COEN 166 Artificial Intelligence

Lab Assignment #6: Neural Network

Divya Syal: 1594122, Marianne Yamazaki Dorr: 1606975: , Suvass Ravala: 1593088

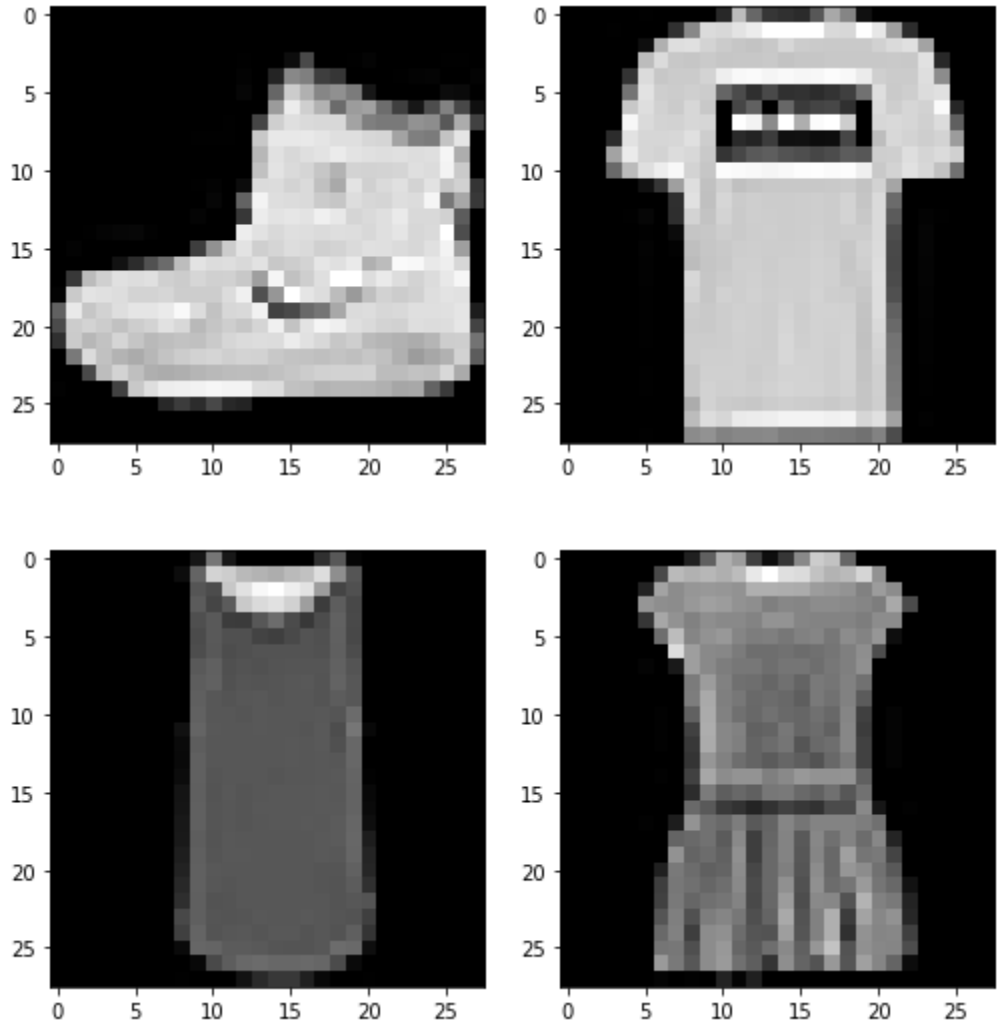
Lab 6 Contributions of each group member (in percentage): Each did 33%

