Fashion- MNIST Data Classification

-AKASH S

Agenda:

- Abstract
- Objective
- Introduction
- Methodology
- Code
- Conclusion

Abstract:

In deep learning, a convolutional neural network (CNN) is a class of deep neural networks, that are typically used to recognize patterns present in images but they are also used for spatial

data analysis, computer vision, natural language processing, signal processing, and various other purposes.

The architecture of a Convolutional Network resembles the connectivity pattern of neurons in the Human Brain and was inspired by the organization of the Visual Cortex.

Here I used CNN to classify different fashion products from a given set of images using Convolutional neural network.

Objective:

The objective is to identify the different fashion products. The target dataset has 10 class labels, as we can see from above (0 – T-shirt/top, 1 – Trouser,….9– Ankle Boot).

Given the images of the articles, we need to classify them into one of these classes, hence, it is essentially a 'Multi-class Classification' problem.

Introduction:

In this project involves different layers of Convolution Neural network(CNN), namely :

- Convolution
- Relu(Rectified linear unit)
- Pooling
- Fully connected layer

With the help of these layers, form a neural network to attain the neuron property for a device.

Methodology:

Convolution layer has main work to analyse the input deeply to mark the pixels in the image.Convolution has the nice property of being **translational invariant**. Intuitively, this means that **each** convolution filter represents a **feature** of interest (e.g **pixels in letters)** and the Convolutional Neural Network **algorithm** learns which **features** comprise the **resulting reference** (i.e. alphabet).

Relu is also known as Activation function.

Rectified Linear Unit (ReLU) transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable.

Pooling Layer, **shrink** the **image** stack into a **smaller size.** Pooling is done **after passing** through the **activation** layer. We do this by implementing the following 4 steps:

- Pick a **window size** (usually 2 or 3)
- Pick a **stride** (usually 2)
- **Walk** your window **across** your **filtered** images
- From each **window,** take the **maximum** value

Fully connected layer is the final layer where the classification actually happens. Here take our filtered and shrinked images and put them into one single list.After the pooling layer also the data in the form of matrix only.Convert it into linearly flatten in fully connected layer.

CNN has generally two models:

(a) Basic CNN model
(b) Complex CNN model

In Basic CNN model, layered one time to get .But in Complex CNN model, more than one time to repeat the layering process to get the predict in more accurate. Here I made Both models to identify(predict) different fashion products.

Code:

For Basic CNN model,

#import libraries

```
import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

import keras
```

"""Load Data"""

```
(X_train, y_train), (X_test,
y_test)=tf.keras.datasets.fashion_mnist.load_data()
```

Print the shape of data

```
X_train.shape,y_train.shape, "***************" ,
X_test.shape,y_test.shape

y_train[200]


class_labels = [    "T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
"Sandal", "Shirt",        "Sneaker", "Bag",        "Ankle boot"]

class_labels
```

showing image

```
plt.imshow(X_train[200],cmap='Greys')
```

```
plt.figure(figsize=(16,16))
j=1
for  i in np.random.randint(0,1000,25):
  plt.subplot(5,5,j);j+=1
  plt.imshow(X_train[i],cmap='Greys')
  plt.axis('off')
  plt.title('{} / {}'.format(class_labels[y_train[i]],y_train[i]))
X_train.ndim
X_train = np.expand_dims(X_train,-1)
X_train.ndim
X_test=np.expand_dims(X_test,-1)
```

feature scaling

```
X_train = X_train/255
X_test= X_test/255
```

Split dataset

```
from sklearn.model_selection import  train_test_split
X_train,X_Validation,y_train,y_Validation=train_test_split(X_train,y_train,test_size=0.2,random_state=2020)
X_train.shape,X_Validation.shape,y_train.shape,y_Validation.shape
```

