

```
#Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.core import Dense
from sklearn.metrics import plot_confusion_matrix, ConfusionMatrixDisplay, confusion_matrix
```

در ابتدا کتابخانه هایی و توابعی که در طول کدنویسی به آنها نیاز خواهیم داشت را فراخوانی میکنیم.

در هر مرحله که از هر کدام استفاده میکنیم توضیح مربوط به آن کتابخانه و تابع نوشته خواهد شد.

```
#Load Data
fashion_mnist = keras.datasets.fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

ابتدای کار دیتاست را که در ماژول keras در tensorflow هست را با دستور فوق دانلود می کنیم.

```
#pre-process
print('Training Data shape:',x_train.shape, y_train.shape)
print('Test Data shape:',x_test.shape, y_test.shape)
```

```
Training Data shape: (60000, 28, 28) (60000,)
Test Data shape: (10000, 28, 28) (10000,)
```

برای اینکه بدانیم دیتاست ما از چه نوع هست و چه تعداد هست. با دستور shape ابعاد ورودی و خروجی های آموزش و تست را چاپ می کنیم. در این دیتاست ۶۰۰۰۰ تا عکس سیاه سفید ۲۸\*۲۸ پیکسل بعنوان ورودی آموزش ، ۶۰۰۰۰ تا خروجی آموزش ، ۱۰۰۰۰ تا عکس سیاه و سفید ۲۸\*۲۸ پیکسل بعنوان ورودی تست و ۱۰۰۰۰ تا خروجی تست داریم.

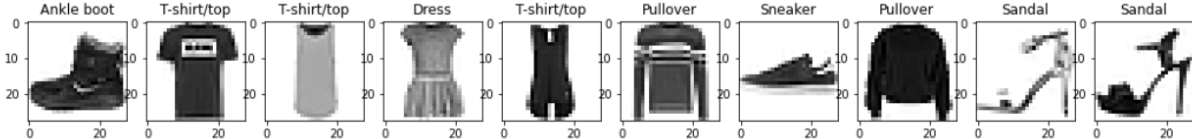
```
#Find the unie clothes from the train labels
label_names = {0:"T-shirt/top",1:"Trouser",2:"Pullover",3:"Dress",4:"Coat",5:"Sandal",6:"Shirt",7:"Sneaker",8:"Bag",9:"Ankle boot"}
classes=np.unique(y_train)
nclasses=len(classes)
print('total number of outputs:', nclasses)
outputs=[]
for i in range (0,nclasses):
    out=label_names[i]
    outputs.append(out)
print('Output Classes:', outputs)
```

total number of outputs: 10  
Output Classes: ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

در این مرحله تعداد خروجی های متمایز را می خواهیم بدانیم چه تعداد است. با کمک تابع unique که از توابع در numpy است این کار را انجام میدهیم. تابع unique برای ما ۱۰ عدد خروجی متمایز را نشان میدهد که هر کدام برچسب مربوط به یک تیپ عکس هست. مثلاً خروجی صفر برای تیشرت است. مقادیر عددی برچسب ها و مقادیر categorical آنها را در یک دیکشنری تعریف می کنیم و انواع مختلف برچسب هایی که داریم را پرینت می گیریم.

```
plt.figure(figsize=(20,10))

#display the 10 image in training data
for i in range (1,11):
    plt.subplot(1,11,i)
    plt.imshow(x_train[i-1,:,:], cmap='gray_r')
    plt.title(label_names[y_train[i-1]])
```



در روی سوال از ما خواسته شده ۱۰ ورودی اول این دیتاست را به همراه برچسب آنها نمایش دهیم.. با استفاده از یک حلقه for و به کمک تابع imshow () و Title () این کار را انجام میدهیم.

```
#flatten the data
#change from matrixs to array of dimension 28*28 to array of dimension 784
x_train=x_train.reshape((len(x_train)), np.prod(x_train.shape[1:])) #(60000,784)
x_test=x_test.reshape((len(x_test)), np.prod(x_test.shape[1:]))
```

قبل از شروع مدلسازی نیاز هست که هر کدام از ورودی هارا که به صورت یک تصویر  $28 \times 28$  یا به اصطلاح یک ماتریس  $28 \times 28$  که هر کدام از درایه های آن مقداری بین ۰ تا ۲۵۵ را دارند به صورت یک بردار در بیاوریم (flatten کنیم). با دستور reshape این کار را انجام میدهیم و برای داده های ورودی آموزش و تست به ترتیب ۶۰۰۰۰ تا بردار با بعد  $28 \times 28 = 784$  و ۱۰۰۰۰ تا بردار با بعد ۷۸۴ خواهیم داشت.

دستور np.prod که استفاده شده برای ضرب کردن تعداد سطر و ستون های ماتریس به کار میرود.