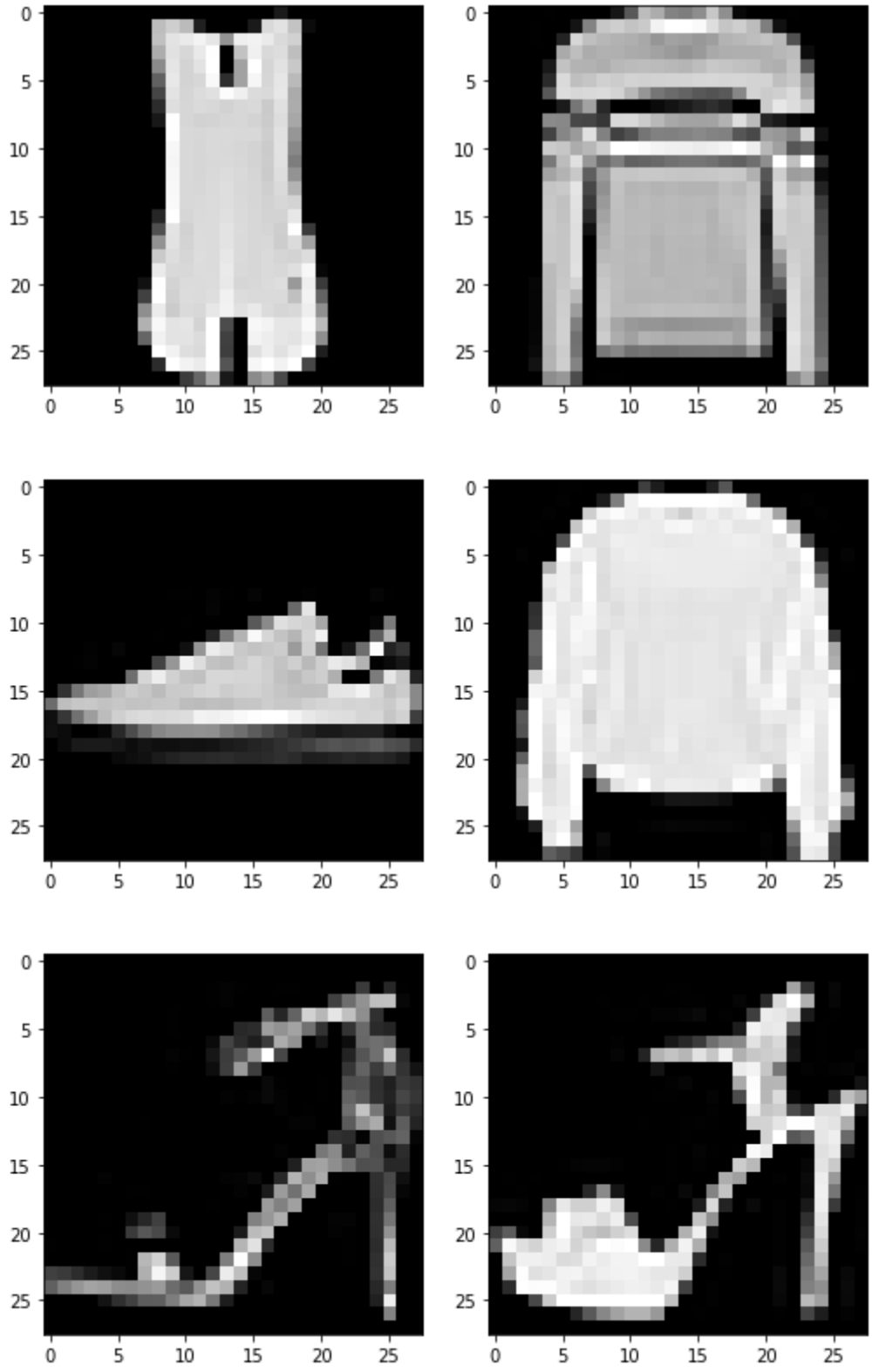
Problem 1 - Fashion mnist image recognition
Source Code:

```
“””
The network has one hidden layer, and an output layer.
The hidden layer has 512 nodes, and adopts the ReLU activation function;
the output layer has 10 nodes, and adopts the softmax activation function to output the class probabilities.
Use the “sparse_categorical_crossentropy” loss function, use ‘adam’ as the optimizer, use a batch size of 32, and train your model for 5 epochs.
"""
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from sklearn import metrics
import numpy as np

#create model
model = Sequential()
#1st layer, flatten input image (28x28 image) into a 1D vector (1x784)
model.add(Flatten(input_shape = (28, 28)))
#2nd layer, hidden layer with 512 nodes and ReLU activation function
model.add(Dense(512, activation = 'relu'))
#3rd layer, output layer with 10 nodes and softmax activation function to output the class probabilities
model.add(Dense(10, activation = 'softmax'))
model.summary()
#train/fit model
model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam',  metrics = ['accuracy'])
model.fit(x_train, y_train, epochs = 5, batch_size = 32)
'''
1.b Give the recognition accuracy rate of the test set, and show the confusion matrix
'''
#Testing accuracy
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
print("Accuracy Rate = ", test_acc)

predicted_probability = model.predict(x_test)
y_test_hat = np.argmax(predicted_probability, axis=1)

#generate confusion matrix
#diagonal elements represent the number of elements that were predicted correctly
#vertical = ground truth labels
#horizontal = predicted categories
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test, y_test_hat, labels=range(10))
print("Confusion Matrix:")
print(cm)
```

1.a Display images.