"""Build **CNN** model"""

```
# Construct simple CNN model
model=keras.models.Sequential([
keras.layers.Conv2D(filters=32,kernel_size=3,strides=(1,1),padding='valid',activation='relu',input_shape=[28,28,1]),
keras.layers.MaxPooling2D(pool_size=(2,2)),
                    keras.layers.Flatten(),
                    keras.layers.Dense(units=128,activation='relu'),
                    keras.layers.Dense(units=10,activation='softmax')
])
model.summary()
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])


model.fit(X_train,y_train,epochs=10,batch_size=512,verbose=1,validation_data=(X_Validation,y_Validation))


y_pred = model.predict(X_test)
y_pred.round(2)
y_test
model.evaluate(X_test, y_test)


plt.figure(figsize=(16,16))
j=1
for i in np.random.randint(0, 1000,25):
  plt.subplot(5,5, j); j+=1
```

```
  plt.imshow(X_test[i].reshape(28,28), cmap = 'Greys')

  plt.title('Actual = {} / {} \nPredicted = {} / {}'.format(class_labels[y_test[i]], y_test[i], class_labels[np.argmax(y_pred[i])],np.argmax(y_pred[i])))

  plt.axis('off')

plt.figure(figsize=(16,30))

j=1

for i in np.random.randint(0, 1000,60):

  plt.subplot(10,6, j); j+=1

  plt.imshow(X_test[i].reshape(28,28), cmap = 'Greys')

  plt.title('Actual = {} / {} \nPredicted = {} / {}'.format(class_labels[y_test[i]], y_test[i], class_labels[np.argmax(y_pred[i])],np.argmax(y_pred[i])))

  plt.axis('off')


"""## Confusion Matrix"""

from sklearn.metrics import confusion_matrix

plt.figure(figsize=(16,9))

y_pred_labels = [ np.argmax(label) for label in y_pred ]

cm = confusion_matrix(y_test, y_pred_labels)


sns.heatmap(cm, annot=True, fmt='d',xticklabels=class_labels, yticklabels=class_labels)


from sklearn.metrics import classification_report
```

cr= classification_report(y_test, y_pred_labels, target_names=class_labels)

print(cr)

"""# Save Model"""

model.save('fashion_mnist_cnn_model.h5')

t = plt.imread('/trouser.webp')

plt.imshow(t)

Resize the input image to reduce the pixel into 28,28.

#Because we trained our CNN model with 28,28 pixel only

from skimage import transform

resize = transform.resize(t,(28,28,3))

img = plt.imshow(resize)

x_pred=np.array([resize])

x_pred

resize

tr=model.predict(x_pred)

print(tr)

File for Basic CNN model at :

https://colab.research.google.com/drive/1hot3buUNDcKpmF9LoaGtYnmoTaKhIsNS

Output For the Basic CNN model:

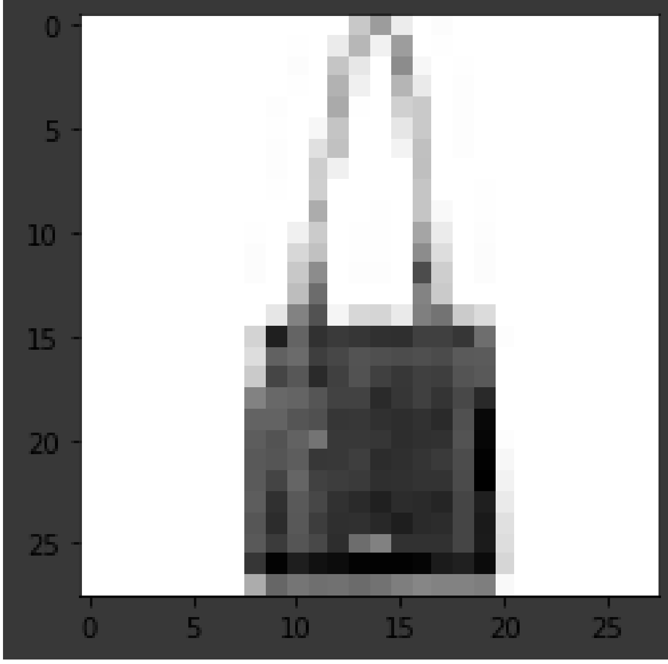
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

```
((60000, 28, 28), (60000,), '**************', (10000, 28, 28), (10000,))
```

```
8
```

