Coursework 2: Neural Networks

This coursework covers the topics covered in class regarding neural networks for image classification.

This coursework includes both coding questions as well as written ones. Please upload the notebook, which contains your code, results and answers as a pdf file onto Cate.

Dependencies: If you work on a college computer in the Computing Lab, where Ubuntu 18.04 is installed by default, you can use the following virtual environment for your work, where relevant Python packages are already installed.

```
source
/vol/bitbucket/wbai/virt/computer_vision_ubuntu18.04/bin/activate
```

Alternatively, you can use pip, pip3 or anaconda etc to install Python packages.

Note 1: please read the both the text and code comment in this notebook to get an idea what you are supposed to implement.

Note 2: If you are using the virtual environment in the Computing Lab, please run the following command in the command line before opening jupyter-notebook and importing tensorflow. This will tell tensorflow where the Nvidia CUDA libariries are.

```
export
LD_LIBRARY_PATH=/vol/cuda/9.0.176/lib64/:"${LD_LIBRARY_PATH}}"
```

In [1]:
```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

```
Using TensorFlow backend.
```

Question 1 (20 points)

Throughout this coursework you will be working with the Fashion-MNIST dataset. If you are interested, you may find relevant information regarding the dataset in this paper.

[1] Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. Han Xiao, Kashif Rasul, Roland Vollgraf. arXiv:1708.07747 (https://arxiv.org/abs/1708.07747)

Be sure that you have the following files in your working directory: data.tar.gz and reader.py. Loading the data can be done as follows:

```
from reader import get_images
(x_train, y_train), (x_test, y_test) = get_images()
```

The dataset is already split into a set of 60,000 training images and a set of 10,000 test images. The images are of size 28x28 pixels and stored as 784-D vector. So if you would like to visualise the images, you need to reshape the array.

There are in total 10 label classes, which are:

- 0: T-shirt/top
- 1: Trousers
- 2: Pullover
- 3: Dress
- 4: Coat
- 5: Sandal
- 6: Shirt
- 7: Sneaker
- 8: Bag
- 9: Ankle boot

1.1 Load data (6 points)

Load the dataset and print the dimensions of the training set and the test set.

In [2]:
```
from reader import get_images
(x_train, y_train), (x_test, y_test) = get_images()

print("dimensions of x_train: ", x_train.shape)
print("dimensions of y_train: ", y_train.shape)
print("dimensions of x_test: ", x_test.shape)
print("dimensions of y_test: ", y_test.shape)
```

