**Run the Cell to import the packages and data**

In [2]:
```
from keras.datasets import fashion_mnist
from keras.utils import to_categorical
import numpy as np
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-2-b10d905e6131> in <module>
----> 1 from keras.datasets import fashion_mnist
      2 from keras.utils import to_categorical
      3 import numpy as np

~/.local/lib/python3.5/site-packages/keras/__init__.py in <module>
     19 """
     20 # pylint: disable=unused-import
---> 21 from tensorflow.python import tf2
     22 from keras import distribute
     23

ImportError: No module named 'tensorflow'
```

**Data Loading**

Run the below cells to load the data

In [3]:
```
# load dataset
(trainX, trainy), (testX, testy) = fashion_mnist.load_data()
# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainy), (testX, testY) = fashion_mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainy = to_categorical(trainy)
    testY = to_categorical(testY)
    return trainX, trainy, testX, testY
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-dacc298f8ca2> in <module>
      1 # load dataset
----> 2 (trainX, trainy), (testX, testy) = fashion_mnist.load_data()
      3 # load train and test dataset
      4 def load_dataset():
      5     # load dataset

NameError: name 'fashion_mnist' is not defined
```

**Subset Generation**

- Perform data split with **StratifiedShuffleSplit** with following parameters
  - test_size = 0.08
  - random_state = seed
- Perform train test split with **StratifiedShuffleSplit** with following parameters
  - test_size = 0.3
  - random_state = seed

In [ ]:
```
seed=9

from sklearn.model_selection import StratifiedShuffleSplit

data_split = StratifiedShuffleSplit(test_size=0.08,random_state=seed)
for train_index, test_index in data_split.split(trainX, trainy):

    split_data_92, split_data_8 = trainX[train_index], trainX[test_index]

    split_label_92, split_label_8 = trainy[train_index], trainy[test_index]
train_test_split = StratifiedShuffleSplit(test_size=0.3,random_state=seed) #t
```

**Data Splitting**

- Print the shape of
  - train_data
  - train_labels
  - test_data
  - test_labels

In [ ]:
```
for train_index, test_index in train_test_split.split(split_data_8,split_labe

    train_data_70, test_data_30 = split_data_8[train_index], split_data_8[tes

    train_label_70, test_label_30 = split_label_8[train_index], split_label_8
train_data = train_data_70 #assigning to variable train_data

train_labels = train_label_70 #assigning to variable train_labels

test_data = test_data_30

test_labels = test_label_30
print('train_data : ',len(train_data))

print('train_labels : ',len(train_labels))

print('test_data : ',len(test_data))

print('test_labels : ',len(test_labels))
```

**Normalization**

- Apply the mean for **data** with following parameters
  - axis = (0,1,2)
  - keepdims=True
- Apply the square root for **data** with following parameters
  - axis = (0,1,2)
  - ddof = 1
  - keepdims = True
- Print the shape of
  - train_data
  - test_data

In [ ]:
```
# definition of normalization function

def normalize(data, eps=1e-8):

    data -= np.mean(data, axis=(0,1,2), keepdims=True)

    std = np.std(data, axis=(0,1,2) , ddof=1 , keepdims=True)

    std[std < eps] = 1.

    data /= std

    return data
train_data=train_data.astype('float64')
test_data=test_data.astype('float64')
# calling the function

train_data = normalize(train_data)

test_data = normalize(test_data)
# prints the shape of train data and test data

print('train_data: ',len(train_data))

print('test_data: ',len(test_data))
```

**ZCA Whitening**

- Print the shape of
  - train_data
  - test_data

In [ ]:
```
# Computing whitening matrix

train_data_flat = train_data.reshape(train_data.shape[0], -1).T

test_data_flat = test_data.reshape(test_data.shape[0], -1).T

print('train_data_flat: ', train_data_flat.shape)

print('test_data_flat: ',  test_data_flat.shape)



train_data_flat_t = train_data_flat.T

test_data_flat_t = test_data_flat.T
```

**Principle Component Analysis (PCA)**

- Keep **n_components** of **train_data_pca** as size of **train_data** columns and fit transform with **train_data_flat**
- Keep **n_components** of **test_data_pca** as size of **test_data** columns and fit transform with **test_data_flat**
- Print the shape of
  - train_data_pca
  - test_data_pca

In [ ]:

```
from sklearn.decomposition import PCA

# n_components specify the no.of components to keep

train_data_pca = PCA(n_components=2).fit_transform(train_data_flat)

test_data_pca =  PCA(n_components=2).fit_transform(test_data_flat)

print('train_data_pca: ',len(train_data_pca))

print('train_data_pca: ',len(test_data_pca))

train_data_pca = train_data_pca.T

test_data_pca = test_data_pca.T
```

**Singular Value Decomposition (SVD)**

Execute the below cells to perform Singular Value Decomposition

In [ ]:

```
from skimage import color
def svdFeatures(input_data):

    svdArray_input_data=[]

    size = input_data.shape[0]

    for i in range (0,size):

        img=color.rgb2gray(input_data[i])

        U, s, V = np.linalg.svd(img, full_matrices=False);

        S=[s[i] for i in range(28)]

        svdArray_input_data.append(S)

        svdMatrix_input_data=np.matrix(svdArray_input_data)

    return svdMatrix_input_data


# apply SVD for train and test data

train_data_svd=svdFeatures(train_data)

test_data_svd=svdFeatures(test_data)
print(train_data_svd.shape)
print(test_data_svd.shape)
```

**Support Vector Machine (SVM)**

- Initialize SVM classifier with following parameters
  - gamma=.001
  - probability=True
- Train the model with train_data_flat_t and train_labels
- Now predict the output with test_data_flat_t
- Evaluate the classifier with score from test_data_flat_t and test_labels
- Print the predicted score

In [ ]:
```
from sklearn import svm #Creating a svm classifier model

clf = svm.SVC(gamma=.001,probability=True) #train_data_flat_tModel training

train = clf.fit(train_data_flat_t,train_labels)
predicted=clf.predict(test_data_flat_t)

score = clf.score(test_data_flat_t,test_labels)
print("score",score)

with open('output.txt', 'w') as file:
    file.write(str(np.mean(score)))
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: