8 [==============================] - 24s 3s/step - loss: 0.6250 - accuracy: 0.6646 - val_loss: 0.7367 - val_accuracy: 0.5004
Epoch 93/100
8/8 [==============================] - 24s 3s/step - loss: 0.6300 - accuracy: 0.6229 - val_loss: 0.7089 - val_accuracy: 0.5458
Epoch 94/100
8/8 [==============================] - 24s 3s/step - loss: 0.6381 - accuracy: 0.6497 - val_loss: 0.6773 - val_accuracy: 0.5626
Epoch 95/100
8/8 [==============================] - 24s 3s/step - loss: 0.6322 - accuracy: 0.6455 - val_loss: 0.7090 - val_accuracy: 0.5326
Epoch 96/100
8/8 [==============================] - 25s 4s/step - loss: 0.6311 - accuracy: 0.6260 - val_loss: 0.6744 - val_accuracy: 0.5840
Epoch 97/100
8/8 [==============================] - 26s 4s/step - loss: 0.6104 - accuracy: 0.6881 - val_loss: 0.7293 - val_accuracy: 0.5233
Epoch 98/100
8/8 [==============================] - 27s 4s/step - loss: 0.6583 - accuracy: 0.6302 - val_loss: 0.6909 - val_accuracy: 0.5788
Epoch 99/100
8/8 [==============================] - 25s 3s/step - loss: 0.6313 - accuracy: 0.5985 - val_loss: 0.7182 - val_accuracy: 0.5029
Epoch 100/100
8/8 [==============================] - 25s 4s/step - loss: 0.5951 - accuracy: 0.6626 - val_loss: 0.6781 - val_accuracy: 0.5581

<tensorflow.python.keras.callbacks.History at 0x172c8467be0>
```