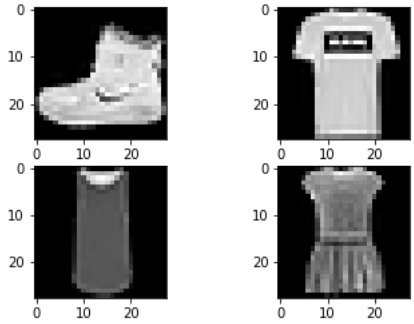
Test tensor flow with FASHION_MNIST data set and repeat what you did with step 3 for MNIST data set in HW1.

(If you have installed keras in your computer, you should have a folder .../keras/datasets/ where you can find a file mnist.py. Now you need to download at CANVAS from week 3 the file fashion_mnist.py to the same folder .../keras/datasets/ Then, you can use from keras.datasets import fashion_mnist as you did with mnist in HW1.)

```
# Plot ad hoc mnist instances
from keras.datasets import fashion_mnist
import matplotlib.pyplot as plt
# load (downloaded if needed) the MNIST dataset
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
# plot 4 images as gray scale
plt.subplot(221)
plt.imshow(X_train[0], cmap=plt.get_cmap('gray'))
plt.subplot(222)
plt.imshow(X_train[1], cmap=plt.get_cmap('gray'))
plt.subplot(223)
plt.imshow(X_train[2], cmap=plt.get_cmap('gray'))
plt.subplot(224)
plt.imshow(X_train[3], cmap=plt.get_cmap('gray'))
# show the plot
plt.show()
import numpy as np
from keras.datasets import fashion_mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.utils import np_utils

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# load data
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()


# flatten 28*28 images to a 784 vector for each image
num_pixels = X_train.shape[1] * X_train.shape[2]
X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32')
X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')

# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_test = X_test / 255

# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]


# define baseline model
def baseline_model():
# create model
 model = Sequential()
 model.add(Dense(num_pixels, input_dim=num_pixels, kernel_initializer='normal', activation='relu'))
 model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
# Compile model
 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
 return model


# build the model
model = baseline_model()
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200, verbose=2)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Baseline Error: %.2f%%" % (100-scores[1]*100))
```



```
Epoch 1/10
300/300 - 7s - loss: 0.5062 - accuracy: 0.8247 - val_loss: 0.4297 - val_accuracy: 0.8447 - 7s/epoch - 23ms/step
Epoch 2/10
300/300 - 5s - loss: 0.3745 - accuracy: 0.8654 - val_loss: 0.3831 - val_accuracy: 0.8633 - 5s/epoch - 18ms/step
Epoch 3/10
300/300 - 6s - loss: 0.3320 - accuracy: 0.8795 - val_loss: 0.3582 - val_accuracy: 0.8715 - 6s/epoch - 21ms/step
Epoch 4/10
300/300 - 5s - loss: 0.3042 - accuracy: 0.8885 - val_loss: 0.3484 - val_accuracy: 0.8745 - 5s/epoch - 17ms/step
Epoch 5/10
300/300 - 6s - loss: 0.2827 - accuracy: 0.8973 - val_loss: 0.3431 - val_accuracy: 0.8769 - 6s/epoch - 21ms/step
Epoch 6/10
300/300 - 5s - loss: 0.2705 - accuracy: 0.9013 - val_loss: 0.3295 - val_accuracy: 0.8835 - 5s/epoch - 17ms/step
Epoch 7/10
300/300 - 5s - loss: 0.2569 - accuracy: 0.9053 - val_loss: 0.3227 - val_accuracy: 0.8856 - 5s/epoch - 18ms/step
Epoch 8/10
300/300 - 6s - loss: 0.2439 - accuracy: 0.9102 - val_loss: 0.3219 - val_accuracy: 0.8884 - 6s/epoch - 20ms/step
Epoch 9/10
300/300 - 5s - loss: 0.2345 - accuracy: 0.9131 - val_loss: 0.3268 - val_accuracy: 0.8835 - 5s/epoch - 17ms/step
Epoch 10/10
300/300 - 6s - loss: 0.2252 - accuracy: 0.9171 - val_loss: 0.3140 - val_accuracy: 0.8909 - 6s/epoch - 21ms/step
Baseline Error: 10.91%
```