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

```
<matplotlib.image.AxesImage at 0x7fd3934535d0>
```




```
3
```

```
4
```

```
((48000, 28, 28, 1), (12000, 28, 28, 1), (48000,), (12000,))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 flatten (Flatten)           (None, 5408)              0

 dense (Dense)               (None, 128)               692352

 dense_1 (Dense)             (None, 10)                1290
```

```
=================================================================
Total params: 693,962
Trainable params: 693,962
Non-trainable params: 0
```

```
Epoch 1/10
94/94 [==============================] - 23s 225ms/step - loss: 0.6338 - accuracy: 0.7922 -
val_loss: 0.4314 - val_accuracy: 0.8508
Epoch 2/10
94/94 [==============================] - 22s 236ms/step - loss: 0.3788 - accuracy: 0.8662 -
val_loss: 0.3813 - val_accuracy: 0.8627
Epoch 3/10
94/94 [==============================] - 21s 223ms/step - loss: 0.3290 - accuracy: 0.8837 -
val_loss: 0.3272 - val_accuracy: 0.8852
Epoch 4/10
94/94 [==============================] - 23s 242ms/step - loss: 0.2953 - accuracy: 0.8956 -
val_loss: 0.3266 - val_accuracy: 0.8851
Epoch 5/10
94/94 [==============================] - 21s 223ms/step - loss: 0.2832 - accuracy: 0.8988 -
val_loss: 0.2983 - val_accuracy: 0.8946
Epoch 6/10
94/94 [==============================] - 25s 262ms/step - loss: 0.2602 - accuracy: 0.9078 -
val_loss: 0.2942 - val_accuracy: 0.8951
Epoch 7/10
94/94 [==============================] - 21s 223ms/step - loss: 0.2450 - accuracy: 0.9127 -
val_loss: 0.2932 - val_accuracy: 0.8969
Epoch 8/10
94/94 [==============================] - 22s 239ms/step - loss: 0.2312 - accuracy: 0.9169 -
val_loss: 0.2846 - val_accuracy: 0.9003
Epoch 9/10
94/94 [==============================] - 23s 244ms/step - loss: 0.2208 - accuracy: 0.9199 -
val_loss: 0.2736 - val_accuracy: 0.9038
Epoch 10/10
94/94 [==============================] - 22s 237ms/step - loss: 0.2138 - accuracy: 0.9234 -
val_loss: 0.2635 - val_accuracy: 0.9068
<keras.callbacks.History at 0x7fd39a2b04d0>
313/313 [==============================] - 3s 8ms/step

array([[0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

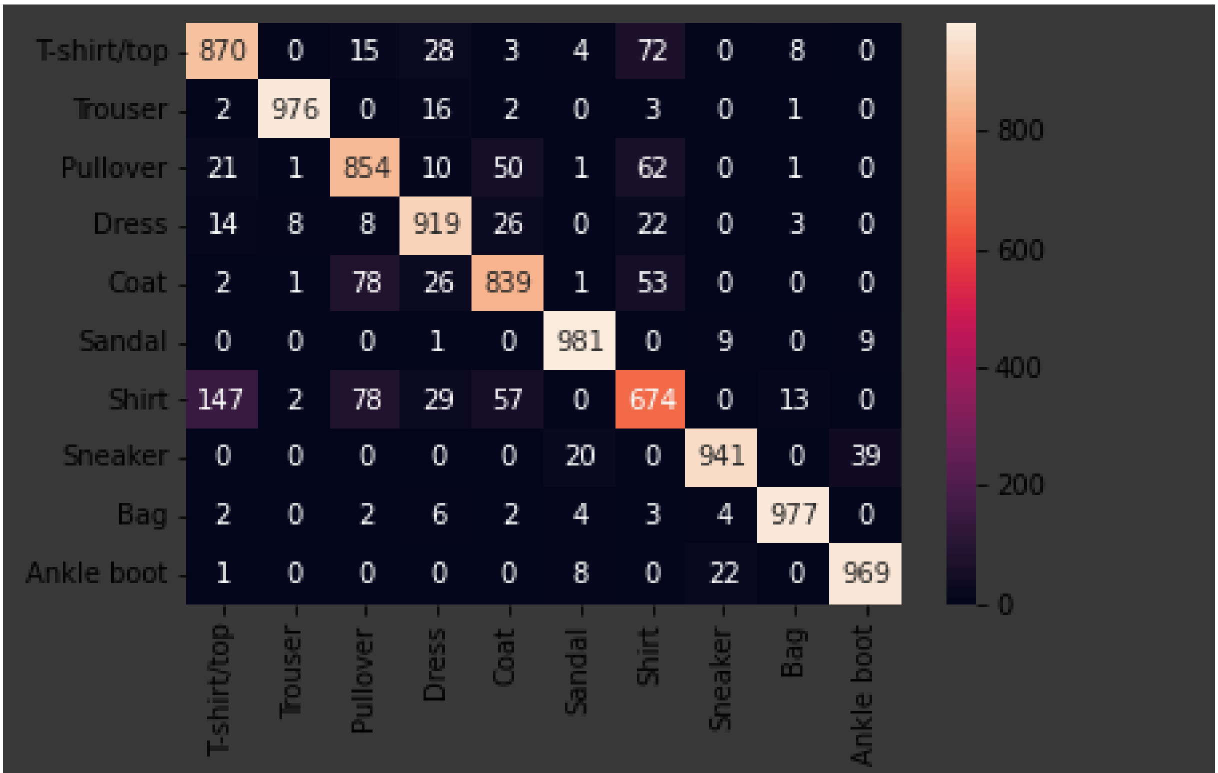
```
array([9, 2, 1, ..., 8, 1, 5], dtype=uint8)
```

```
313/313 [==============================] - 4s 13ms/step - loss: 0.2761 - accuracy: 0.9000
[0.2761099636554718, 0.8999999761581421]
```


```
             precision    recall  f1-score   support

 T-shirt/top       0.82      0.87      0.85      1000
     Trouser       0.99      0.98      0.98      1000
    Pullover       0.83      0.85      0.84      1000
       Dress       0.89      0.92      0.90      1000
        Coat       0.86      0.84      0.85      1000
      Sandal       0.96      0.98      0.97      1000
       Shirt       0.76      0.67      0.71      1000
     Sneaker       0.96      0.94      0.95      1000
         Bag       0.97      0.98      0.98      1000
  Ankle boot       0.95      0.97      0.96      1000

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```



<matplotlib.image.AxesImage at 0x7fd39378c310>

trouser

For Complex CNN model,

#Import Libraries

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

import keras

(X_train, y_train), (X_test, y_test)=tf.keras.datasets.fashion_mnist.load_data()

class_labels = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

"""Build **2 Complex CNN** model"""

#Building CNN model

cnn_model2 = keras.models.Sequential([

keras.layers.Conv2D(filters=32, kernel_size=3, strides=(1,1), padding='valid',activation= 'relu', input_shape=[28,28,1]),

keras.layers.MaxPooling2D(pool_size=(2,2)),

keras.layers.Conv2D(filters=64, kernel_size=3, strides=(2,2), padding='same', activation='relu'),

keras.layers.MaxPooling2D(pool_size=(2,2)),

```
keras.layers.Flatten(),
keras.layers.Dense(units=128, activation='relu'),
keras.layers.Dropout(0.25),
keras.layers.Dense(units=256, activation='relu'),
keras.layers.Dropout(0.25),
keras.layers.Dense(units=128, activation='relu'),
keras.layers.Dense(units=10, activation='softmax')
])
```

complie the model

```
cnn_model2.compile(optimizer='adam', loss=
'sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
from sklearn.model_selection import  train_test_split
X_train,X_Validation,y_train,y_Validation=train_test_split(X_train,y_
train,test_size=0.2,random_state=2020)
X_train.shape,X_Validation.shape,y_train.shape,y_Validation.shape
```

#Train the Model

```
cnn_model2.fit(X_train, y_train, epochs=20, batch_size=512,
verbose=1, validation_data=(X_Validation, y_Validation))
```

```
cnn_model2.save('fashion_mnist_cnn_model2.h5')
```

```
y_pred = cnn_model2.predict(X_test)
y_pred.round(2)
y_test
```

"""Test the Model"""

Testing the model of 2 complex CNN

```
cnn_model2.evaluate(X_test, y_test)


plt.figure(figsize=(16,16))
j=1
for i in np.random.randint(0, 1000,25):
  plt.subplot(5,5, j); j+=1
  plt.imshow(X_test[i].reshape(28,28), cmap = 'Greys')
  plt.title('Actual = {} / {} \nPredicted = {} / 
{}'.format(class_labels[y_test[i]], y_test[i], 
class_labels[np.argmax(y_pred[i])],np.argmax(y_pred[i])))
  plt.axis('off')


plt.figure(figsize=(16,30))
j=1
for i in np.random.randint(0, 1000,60):
  plt.subplot(10,6, j); j+=1
  plt.imshow(X_test[i].reshape(28,28), cmap = 'Greys')
```

```
  plt.title('Actual = {} / {} \nPredicted = {} / {}'.format(class_labels[y_test[i]], y_test[i], class_labels[np.argmax(y_pred[i])],np.argmax(y_pred[i])))
  plt.axis('off')
```

"""## Confusion Matrix"""

```
from sklearn.metrics import confusion_matrix
plt.figure(figsize=(16,9))
y_pred_labels = [ np.argmax(label) for label in y_pred ]
cm = confusion_matrix(y_test, y_pred_labels)

sns.heatmap(cm, annot=True, fmt='d',xticklabels=class_labels, yticklabels=class_labels)

from sklearn.metrics import classification_report
cr= classification_report(y_test, y_pred_labels, target_names=class_labels)
print(cr)

t = plt.imread('/trouser.webp')
plt.imshow(t)

# Resize the input image to reduce the pixel into 28,28.
#Because we trained our CNN model with 28,28 pixel only
```

```
from skimage import transform
resize = transform.resize(t,(28,28,3))
img = plt.imshow(resize)
x_pred=np.array([resize])
x_pred
resize
tr=model.predict(x_pred)
print(tr)
```

"""Build **3 complex CNN** model"""

```
"""######## very complex model"""
#Building CNN model
cnn_model3 = keras.models.Sequential([
                keras.layers.Conv2D(filters=64, kernel_size=3,
strides=(1,1), padding='valid',activation= 'relu',
input_shape=[28,28,1]),
                keras.layers.MaxPooling2D(pool_size=(2,2)),
                keras.layers.Conv2D(filters=128, kernel_size=3,
strides=(2,2), padding='same', activation='relu'),
                keras.layers.MaxPooling2D(pool_size=(2,2)),
                keras.layers.Conv2D(filters=64, kernel_size=3,
strides=(2,2), padding='same', activation='relu'),
```

```
                    keras.layers.MaxPooling2D(pool_size=(2,2)),
                    keras.layers.Flatten(),
                    keras.layers.Dense(units=128, activation='relu'),
                    keras.layers.Dropout(0.25),
                    keras.layers.Dense(units=256, activation='relu'),
                    keras.layers.Dropout(0.5),
                    keras.layers.Dense(units=256, activation='relu'),
                    keras.layers.Dropout(0.25),
                    keras.layers.Dense(units=128, activation='relu'),
                    keras.layers.Dropout(0.10),
                    keras.layers.Dense(units=10, activation='softmax')
                    ])

# complie the model
cnn_model3.compile(optimizer='adam', loss= 'sparse_categorical_crossentropy', metrics=['accuracy'])
```

#Train the Model

```
cnn_model3.fit(X_train, y_train, epochs=50, batch_size=512, verbose=1, validation_data=(X_Validation, y_Validation))

cnn_model3.save('fashion_mnist_cnn_model3.h5')

cnn_model3.evaluate(X_test, y_test)
```

```
y_pred = cnn_model3.predict(X_test)
y_pred.round(2)

y_test

"""**Test** the model"""
#Testing the model of 3 complex CNN

cnn_model3.evaluate(X_test, y_test)

plt.figure(figsize=(16,16))
j=1
for i in np.random.randint(0, 1000,25):
  plt.subplot(5,5, j); j+=1
  plt.imshow(X_test[i].reshape(28,28), cmap = 'Greys')
  plt.title('Actual = {} / {} \nPredicted = {} / {}'.format(class_labels[y_test[i]], y_test[i], class_labels[np.argmax(y_pred[i])],np.argmax(y_pred[i])))
  plt.axis('off')

plt.figure(figsize=(16,30))
j=1
for i in np.random.randint(0, 1000,60):
```

```
  plt.subplot(10,6, j); j+=1
  plt.imshow(X_test[i].reshape(28,28), cmap = 'Greys')
  plt.title('Actual = {} / {} \nPredicted = {} / {}'.format(class_labels[y_test[i]], y_test[i], class_labels[np.argmax(y_pred[i])],np.argmax(y_pred[i])))
  plt.axis('off')

"""## Confusion Matrix"""

from sklearn.metrics import confusion_matrix
plt.figure(figsize=(16,9))
y_pred_labels = [ np.argmax(label) for label in y_pred ]
cm = confusion_matrix(y_test, y_pred_labels)

sns.heatmap(cm, annot=True, fmt='d',xticklabels=class_labels, yticklabels=class_labels)

from sklearn.metrics import classification_report
cr= classification_report(y_test, y_pred_labels, target_names=class_labels)
print(cr)
```