```
#change to float datatype
x_train=x_train.astype('float32')
x_test=x_test.astype('float32')

#normalization from [0:255] to [0:1] // scale the data to lie between 0 to 1
x_train /= 255
x_test /= 255

#convert Labels to one_hot vectors
y_train=np_utils.to_categorical(y_train)
y_test=np_utils.to_categorical(y_test)
```

در ادامه مراحل پیش پردازش :

نوع داده های ورودی را به float تبدیل میکنیم. ( عدد اعشاری)

مقادیر داده های ورودی را بر ۲۵۵ تقسیم میکنیم که همه آنها بین ۰ و ۱ باشند.

برچسب ها را به one-hot vector تبدیل میکنیم. یعنی بردارهایی که فقط درایه ای که برچسب درست دارد مقدار ۱ و مابقی مقدار ۰ دارند. این کار را با استفاده از np\_utils انجام میدهیم که در ابتدای کد از کتابخانه keras فراخوانی شده.

```
: #create model
model=Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))#Hidden Layer 1
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()

#configure the network
model.compile(loss='categorical_crossentropy', optimizer='Adam',metrics=['accuracy'])
```

Model: "sequential\_5"

Layer (type)	Output Shape	Param #
dense_15 (Dense)	(None, 512)	401920
dense_16 (Dense)	(None, 512)	262656
dense_17 (Dense)	(None, 10)	5130
Total params: 669,706		
Trainable params: 669,706		
Non-trainable params: 0		

دیتاست آماده مدلسازی هست. مدل را sequential انتخاب کردیم. با دو لایه مخفی و ۵۱۲ نرون در هر لایه و تابع فعالسازی relu. در لایه اخر چون ۱۰ نوع خروجی مختلف باید داشته باشیم تعداد نرون ها را ۱۰ قرار دادیم و تابع softmax را استفاده کردیم که برای هر خروجی احتمال را محاسبه کند و دارای بیشترین احتمال را به ما بدهد.

برای کامپایل کردن مدل از تابع هزینه `categorical_crossentropy` و از `adam optimizer` استفاده کردیم. و معیار را هم `accuracy` قرار دادیم. زیرا میخواهیم دقت و درستی مدل را بررسی کند.

