```
"""
Code, apart from the confusion matrix,taken from:
 https://www.tensorflow.org/tutorials/keras/basic_classification#
"""

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import itertools

# Load minst fashion data set
"""Minst fashion data set  is a collection of 70k images of 10
different
fashion items. It is loaded as training and test images and labels
(60K training
images and 10K test images).

0    T-shirt/top
1    Trouser
2    Pullover
3    Dress
4    Coat
5    Sandal
6    Shirt
7    Sneaker
8    Bag
9    Ankle boot
"""
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = \
    fashion_mnist.load_data()

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Show an example image (an ankle boot)
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()

# Scale images to values 0-1 (currently 0-255)
train_images = train_images / 255.0
test_images = test_images / 255.0

# Plot first 25 images with labels
```

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()

## BUILD MODEL

# Set up neural network layers
"""The first layer flattess the 28x28 image to a 1D array (784
pixels).
The second layer is a fully connected (dense) layer of 128
nodes/neurones.
The last layer is a 10 node softmax layer, giving probability of
each class.
Softmax adjusts probabilities so that they total 1."""

model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)])

# Compile the model
"""Optimizer: how model corrects itself and learns.
Loss function: How accurate the model is.
Metrics: How to monitor performance of model"""

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model (epochs is the number of times the training data
is applied)
model.fit(train_images, train_labels, epochs=5)

# Evaluate accuracy
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)

# APPLY MODEL

# Make predictions
predictions = model.predict(test_images)
print ('\nClass propbabilities for test image 0')
print (predictions[0])
print ('\nPrdicted class for test image 0:',
np.argmax(predictions[0]))
print ('Actual classification for test image 0:', test_labels[0])
```

```
# Plot image and predictions
def plot_image(i, predictions_array, true_label, img):
  predictions_array, true_label, img = predictions_array[i],
true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}%
({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)

def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array[i],
true_label[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array,
color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')

# Plot images and graph for  selected images
# Blue bars shows actual classification
# Red bar shows an incorrect classificiation
num_rows = 6
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions, test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions, test_labels)
plt.show()

# SHOW CONFUSION MATRIX

def plot_confusion_matrix(cm, classes,
```

```
                              normalize=False,
                              title='Confusion matrix',
                              cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        cm = cm * 100
        print("\nNormalized confusion matrix")
    else:
        print('\nConfusion matrix, without normalization')
    print(cm)
    print ()

    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 = '.0f' 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.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

# Compute confusion matrix
y_pred = np.argmax(predictions, axis=1)
cnf_matrix = confusion_matrix(test_labels, y_pred)
np.set_printoptions(precision=2) # set NumPy to 2 decimal places

# 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')
```

```
# Making a prediction of a single image
"""tf.keras models are optimized to make predictions on a batch,
or collection,
of examples at once. So even though we're using a single image, we
need to add
it to a list:"""

# Grab an example image
img = test_images[0]
# Add the image to a batch where it's the only member.
img = (np.expand_dims(img,0))
# Make prediction
predictions_single = model.predict(img)
# Plot results
plot_value_array(0, predictions_single, test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
plt.show()
```

```
# DEALINGS IN THE SOFTWARE.
```