1.b Recognition accuracy rate for the test set, and the confusion matrix.

```
Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_22 (Flatten)        (None, 784)               0

 dense_40 (Dense)            (None, 512)               401920

 dense_41 (Dense)            (None, 10)                5130

=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.4749 -
accuracy: 0.8299
Epoch 2/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.3577 -
accuracy: 0.8698
Epoch 3/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.3206 -
accuracy: 0.8835
Epoch 4/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2968 -
accuracy: 0.8904
Epoch 5/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2822 -
accuracy: 0.8956
313/313 [==============================] - 1s 3ms/step - loss: 0.3612 -
accuracy: 0.8720
Accuracy Rate =  0.871999979019165
313/313 [==============================] - 1s 3ms/step
Confusion Matrix:
[[842   0  11  67   3   1  74   0   2   0]
 [  3 963   2  24   3   0   5   0   0   0]
 [ 16   0 760  26 106   0  92   0   0   0]
 [ 24   6   5 943   5   1  14   0   2   0]
 [  0   1  90  93 742   0  74   0   0   0]
 [  0   0   0   1   0 965   0  20   1  13]
 [160   0  65  61  60   0 652   0   2   0]
 [  0   0   0   0   0  13   0 945   0  42]
 [  8   0   9  16   7   2  13   3 942   0]
 [  0   0   0   0   0   9   1  24   0 966]]
```

Problem 2 - Fashion mnist image compression
Source Code:

```
import tensorflow as tf
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0 # normalize the pixel values to be in [0, 1]

from tensorflow.keras import datasets, layers, models
from keras.models import Sequential
import numpy as np
from keras.layers import Dropout, Flatten, Dense, Reshape
import math
"""
(1) The input layer is the flattened image, that is, a 1-dimensional vector with m × n elements
(2) A (Dense)compressed layer (hidden layer) with P nodes, P < m × n, followed by ReLU activation
(3) An (Dense)expansion layer (hidden layer) with m × n × T nodes, T = 2 is the expansion factor, followed by
ReLU activation
(4) An output layer with m × n nodes, followed by Sigmoid activation
(5) A reshape layer that convert the 1-dimensional vector output to the m × n 2-dimensional image
"""
#3 values of P: 10, 50, 200

#display the first 10 test original images
import matplotlib.pyplot as plt
for i in range(10):
    plt.imshow(x_test[i,:,:], cmap='gray')
    plt.show()

def psnr(P):
    model = models.Sequential()
    #1st layer, which will flatten the input image (dimension = 28 x 28 pixels) into a 1D vector (1 x 784)
    model.add(Flatten(input_shape = (28, 28)))
    #2nd layer, Compression layer, which has P number of nodes (P < 28x28), and it uses the ReLU activation function
    model.add(Dense(P, activation='relu'))
    #3rd layer, expansion layer with 28 x 28 x 2 nodes and uses ReLU activation function
```

```
    model.add(Dense(28*28*2, activation = 'relu'))
    #4th layer, output layer with 28x28 nodes, using a sigmoid activation
    model.add(Dense(28*28, activation = 'sigmoid'))
    #5th layer, reshape to convert the 1d vector back to a 28x28 2d image.
    model.add(Reshape((28, 28)))

    model.summary()

    #training/fitting the model with our fashion_mnist data that we imported
    model.compile(loss = 'mean_squared_error', optimizer = 'adam', metrics =
['accuracy'] )
    #dont need the labels because not classification task

    model.fit(x_train, x_train, epochs = 10, batch_size = 64)

    #MAXI = 1
    #compare the original picture with the decompressed image
    #original image= test image
    #decompressed image = what we get from model
    #use direct subtraction to get difference between original picture and
decompressed image
    predicted_images = model.predict(x_test)
    #calculate mse

    #display the first 10 images decompressed
    for i in range(10):
        plt.imshow(predicted_images[i,:,:], cmap='gray')
        plt.show()

    PSNR_values = 0
    for x in range(len(x_test)):
        difference = x_test[x] - predicted_images[x]
        dif = difference**2
        new_sum = sum(sum(dif))
        mse = new_sum/784
        #10log10 (MAXI^2/mse ),
        #peak signal to noise ratio
        PSNR = 10*math.log(1/mse, 10)
        PSNR_values += PSNR

    #average of PSNR list
    PSNR_avg = PSNR_values/len(x_test)
    return PSNR_avg


print(psnr(10), "\n")
```

```
print(psnr(50), "\n")
print(psnr(200), "\n")
```

