A MNIST-like fashion product database

In this, we classify the images into respective classes given in the dataset. We use a Neural Net and a Deep Neural Net in Keras to solve this and check the accuracy scores.

Load tensorflow

```
import tensorflow as tf

tf.set_random_seed(42)
```

```
tf.__version__
```

```
'1.14.0'
```

Collect Data

```
import keras
```

```
Using TensorFlow backend.
```

```
(trainX, trainY), (testX, testY) = keras.datasets.fashion_mnist.load_data()
```

```
print(testY[0:5])
```

```
[9 2 1 1 6]
```

Convert both training and testing labels into one-hot vectors.

Hint: check **tf.keras.utils.to_categorical()**

```
trainY = tf.keras.utils.to_categorical(trainY, num_classes=10)
testY = tf.keras.utils.to_categorical(testY, num_classes=10)
```

```
import numpy
numpy.unique(trainY)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

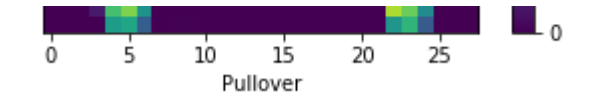
```
print(trainY.shape)
print('First 5 examples now are: ', trainY[0:5])
```

```
(60000,)
('First 5 examples now are: ', array([9, 0, 0, 3, 0], dtype=uint8))
```

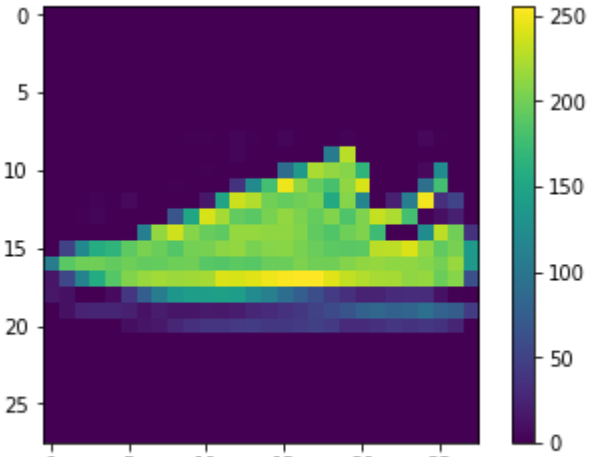
Visualize the data

Plot first 10 images in the triaining set and their labels.

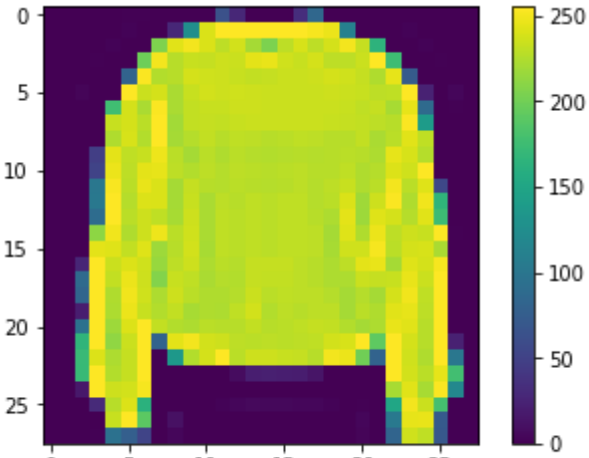
```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']