```
dimensions of x_train:  (60000, 784)
dimensions of y_train:  (60000,)
dimensions of x_test:  (10000, 784)
dimensions of y_test:  (10000,)
```

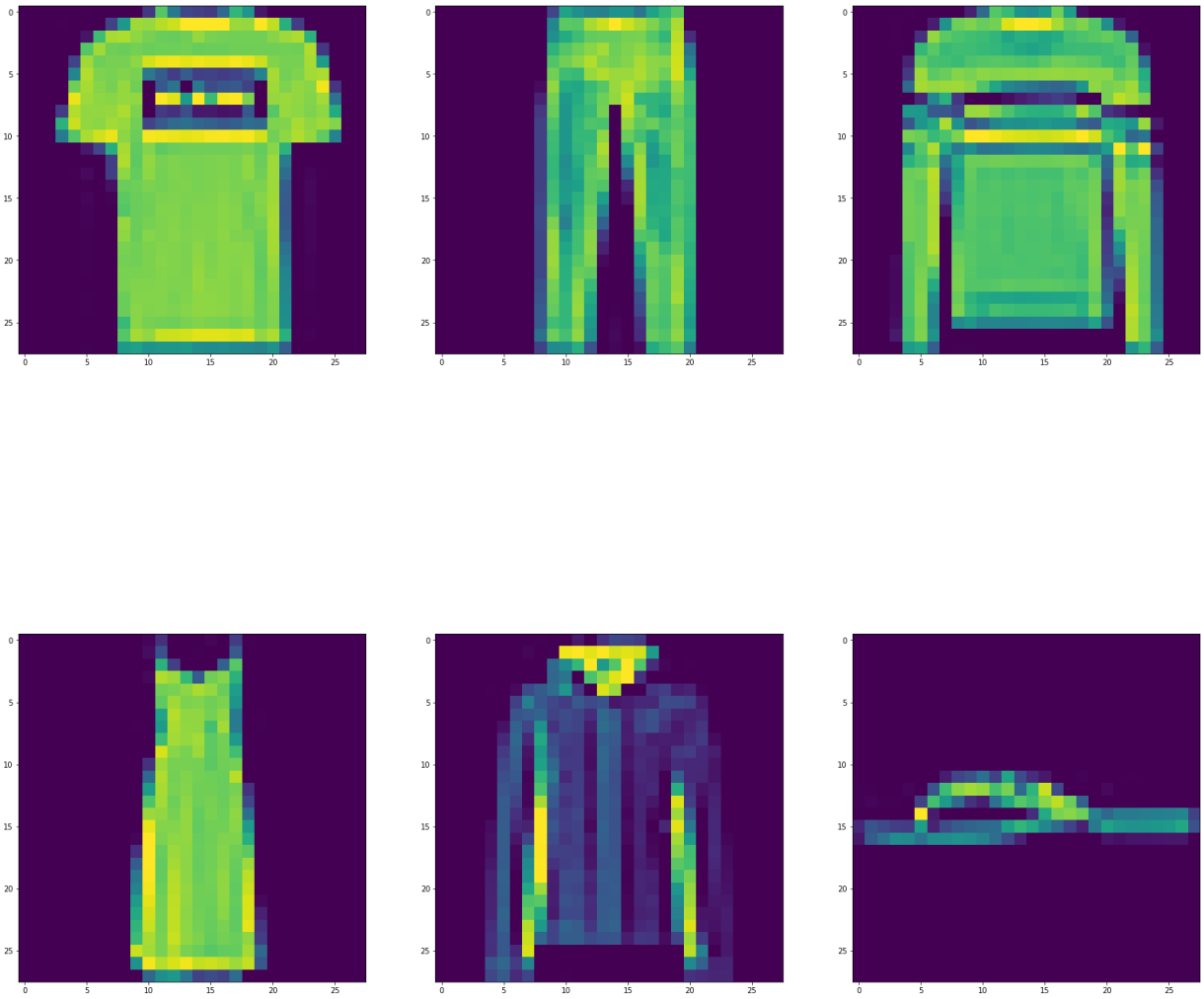
1.2 Visualize data (6 points)

Visualise 3 training images (T-shirt, trousers and pullover) and 3 test images (dress, coat and sandal).

In [3]:
```
print('-----------------3 trainning images-------------------------')
fig=plt.figure(figsize=(28, 28))
columns = 3
rows = 2
for i in range(len(x_train)):
    if y_train[i] == 0:
        fig.add_subplot(rows, columns, 1)
        plt.imshow(x_train[i].reshape(28,28))
        print('label = ', y_train[i],end =" ")
        break

for i in range(len(x_train)):
    if y_train[i] == 1:
        fig.add_subplot(rows, columns, 2)
        plt.imshow(x_train[i].reshape(28,28))
        print('label = ', y_train[i],end =" ")
        break

for i in range(len(x_train)):
    if y_train[i] == 2:
        fig.add_subplot(rows, columns, 3)
        plt.imshow(x_train[i].reshape(28,28))
        print('label = ', y_train[i])
        break

print('-----------------3 test images-------------------------')
for i in range(len(x_test)):
    if y_test[i] == 3:
        fig.add_subplot(rows, columns, 4)
        plt.imshow(x_test[i].reshape(28,28))
        print('label = ', y_test[i],end =" ")
        break
```

```
for i in range(len(x_test)):
    if y_test[i] == 4:
        fig.add_subplot(rows, columns, 5)
        plt.imshow(x_test[i].reshape(28,28))
        print('label = ', y_test[i],end =" ")
        break

for i in range(len(x_test)):
    if y_test[i] == 5:
        fig.add_subplot(rows, columns, 6)
        plt.imshow(x_test[i].reshape(28,28))
        print('label = ', y_test[i],end =" ")
        break
plt.show()
```

```
-----------------3 trainning images--------------------------
label =  0 label =  1 label =  2
-----------------3 test images--------------------------
label =  3 label =  4 label =  5
```



1.3 Data balance (4 points)

Print out the number of training samples for each class.

In [4]:
```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
number_sample = [0]*10
print(number_sample)
for i in y_train:
    number_sample[i] += 1
for i in range(len(number_sample)):
    print(class_names[i],number_sample[i])
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
T-shirt/top 6000
Trouser 6000
Pullover 6000
Dress 6000
Coat 6000
Sandal 6000
Shirt 6000
Sneaker 6000
Bag 6000
Ankle boot 6000
```

1.4 Discussion (4 points)

Is the dataset balanced? What would happen if the dataset is not balanced in the context of image classification?

Yes, the data is balanced, if the dataset is not balanced, thoes trainning data with large amount might have higher accuracy than others, on the other hand, those trainning data with smaller amount tend to have lower accuracy. Because the algorithm tend to faver the large trainning data.

In [ ]:
```
```

Question 2 (40 points)

Build a neural network and train it with the Fashion-MNIST dataset. Here, we use the keras library, which is a high-level neural network library built upon tensorflow.

In [5]:
```
# Convert the label class into a one-hot representation
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
print(y_train.shape)
print(y_test.shape)
```

```
(60000, 10)
(10000, 10)
```

2.1 Build a multi-layer perceptron, also known as multi-layer fully connected network. You need to define the layers, the loss function, the optimiser and evaluation metric. (30 points)

In [6]:
```
model = keras.Sequential()
model.add(Dense(32, activation='relu', input_dim=784))
# model.add(Dropout(0.5))
model.add(Dense(16, activation='relu'))

model.add(Dense(8, activation='relu'))

model.add(keras.layers.Dense(10, activation='softmax'))
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_2 (Dense)              (None, 16)                528
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 136
_________________________________________________________________
dense_4 (Dense)              (None, 10)                90
=================================================================
Total params: 25,874
Trainable params: 25,874
Non-trainable params: 0
_________________________________________________________________
None
```

2.2 Define the optimisation parameters including the batch size and the number of epochs and then run the optimiser. (10 points)

We have tested that for an appropriate network architecture, on a personal laptop and with only CPU, it takes about a few seconds per epoch to train the network. For 100 epochs, it takes about a coffee break's time to finish the training. If you run it on a powerful GPU, it would be even much faster.

In [7]:
```
batch_size = 32
epochs = 10

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs)
```

```
Epoch 1/10
60000/60000 [==============================] - 8s 140us/step - loss:
0.2893 - acc: 0.9125
Epoch 2/10
60000/60000 [==============================] - 8s 127us/step - loss:
0.1714 - acc: 0.9282
Epoch 3/10
60000/60000 [==============================] - 8s 127us/step - loss:
0.1589 - acc: 0.9331
Epoch 4/10
60000/60000 [==============================] - 8s 128us/step - loss:
0.1383 - acc: 0.9467
Epoch 5/10
60000/60000 [==============================] - 9s 158us/step - loss:
0.1261 - acc: 0.9525
Epoch 6/10
60000/60000 [==============================] - 8s 135us/step - loss:
0.1205 - acc: 0.9560
Epoch 7/10
60000/60000 [==============================] - 8s 128us/step - loss:
0.1151 - acc: 0.9582
Epoch 8/10
60000/60000 [==============================] - 8s 128us/step - loss:
0.1148 - acc: 0.9600
Epoch 9/10
60000/60000 [==============================] - 8s 128us/step - loss:
0.1098 - acc: 0.9622
Epoch 10/10
60000/60000 [==============================] - 8s 128us/step - loss:
0.1071 - acc: 0.9633
```

Out[7]: <keras.callbacks.History at 0xb392cd860>

Question 3 (20 points)

Evaluate the performance of your network with the test data. Visualize the performance using appropriate metrics and graphs (eg. confusion matrix). Comment on your per class performance and how it could be better.

In [8]:
```
# This function is provided for you to display the confusion matrix.
# For more information about the confusion matrix, you can read at
# https://en.wikipedia.org/wiki/Confusion_matrix
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusi
    """
        This function prints and plots the confusion matrix.

        cm: confusion matrix, default to be np.int32 data type
        classes: a list of the class labels or class names
        normalize: normalize the matrix so that each row amounts to or
        cmap: color map
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

3.1 Evaluate the classification accuracy on the test set (10 points)

In [9]:
```
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

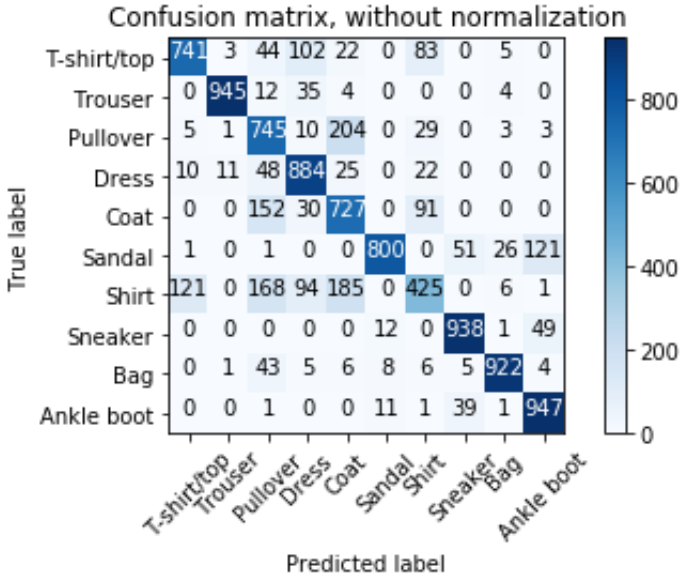
```
10000/10000 [==============================] - 0s 39us/step
Test accuracy: 0.9635600208282471
```

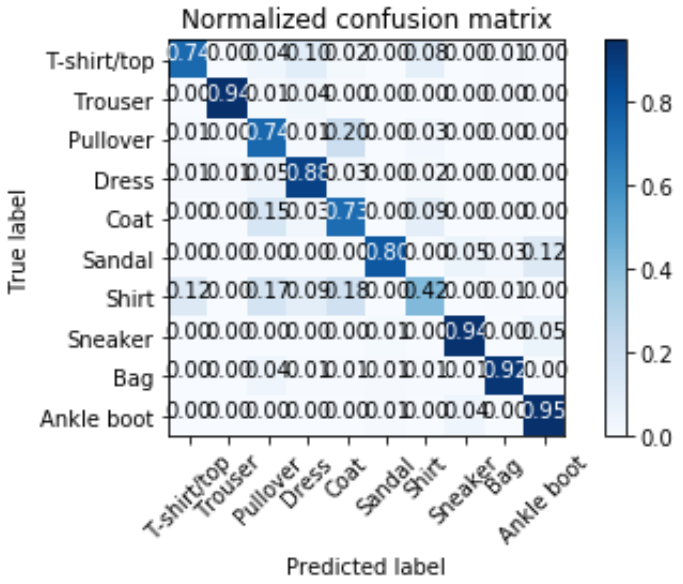
3.2 Calculate and plot the confusion matrix (10 points)

In [10]:
```
from sklearn.metrics import confusion_matrix
import itertools
# Compute confusion matrix
# print(x_test.shape)
y_pred = model.predict(x_test)

print(y_pred.shape)
print(y_test.shape)

decoded1 = np.argmax(y_test, axis=1)
print(decoded1.shape)
decoded2 = np.argmax(y_pred, axis=1)
print(decoded2.shape)


cnf_matrix = confusion_matrix(decoded1, decoded2)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')
```