2.a Results:

P = 10, PSNR average = 19.76734051269429

P = 50, PSNR average = 22.693932868164858

P = 200, PSNR average = 25.413642207878297

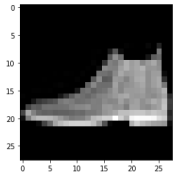
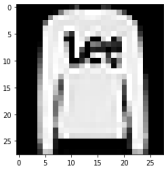
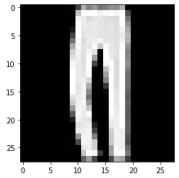
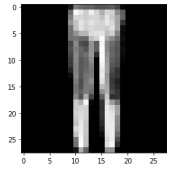
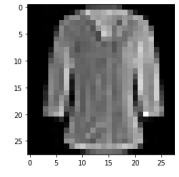
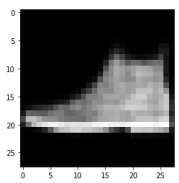
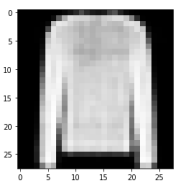
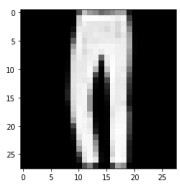
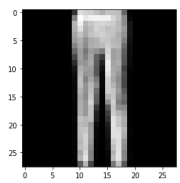
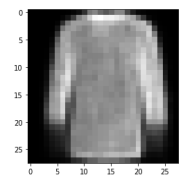
```
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_19 (Flatten)        (None, 784)               0

 dense_31 (Dense)            (None, 10)                7850

 dense_32 (Dense)            (None, 1568)              17248

 dense_33 (Dense)            (None, 784)               1230096

 reshape_9 (Reshape)         (None, 28, 28)            0

=================================================================
Total params: 1,255,194
Trainable params: 1,255,194
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
938/938 [==============================] - 20s 20ms/step - loss: 0.0261 -
accuracy: 0.1010
Epoch 2/10
938/938 [==============================] - 18s 20ms/step - loss: 0.0163 -
accuracy: 0.1345
Epoch 3/10
938/938 [==============================] - 17s 18ms/step - loss: 0.0150 -
accuracy: 0.1533
Epoch 4/10
938/938 [==============================] - 15s 16ms/step - loss: 0.0143 -
accuracy: 0.1644
Epoch 5/10
938/938 [==============================] - 19s 20ms/step - loss: 0.0139 -
accuracy: 0.1718
```

```
Epoch 6/10
938/938 [==============================] - 16s 17ms/step - loss: 0.0136 -
accuracy: 0.1769
Epoch 7/10
938/938 [==============================] - 17s 18ms/step - loss: 0.0134 -
accuracy: 0.1797
Epoch 8/10
938/938 [==============================] - 17s 18ms/step - loss: 0.0132 -
accuracy: 0.1837
Epoch 9/10
938/938 [==============================] - 16s 17ms/step - loss: 0.0130 -
accuracy: 0.1861
Epoch 10/10
938/938 [==============================] - 15s 16ms/step - loss: 0.0129 -
accuracy: 0.1883
313/313 [==============================] - 1s 4ms/step
19.76734051269429

Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_20 (Flatten)        (None, 784)               0

 dense_34 (Dense)            (None, 50)                39250

 dense_35 (Dense)            (None, 1568)              79968

 dense_36 (Dense)            (None, 784)               1230096

 reshape_10 (Reshape)        (None, 28, 28)            0

=================================================================
Total params: 1,349,314
Trainable params: 1,349,314
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
938/938 [==============================] - 15s 15ms/step - loss: 0.0201 -
accuracy: 0.1350
Epoch 2/10
938/938 [==============================] - 14s 15ms/step - loss: 0.0104 -
accuracy: 0.1945
Epoch 3/10
938/938 [==============================] - 15s 16ms/step - loss: 0.0090 -
accuracy: 0.2205
```

```
Epoch 4/10
938/938 [==============================] - 13s 14ms/step - loss: 0.0082 -
accuracy: 0.2370
Epoch 5/10
938/938 [==============================] - 13s 14ms/step - loss: 0.0077 -
accuracy: 0.2482
Epoch 6/10
938/938 [==============================] - 14s 15ms/step - loss: 0.0073 -
accuracy: 0.2553
Epoch 7/10
938/938 [==============================] - 14s 15ms/step - loss: 0.0071 -
accuracy: 0.2619
Epoch 8/10
938/938 [==============================] - 13s 14ms/step - loss: 0.0069 -
accuracy: 0.2665
Epoch 9/10
938/938 [==============================] - 13s 14ms/step - loss: 0.0067 -
accuracy: 0.2704
Epoch 10/10
938/938 [==============================] - 13s 14ms/step - loss: 0.0066 -
accuracy: 0.2728
313/313 [==============================] - 1s 4ms/step
22.693932868164858

Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_21 (Flatten)        (None, 784)               0

 dense_37 (Dense)            (None, 200)               157000

 dense_38 (Dense)            (None, 1568)              315168

 dense_39 (Dense)            (None, 784)               1230096

 reshape_11 (Reshape)        (None, 28, 28)            0

=================================================================
Total params: 1,702,264
Trainable params: 1,702,264
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
938/938 [==============================] - 16s 16ms/step - loss: 0.0164 -
accuracy: 0.1681
```

```
Epoch 2/10
938/938 [==============================] - 15s 16ms/step - loss: 0.0071 -
accuracy: 0.2622
Epoch 3/10
938/938 [==============================] - 15s 16ms/step - loss: 0.0055 -
accuracy: 0.3019
Epoch 4/10
938/938 [==============================] - 17s 19ms/step - loss: 0.0048 -
accuracy: 0.3227
Epoch 5/10
938/938 [==============================] - 16s 17ms/step - loss: 0.0043 -
accuracy: 0.3387
Epoch 6/10
938/938 [==============================] - 16s 17ms/step - loss: 0.0040 -
accuracy: 0.3490
Epoch 7/10
938/938 [==============================] - 16s 17ms/step - loss: 0.0038 -
accuracy: 0.3572
Epoch 8/10
938/938 [==============================] - 16s 17ms/step - loss: 0.0036 -
accuracy: 0.3628
Epoch 9/10
938/938 [==============================] - 16s 17ms/step - loss: 0.0035 -
accuracy: 0.3688
Epoch 10/10
938/938 [==============================] - 17s 18ms/step - loss: 0.0034 -
accuracy: 0.3727
313/313 [==============================] - 2s 6ms/step
25.413642207878297
```