```
: #training model fitting
myhistory=model.fit(x_train, y_train, epochs=10, batch_size=32 , validation_split=0.3)
```

بعد از اینکه مدل را ساختیم. حال باید آن را روی دیتای `validation` ارزیابی کنیم. برای این کار از دستور `fit()` استفاده میکنیم. داده های ورودی و خروجی آموزش را میدهیم. `Epochs` تعداد تکرار یا `iteration` ها هست که مشخص میکنیم. `Batch_size` یعنی در هر بار که داده ها برای ارزیابی میروند، هر دسته شامل چه تعداد دیتا باشد. اینجا ۳۲ قرار دادیم و دیتا ها ۳۲ تا ۳۲ تا ارزیابی میشوند. و در نهایت `validation_split` را ۰/۳ مشخص میکنیم که یعنی ۳۰٪ از داده های آموزش را به عنوان `validation data` بگیرد.

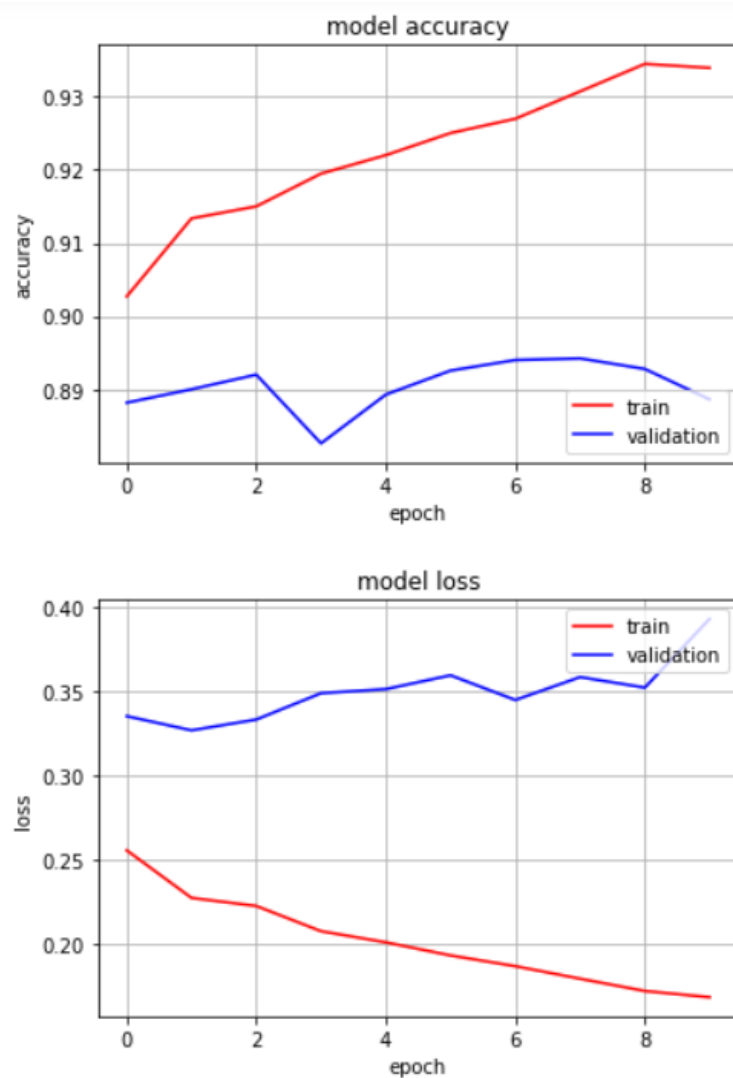
```
Epoch 1/10
1313/1313 [=====] - 13s 10ms/step - loss: 0.2558 - accuracy: 0.9027 - val_loss: 0.3351 - val_accuracy:
0.8882
Epoch 2/10
1313/1313 [=====] - 14s 10ms/step - loss: 0.2277 - accuracy: 0.9134 - val_loss: 0.3267 - val_accuracy:
0.8900
Epoch 3/10
1313/1313 [=====] - 13s 10ms/step - loss: 0.2229 - accuracy: 0.9150 - val_loss: 0.3331 - val_accuracy:
0.8920
Epoch 4/10
1313/1313 [=====] - 13s 10ms/step - loss: 0.2080 - accuracy: 0.9195 - val_loss: 0.3487 - val_accuracy:
0.8826
Epoch 5/10
1313/1313 [=====] - 13s 10ms/step - loss: 0.2014 - accuracy: 0.9220 - val_loss: 0.3511 - val_accuracy:
0.8893
Epoch 6/10
1313/1313 [=====] - 13s 10ms/step - loss: 0.1936 - accuracy: 0.9250 - val_loss: 0.3593 - val_accuracy:
0.8926
Epoch 7/10
1313/1313 [=====] - 12s 9ms/step - loss: 0.1873 - accuracy: 0.9270 - val_loss: 0.3447 - val_accuracy:
0.8940
Epoch 8/10
1313/1313 [=====] - 12s 9ms/step - loss: 0.1799 - accuracy: 0.9307 - val_loss: 0.3582 - val_accuracy:
0.8942
Epoch 9/10
1313/1313 [=====] - 12s 9ms/step - loss: 0.1726 - accuracy: 0.9345 - val_loss: 0.3520 - val_accuracy:
0.8928
Epoch 10/10
1313/1313 [=====] - 12s 9ms/step - loss: 0.1689 - accuracy: 0.9339 - val_loss: 0.3927 - val_accuracy:
0.8886
```

نتایج را مقادیر داده شده در بالا به این صورت میبینیم که در ۱۰ تکرار سرعت و دقت را نمایش می دهد.

```
#plotting metrics
#plot the accuracy curve
fig=plt.figure()
plt.plot(myhistory.history['accuracy'],'r')
plt.plot(myhistory.history['val_accuracy'],'b')
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','validation'], loc='lower right')
plt.grid()