```
(10000, 10)
(10000, 10)
(10000,)
(10000,)
Confusion matrix, without normalization
[[741   3  44 102  22   0  83   0   5   0]
 [  0 945  12  35   4   0   0   0   4   0]
 [  5   1 745  10 204   0  29   0   3   3]
 [ 10  11  48 884  25   0  22   0   0   0]
 [  0   0 152  30 727   0  91   0   0   0]
 [  1   0   1   0   0 800   0  51  26 121]
 [121   0 168  94 185   0 425   0   6   1]
 [  0   0   0   0   0  12   0 938   1  49]
 [  0   1  43   5   6   8   6   5 922   4]]
```

```
 [   0    0    1    0    0   11    1   39    1  947]]
Normalized confusion matrix
[[0.74 0.   0.04 0.1  0.02 0.   0.08 0.   0.01 0.  ]
 [0.   0.94 0.01 0.04 0.   0.   0.   0.   0.   0.  ]
 [0.01 0.   0.74 0.01 0.2  0.   0.03 0.   0.   0.  ]
 [0.01 0.01 0.05 0.88 0.03 0.   0.02 0.   0.   0.  ]
 [0.   0.   0.15 0.03 0.73 0.   0.09 0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.8  0.   0.05 0.03 0.12]
 [0.12 0.   0.17 0.09 0.18 0.   0.42 0.   0.01 0.  ]
 [0.   0.   0.   0.   0.   0.01 0.   0.94 0.   0.05]
 [0.   0.   0.04 0.01 0.01 0.01 0.01 0.01 0.92 0.  ]
 [0.   0.   0.   0.   0.   0.01 0.   0.04 0.   0.95]]
```





Question 4 (20 points)

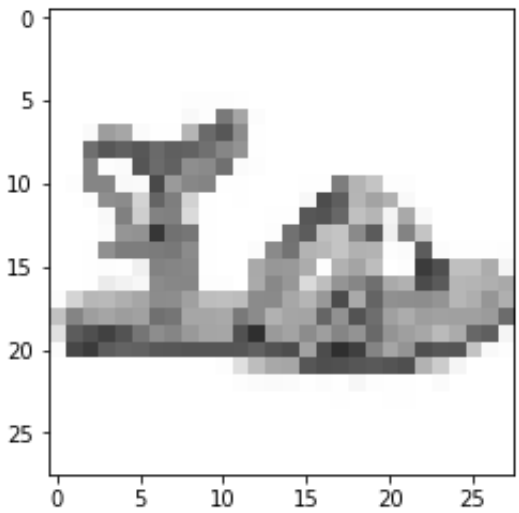
Take two photos, one of your clothes or shoes that belongs to one of 10 classes, the other that does not belong to any class.

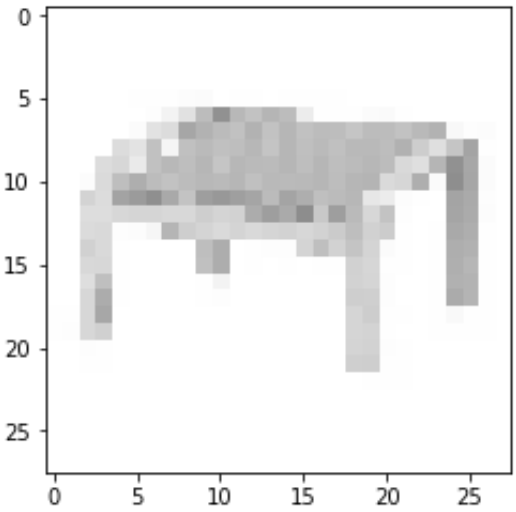
Use either Python or other software (Photoshop, Gimp, or any image editer) to convert the photos into grayscale, crop the region of interest and reshape into the size of 28x28.

4.1 Load and visualise your own images (6 points)

In [16]:

```
import PIL
import matplotlib.image as mpimg

# load the sandal image and convert into grey scale
sample1 = PIL.Image.open('sandals.jpg').convert('L')
sample1 = sample1.resize((28, 28))    # resize it into 28*28

imgplot = plt.imshow(sample1)
plt.show()

sample2 = PIL.Image.open('chair.jpg').convert('L')
sample2 = sample2.resize((28, 28))    # best down-sizing filter

imgplot = plt.imshow(sample2)
plt.show()


test1 = np.array(sample1)

test2 = np.array(sample2)
testList = np.array([test1,test2])
print(testList.shape)
```

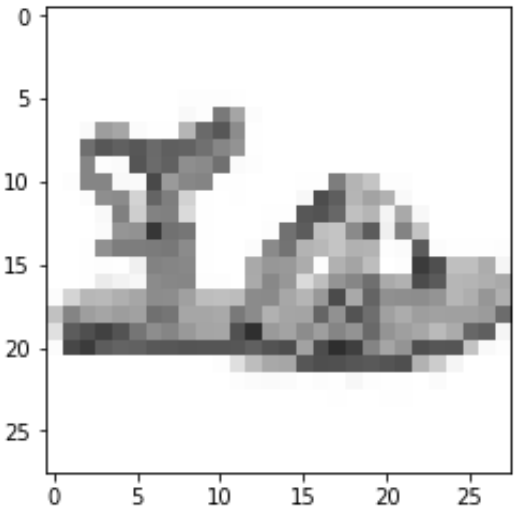




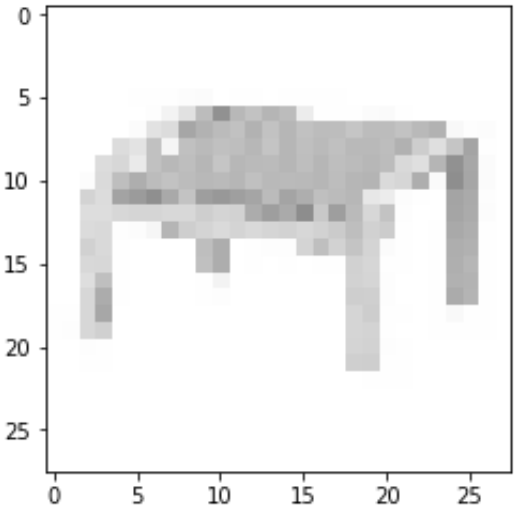

```
(2, 28, 28)
```

4.2 Test your network on the two images and show the classification results (10 points)

In [17]:
```
predictions_one_hot = model.predict(testList.reshape(2,784))
predictions = np.argmax(predictions_one_hot, axis=1)
imgplot = plt.imshow(sample1)
plt.show()
print("The first graph I predict is label ", predictions[0], class_nam
imgplot = plt.imshow(sample2)
plt.show()
print("The second graph I predict is label ", predictions[1], class_na
```




```
The first graph I predict is label  5 Sandal
```




```
The second graph I predict is label  8 Bag
```

4.3 Discuss the classification results and provide one method to improve real life performance of the network (4 points)

The first image is a sandal when we use it to test the model. The fucntion gives the right prediction. The second image is a chair. It somehow looks like a bag so the model.predict assume it is a bag.

To imporve the real life performance:

1.we need to have less noise in the background of the image as possible. Possible make the image pure black or white.

2.having more suitable amount of the layers

3.having more balanced data set will increase the accurate rate.

In [ ]:

5. Survey

How long did the coursework take you to solve?

3 days

Type *Markdown* and LaTeX: α^2