2.b Display the figure

	Image 1	**Image 2**	**Image 3**	**Image 4**	**Image 5**
Original Test Image					
P = 10					

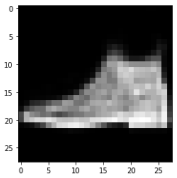
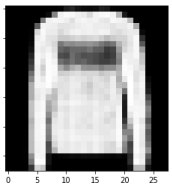
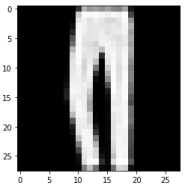
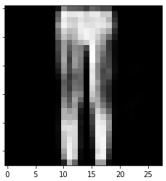
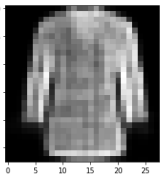
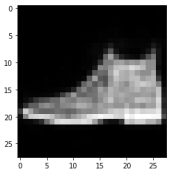
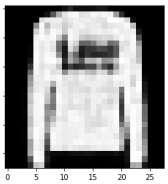
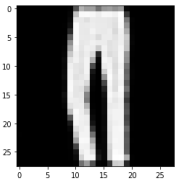
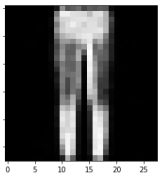
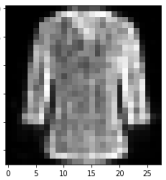
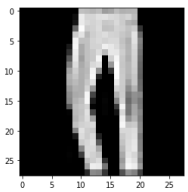
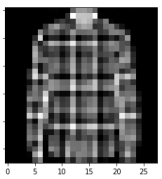
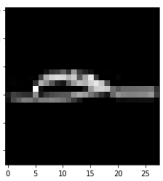
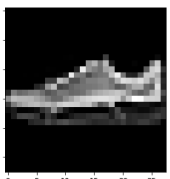
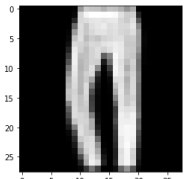
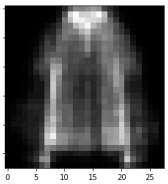
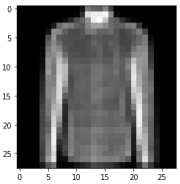
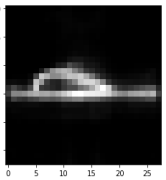
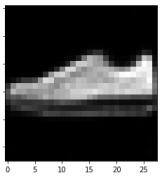
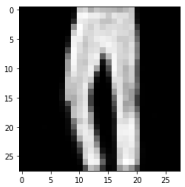
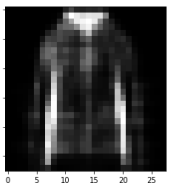
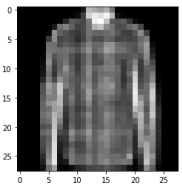
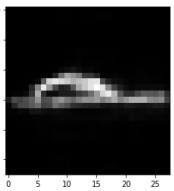
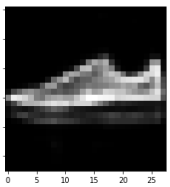
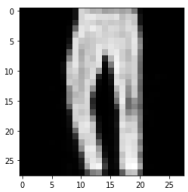
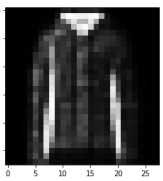
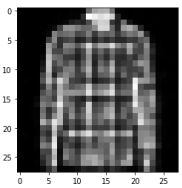
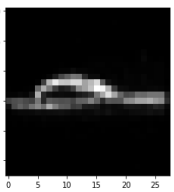
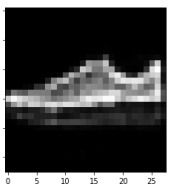
P = 50					
P = 200					