#plot the loss curves
fig=plt.figure()
plt.plot(myhistory.history['loss'],'r')
plt.plot(myhistory.history['val_loss'],'b')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','validation'], loc='upper right')
plt.grid()
```

برای درک و تحلیل بهتر دقت را روی نمودار نمایش می دهیم:



نتایج به صورت فوق است.



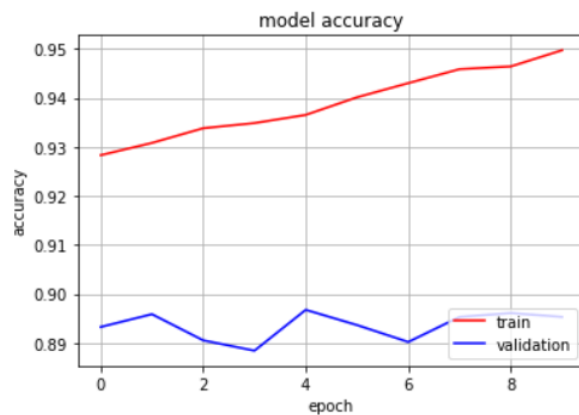
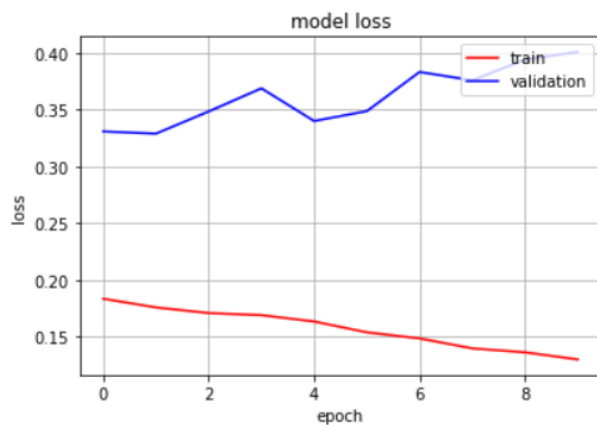
سایز mini batch را تغییر می دهیم.

Batch\_size=64

```
#training model fitting
```

```
myhistory=model.fit(x_train, y_train, epochs=10, batch_size=64 , validation_split=0.3)
```

```
Epoch 1/10
657/657 [=====] - 4s 6ms/step - loss: 0.1833 - accuracy: 0.9283 - val_loss: 0.3309 - val_accuracy: 0.8933
Epoch 2/10
657/657 [=====] - 4s 6ms/step - loss: 0.1756 - accuracy: 0.9308 - val_loss: 0.3290 - val_accuracy: 0.8959
Epoch 3/10
657/657 [=====] - 4s 7ms/step - loss: 0.1706 - accuracy: 0.9338 - val_loss: 0.3486 - val_accuracy: 0.8906
Epoch 4/10
657/657 [=====] - 4s 6ms/step - loss: 0.1688 - accuracy: 0.9349 - val_loss: 0.3690 - val_accuracy: 0.8884
Epoch 5/10
657/657 [=====] - 4s 7ms/step - loss: 0.1632 - accuracy: 0.9365 - val_loss: 0.3401 - val_accuracy: 0.8968
Epoch 6/10
657/657 [=====] - 4s 6ms/step - loss: 0.1536 - accuracy: 0.9401 - val_loss: 0.3488 - val_accuracy: 0.8937
Epoch 7/10
657/657 [=====] - 4s 6ms/step - loss: 0.1483 - accuracy: 0.9430 - val_loss: 0.3834 - val_accuracy: 0.8902
Epoch 8/10
657/657 [=====] - 4s 6ms/step - loss: 0.1395 - accuracy: 0.9459 - val_loss: 0.3761 - val_accuracy: 0.8953
Epoch 9/10
657/657 [=====] - 4s 6ms/step - loss: 0.1360 - accuracy: 0.9464 - val_loss: 0.3943 - val_accuracy: 0.8961
Epoch 10/10
657/657 [=====] - 4s 7ms/step - loss: 0.1297 - accuracy: 0.9497 - val_loss: 0.4011 - val_accuracy: 0.8953
```



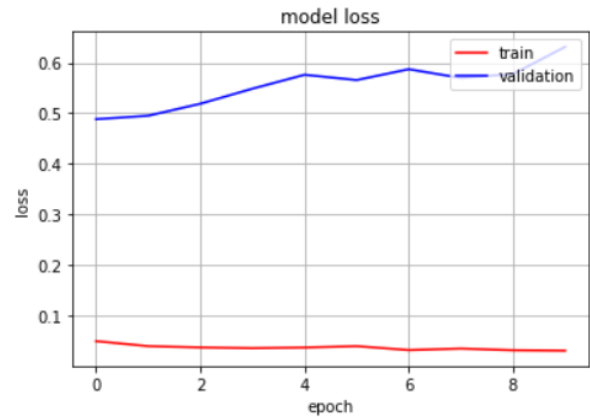
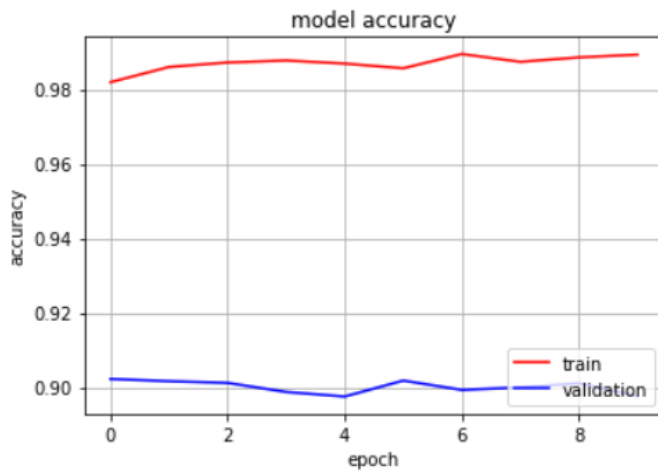
```
313/313 [=====] - 1s 2ms/step - loss: 0.4356 - accuracy: 0.8882
test loss= 0.43564316630363464
test accuracy= 0.8881999850273132
```

Batch\_size=256

*#training model fitting*

```
myhistory=model.fit(x_train, y_train, epochs=10, batch_size=128 , validation_split=0.3)
```

```
Epoch 1/10
165/165 [=====] - 2s 10ms/step - loss: 0.0497 - accuracy: 0.9819 - val_loss: 0.4883 - val_accuracy: 0.9022
Epoch 2/10
165/165 [=====] - 2s 10ms/step - loss: 0.0399 - accuracy: 0.9860 - val_loss: 0.4951 - val_accuracy: 0.9016
Epoch 3/10
165/165 [=====] - 2s 11ms/step - loss: 0.0373 - accuracy: 0.9872 - val_loss: 0.5188 - val_accuracy: 0.9012
Epoch 4/10
165/165 [=====] - 2s 10ms/step - loss: 0.0358 - accuracy: 0.9877 - val_loss: 0.5485 - val_accuracy: 0.8987
Epoch 5/10
165/165 [=====] - 2s 11ms/step - loss: 0.0371 - accuracy: 0.9869 - val_loss: 0.5760 - val_accuracy: 0.8975
Epoch 6/10
165/165 [=====] - 2s 11ms/step - loss: 0.0398 - accuracy: 0.9856 - val_loss: 0.5656 - val_accuracy: 0.9018
Epoch 7/10
165/165 [=====] - 2s 10ms/step - loss: 0.0321 - accuracy: 0.9895 - val_loss: 0.5870 - val_accuracy: 0.8993
Epoch 8/10
165/165 [=====] - 2s 10ms/step - loss: 0.0351 - accuracy: 0.9874 - val_loss: 0.5698 - val_accuracy: 0.8999
Epoch 9/10
165/165 [=====] - 2s 10ms/step - loss: 0.0318 - accuracy: 0.9886 - val_loss: 0.5776 - val_accuracy: 0.9011
Epoch 10/10
165/165 [=====] - 2s 10ms/step - loss: 0.0309 - accuracy: 0.9893 - val_loss: 0.6312 - val_accuracy: 0.8976
```



```
313/313 [=====] - 1s 3ms/step - loss: 0.7290 - accuracy: 0.8907
test loss= 0.7290086150169373
test accuracy= 0.8906999826431274
```



نتیجه گیری:

با توجه به نمودارها و اعداد و ارقام بدست آمده:

سرعت:

هر `batch_size` کوچکتر باشد سرعت محاسبات بیشتر میشود. در سایز ۳۲ هر گام در ۱۲-۱۳ ثانیه محاسبه شد. در سایز ۶۴ هر گام در ۴ ثانیه و در سایز ۲۵۶ هر گام در ۲ ثانیه انجام گرفت.

میزان دقت:

مدلی که ساختیم برای `validation data` با بزرگتر شدن سایز `batch` عملکرد بهتری داشت. به طوری که ما برای تابع `loss` که به دنبال مینیمم بودیم با سایز ۲۵۶ کمترین مقدارها را گرفتیم در هر `epoch`.

همچنین دقت یا `accuracy` مدل هم در سایز ۲۵۶ بیشترین بود. یعنی دقیق تر عمل کرد.

اما با توجه به نمودارهای مقایسه `loss` و `accuracy` دیتاهای `train` و `validation` مدلی که ساختیم قابل قبول نیست چون در همه `batch size`ها با توجه به آنچه در نمودار میبینیم فاصله بین `min loss` و `max accuracy` در `train` نسبت به `validation` خیلی زیاد است و نیازمند تغییر معماری شبکه و مدل برای به دست آوردن مدل بهتری است.