Hai Everyone. This is a simple model for classification.

STEPS

1] Importing required libraries

2] Importing the data

3] Data pre-processing

4] creating a model

5] Fitting the model

Step 1 : Importing the required libraries.

In [1]:

```
# This Python 3 environment comes with many helpful anal,
ytics libraries installed
# It is defined by the kaggle/python docker image: http
s://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.
g. pd.read_csv)
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Convolution2D
from keras.layers.convolutional import MaxPooling2D
from sklearn.preprocessing import LabelEncoder,OneHotEn
coder
from keras import backend as K


# Input data files are available in the "../input/" direc
tory.
# For example, running this (by clicking run or pressing
 Shift+Enter) will list the files in the input directory

from subprocess import check_output
print(check_output(["ls", "../input"]).decode("utf8"))


# Any results you write to the current directory are save
d as output.
```

```
Using TensorFlow backend.
/opt/conda/lib/python3.6/importlib/_bootstr
```



```
ap.py:219: RuntimeWarning: compiletime vers
ion 3.5 of module 'tensorflow.python.framew
ork.fast_tensor_util' does not match runtim
e version 3.6
  return f(*args, **kwds)


fashion-mnist_test.csv
fashion-mnist_train.csv
t10k-images-idx3-ubyte
t10k-labels-idx1-ubyte
train-images-idx3-ubyte
train-labels-idx1-ubyte
```

Step 2 :As mentioned above the input file present in "../input/". We import it using pandas

In [2]:

```
# Read training and test data files
train = pd.read_csv("../input/fashion-mnist_train.csv")
.values
test  = pd.read_csv("../input/fashion-mnist_test.csv").
values
```

Step3: This is important step becaues the keras function accepts the inputs as images. But here we are using CSV file which is a array. So at first we have to convert these into image(All images are the 3-D matrix of pixels). So now we re-shape the matrix into a 3-D matrix.we separate the X(input) and Y(output) from Data

We also re-scale the data in the range 0-1 because it will be faster to process. so we divide all the values by 255(The max value in the matrix(pixel) is 255)

In [3]:

```
# Reshape and normalize training data
trainX = train[:, 1:].reshape(train.shape[0],1,28, 28).
astype( 'float32' )
X_train = trainX / 255.0

y_train = train[:,0]


# Reshape and normalize test data
testX = test[:,1:].reshape(test.shape[0],1, 28, 28).ast
ype( 'float32' )
X_test = testX / 255.0

y_test = test[:,0]
```

The model always gives a matrix of probability as output. The max value in that will be the output label. so the next process of data pre-processing is to encode the output. As I said above the output will be in a matrix of 10 values. so we will define 10 neurons in the output layer. But our csv file contains single values. so we enode the values. Example: In the classification of cat(0), dog(1) and monkey(2).The output of the model will look like this [0.23 , 0.01 , 0.91]. But in csv file it will be a single digit i.e 2. so we encode 2 as [0 , 0 , 1], 1 as [0 , 1 , 0] and 0 as [1 , 0 , 0].

This is done by LabelBinarizer() function.

In [4]:

```
from sklearn import preprocessing
lb = preprocessing.LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_test = lb.fit_transform(y_test)
```

Step 4: Now we build a model. The model has 2 Convolution layer 2 MaxPooling layer and 2 hidden layers. The flatten mainly used to flatten the 3-D array of previous layer into a single layer(Because the Ann model only take 1-D array as input).So the flatten creates the input layer. There are 10 labels so there will be 10 neurons in the output layer. There are 784 columns(Pixels).But we reshaped it to 28*28. So the Convolution layer input_shape will be (1,28,28) . Dropout() is used to avoid over-fitting.

In [5]:

```
model = Sequential()
K.set_image_dim_ordering('th')
model.add(Convolution2D(30, 5, 5, border_mode= 'valid'
, input_shape=(1, 28, 28),activation= 'relu' ))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Convolution2D(15, 3, 3, activation= 'relu' ))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation= 'relu' ))
model.add(Dense(50, activation= 'relu' ))
model.add(Dense(10, activation= 'softmax' ))
  # Compile model
model.compile(loss= 'categorical_crossentropy' , optimi
zer= 'adam' , metrics=[ 'accuracy' ])
```

```
/opt/conda/lib/python3.6/site-packages/ipyk
ernel_launcher.py:3: UserWarning: Update yo
ur `Conv2D` call to the Keras 2 API: `Conv2
D(30, (5, 5), input_shape=(1, 28, 28..., ac
tivation="relu", padding="valid")`
  This is separate from the ipykernel packa
ge so we can avoid doing imports until
```



```
/opt/conda/lib/python3.6/site-packages/ipyk
ernel_launcher.py:5: UserWarning: Update yo
ur `Conv2D` call to the Keras 2 API: `Conv2
D(15, (3, 3), activation="relu")`
  """
```

Step 5 : Now we fit the model with the data. You can increase the accuracy by increasing the number of epochs , Conv layer , Maxpool layers.(In this case). But if you are delaing with images then best way is to use Image Data Generator. This uses real-time data augmentation to produce wide varity of images for a same label.The main advantage of this is the amount of Data is reduced.

In [6]:

```
model.fit(X_train, y_train,
          epochs=20,
          batch_size= 160)
score = model.evaluate(X_test, y_test, batch_size=128)
```

```
Epoch 1/20
60000/60000 [==============================
=] - 122s 2ms/step - loss: 0.6972 - acc: 0.
7413
Epoch 2/20
60000/60000 [==============================
=] - 121s 2ms/step - loss: 0.4482 - acc: 0.
8390
Epoch 3/20
60000/60000 [==============================
=] - 122s 2ms/step - loss: 0.3910 - acc: 0.
8590
Epoch 4/20
60000/60000 [==============================
=] - 120s 2ms/step - loss: 0.3580 - acc: 0.
8694
Epoch 5/20
60000/60000 [==============================
=] - 121s 2ms/step - loss: 0.3349 - acc: 0.
8784
Epoch 6/20
60000/60000 [==============================
=] - 120s 2ms/step - loss: 0.3165 - acc: 0.
8835
Epoch 7/20
60000/60000 [==============================
=] - 120s 2ms/step - loss: 0.3021 - acc: 0.
8896
Epoch 8/20
60000/60000 [==============================
=] - 119s 2ms/step - loss: 0.2902 - acc: 0.
8930
Epoch 9/20
```