import matplotlib.pyplot as plt

for i in range(0,10):
  plt.imshow(trainX[i])
  plt.colorbar()
  plt.grid(False)
  plt.xlabel(class_names[trainY[i]])
  plt.show()
```



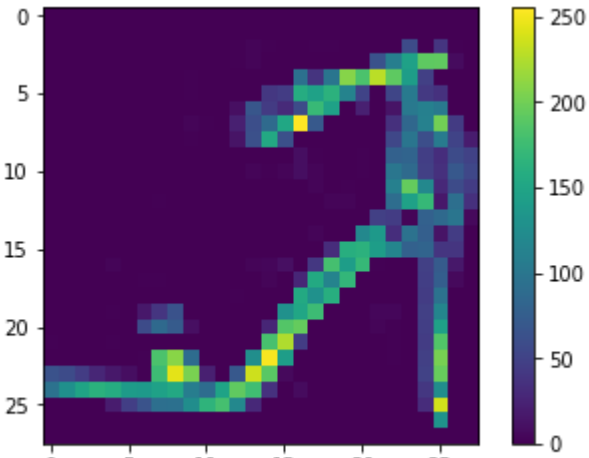
Pullover



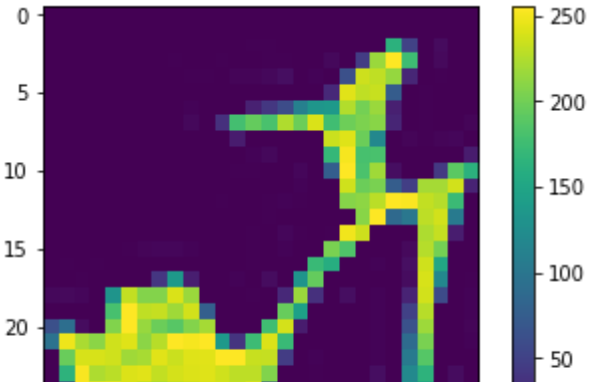
Sneaker

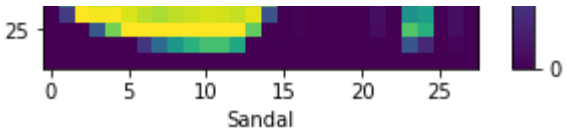


Pullover



Sandal




25
0
5
10
15
20
25
Sandal
0

```
trainX.shape
```

```
(60000, 28, 28)
```

Build a neural Network with a cross entropy loss function and sgd optimizer in Keras. The output layer with 10 neurons as we have 10 classes.

```
#Initialize Sequential Graph (model)
model = tf.keras.Sequential()
#keras.layers.Flatten(input_shape=(28, 28)),
#Add Dense layer for prediction - Keras declares weights and bias automatically
model.add(tf.keras.layers.Reshape((784,),input_shape=(28,28,)))

model.add(tf.keras.layers.Dense(100, input_shape=(28,)))

#Normalize the data
#model.add(tf.keras.layers.BatchNormalization())
#Add 1st hidden layer
#model.add(tf.keras.layers.Dense(200, activation='sigmoid'))
#Add OUTPUT layer
model.add(tf.keras.layers.Dense(10, activation='softmax'))
#Compile the model - add Loss and Gradient Descent optimizer
model.compile(optimizer='sgd', loss='mse',metrics=['accuracy'])
```

Execute the model using model.fit()

```
model.fit(trainX, trainY, epochs=100)
```

```
60000/60000 [==============================] - 3s 51us/sample - loss: 0.1628 - acc:
Epoch 69/100
60000/60000 [==============================] - 3s 51us/sample - loss: 0.1624 - acc:
Epoch 70/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1629 - acc:
Epoch 71/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1625 - acc:
Epoch 72/100
60000/60000 [==============================] - 3s 53us/sample - loss: 0.1622 - acc:
Epoch 73/100
60000/60000 [==============================] - 3s 53us/sample - loss: 0.1625 - acc:
Epoch 74/100
60000/60000 [==============================] - 3s 54us/sample - loss: 0.1634 - acc:
Epoch 75/100
60000/60000 [==============================] - 3s 54us/sample - loss: 0.1620 - acc:
Epoch 76/100
60000/60000 [==============================] - 3s 52us/sample - loss: 0.1622 - acc:
Epoch 77/100
60000/60000 [==============================] - 3s 51us/sample - loss: 0.1622 - acc:
Epoch 78/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1620 - acc:
Epoch 79/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1495 - acc:
Epoch 80/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1442 - acc:
Epoch 81/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1436 - acc:
Epoch 82/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1433 - acc:
Epoch 83/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1427 - acc:
Epoch 84/100
60000/60000 [==============================] - 3s 51us/sample - loss: 0.1425 - acc:
Epoch 85/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1423 - acc:
Epoch 86/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1424 - acc:
Epoch 87/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1427 - acc:
Epoch 88/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1423 - acc:
Epoch 89/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1423 - acc:
Epoch 90/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1422 - acc:
Epoch 91/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1428 - acc:
Epoch 92/100
60000/60000 [==============================] - 3s 51us/sample - loss: 0.1426 - acc:
Epoch 93/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1427 - acc:
Epoch 94/100
60000/60000 [==============================] - 3s 49us/sample - loss: 0.1424 - acc:
Epoch 95/100
60000/60000 [==============================] - 3s 49us/sample - loss: 0.1422 - acc:
Epoch 96/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1421 - acc:
Epoch 97/100
60000/60000 [==============================] - 3s 49us/sample - loss: 0.1420 - acc:
Epoch 98/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1419 - acc:
Epoch 99/100
```

```
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1421 - acc:
Epoch 100/100
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1425 - acc:
<tensorflow.python.keras.callbacks.History at 0x7fd9b2e900d0>
```

```
# evaluate the model
model.evaluate(testX, testY, verbose=0)
```

In the above Neural Network model add Batch Normalization layer after the input layer and repeat the steps.

```
#Initialize Sequential Graph (model)
model = tf.keras.Sequential()
#keras.layers.Flatten(input_shape=(28, 28)),
#Add Dense layer for prediction - Keras declares weights and bias automatically
model.add(tf.keras.layers.Reshape((784,),input_shape=(28,28,)))

model.add(tf.keras.layers.Dense(1, input_shape=(28,)))

#Normalize the data
model.add(tf.keras.layers.BatchNormalization())
#Add 1st hidden layer
#model.add(tf.keras.layers.Dense(200, activation='sigmoid'))
#Add OUTPUT layer
model.add(tf.keras.layers.Dense(10, activation='softmax'))
#Compile the model - add Loss and Gradient Descent optimizer
model.compile(optimizer='sgd', loss='mse', metrics=['accuracy'])
```

Execute the model

```
model.fit(trainX, trainY, epochs=100,batch_size=100)
```

```
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0766 - acc:
Epoch 69/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0765 - acc:
Epoch 70/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0765 - acc:
Epoch 71/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0764 - acc:
Epoch 72/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0763 - acc:
Epoch 73/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0763 - acc:
Epoch 74/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0762 - acc:
Epoch 75/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0762 - acc:
Epoch 76/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0761 - acc:
Epoch 77/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0760 - acc:
Epoch 78/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0760 - acc:
Epoch 79/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0759 - acc:
Epoch 80/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0758 - acc:
Epoch 81/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0758 - acc:
Epoch 82/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0757 - acc:
Epoch 83/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0757 - acc:
Epoch 84/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0756 - acc:
Epoch 85/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0755 - acc:
Epoch 86/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0755 - acc:
Epoch 87/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0754 - acc:
Epoch 88/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0753 - acc:
Epoch 89/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0753 - acc:
Epoch 90/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0752 - acc:
Epoch 91/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0751 - acc:
Epoch 92/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0751 - acc:
Epoch 93/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0751 - acc:
Epoch 94/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0750 - acc:
Epoch 95/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0749 - acc:
Epoch 96/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0749 - acc:
Epoch 97/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0748 - acc:
Epoch 98/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0748 - acc:
Epoch 99/100
```

```
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0747 - acc:
Epoch 100/100
60000/60000 [==============================] - 1s 14us/sample - loss: 0.0747 - acc:
<tensorflow.python.keras.callbacks.History at 0x7fd9b1b17d90>
```

```
pred = model.predict(testX[1:20])
# evaluate the model
model.evaluate(testX, testY, verbose=0)
```

```
[0.14197733678817748, 0.2901]
```

Customize the learning rate to 0.001 in sgd optimizer and run the model

```
from keras.optimizers import SGD
#Initialize Sequential Graph (model)
model = tf.keras.Sequential()
#keras.layers.Flatten(input_shape=(28, 28)),
#Add Dense layer for prediction - Keras declares weights and bias automatically
model.add(tf.keras.layers.Reshape((784,),input_shape=(28,28,)))

model.add(tf.keras.layers.Dense(1, input_shape=(28,)))

#Normalize the data
model.add(tf.keras.layers.BatchNormalization())
#Add 1st hidden layer
#model.add(tf.keras.layers.Dense(200, activation='sigmoid'))
#Add OUTPUT layer
model.add(tf.keras.layers.Dense(10, activation='softmax'))
#Compile the model - add Loss and Gradient Descent optimizer
epochs = 50
learning_rate = 0.001
decay_rate = learning_rate / epochs
momentum = 0.5
sgd = SGD(lr=learning_rate, momentum=momentum, decay=decay_rate, nesterov=False)
model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])
```

```
model.fit(trainX, trainY, epochs=100,batch_size=100)
```