	Image 6	**Image 7**	**Image 8**	**Image 9**	**Image 10**
Original Test Image					
P = 10					
P = 50					
P = 200					

PART 1 Questions:

- Explain the source code: how to load dataset, build network layers, specify the loss function, batch size and epoch number, how to train the model, perform prediction on the test set, and generate the final class label for the test set?
    - To load the dataset, we called “models.Sequential”
    - To build the network layers, we used “model.add()” sequentially to add layers; the first layer flattened the input image by providing argument “Flatten(input_shape = (28, 28)); the middle layers were dense/fully connected layers by providing argument “Dense(P, activation = ‘action_function_need’)”; the last layer reshapes to convert the 1D vector back to a 28x28 2D image by providing argument “Reshape(28,28)
    - To specify the loss function, batch size, and epoch number we used “model.compile(loss = 'mean_squared_error', optimizer = 'adam', metrics = ['accuracy'])” and “model.fit(x_train, x_train, epochs = 10, batch_size = 64)”
    - To train the model we did ““model.compile(loss = 'mean_squared_error', optimizer = 'adam', metrics = ['accuracy'])” and “model.fit(x_train, x_train, epochs = 10, batch_size = 64)”.
    - To perform the prediction on the test set “predicted_probability = model.predict(x_test)”
    - To generate the final class label for the test set we “y_test_hat = np.argmax(predicted_probability, axis=1)”
- Explain the confusion matrix. What do the diagonal elements of the matrix represent, and what do the off- diagonal elements of the matrix represent? From these results, do you think your neural network works well for this image recognition task?
    - The confusion matrix represents how accurate our prediction model is by showing us what our model predicted(horizontal) and compare it to the actual data(vertical). The diagonal elements represent the correctly predicted data. Yes the model works well as we have an accuracy of 87% so it is good.

PART 2 Questions:

- Explain the source code: how to build network layers, specify the loss function, batch size and epoch number, how to calculate the average PSNR for the test set?
    - In order to build the networks we first create a sequential model. After that we flatten the 28 x 28 matrix into a 1 x 784 vector with all the data. We then use the dense function in order to compress, then expand it. Then we

compress it back to create the output layer and then fit the model. The loss function is mse, with 10 epochs and batch size of 64. In order to calculate the average PSNR, we run through all the images in a for loop then subtract the differences in the predicted and actual. Then we calculate the PSNR for that image then add it all up and divide by the length.

- Which layers of the network belong to the encoder (compression), which layers of the network belong to the decoder (decompression)?
    - The 2nd and 4th layers are compression and the 3rd layer is decompression.
- For different P values, what are the quality of the decoded images, and why?
    - The quality of the pictures gets closer to the original image the bigger the p value and it is because we have more nodes to use so we can store more data.