File for Complex CNN model at :

https://colab.research.google.com/drive/1t6TxKZHbCmZe2oxarIpyLat9etj9gEp2

Output For the 2 Complex CNN model:

```
Epoch 1/20
94/94 [==============================] - 31s 319ms/step - loss: 1.5704
- accuracy: 0.5886 - val_loss: 0.5501 - val_accuracy: 0.7916
Epoch 2/20
94/94 [==============================] - 29s 312ms/step - loss: 0.5844
- accuracy: 0.7830 - val_loss: 0.4634 - val_accuracy: 0.8322
Epoch 3/20
94/94 [==============================] - 27s 290ms/step - loss: 0.5041
- accuracy: 0.8135 - val_loss: 0.4140 - val_accuracy: 0.8477
Epoch 4/20
94/94 [==============================] - 28s 302ms/step - loss: 0.4444
- accuracy: 0.8375 - val_loss: 0.3856 - val_accuracy: 0.8560
Epoch 5/20
94/94 [==============================] - 27s 291ms/step - loss: 0.4063
- accuracy: 0.8504 - val_loss: 0.3707 - val_accuracy: 0.8624
Epoch 6/20
94/94 [==============================] - 30s 319ms/step - loss: 0.3813
- accuracy: 0.8582 - val_loss: 0.3525 - val_accuracy: 0.8705
Epoch 7/20
94/94 [==============================] - 29s 311ms/step - loss: 0.3675
- accuracy: 0.8646 - val_loss: 0.3473 - val_accuracy: 0.8741
Epoch 8/20
94/94 [==============================] - 29s 306ms/step - loss: 0.3453
- accuracy: 0.8725 - val_loss: 0.3371 - val_accuracy: 0.8741
Epoch 9/20
94/94 [==============================] - 28s 295ms/step - loss: 0.3280
- accuracy: 0.8789 - val_loss: 0.3351 - val_accuracy: 0.8763
Epoch 10/20
94/94 [==============================] - 29s 307ms/step - loss: 0.3146
- accuracy: 0.8827 - val_loss: 0.3345 - val_accuracy: 0.8750
Epoch 11/20
94/94 [==============================] - 28s 293ms/step - loss: 0.3030
- accuracy: 0.8873 - val_loss: 0.3146 - val_accuracy: 0.8858
Epoch 12/20
94/94 [==============================] - 30s 321ms/step - loss: 0.2916
- accuracy: 0.8927 - val_loss: 0.3081 - val_accuracy: 0.8885
Epoch 13/20
94/94 [==============================] - 29s 307ms/step - loss: 0.2805
- accuracy: 0.8944 - val_loss: 0.3152 - val_accuracy: 0.8848
Epoch 14/20
94/94 [==============================] - 27s 291ms/step - loss: 0.2753
- accuracy: 0.8977 - val_loss: 0.3144 - val_accuracy: 0.8867
Epoch 15/20
94/94 [==============================] - 29s 306ms/step - loss: 0.2687
- accuracy: 0.9000 - val_loss: 0.3104 - val_accuracy: 0.8854
Epoch 16/20
94/94 [==============================] - 29s 306ms/step - loss: 0.2580
- accuracy: 0.9025 - val_loss: 0.3157 - val_accuracy: 0.8869
```

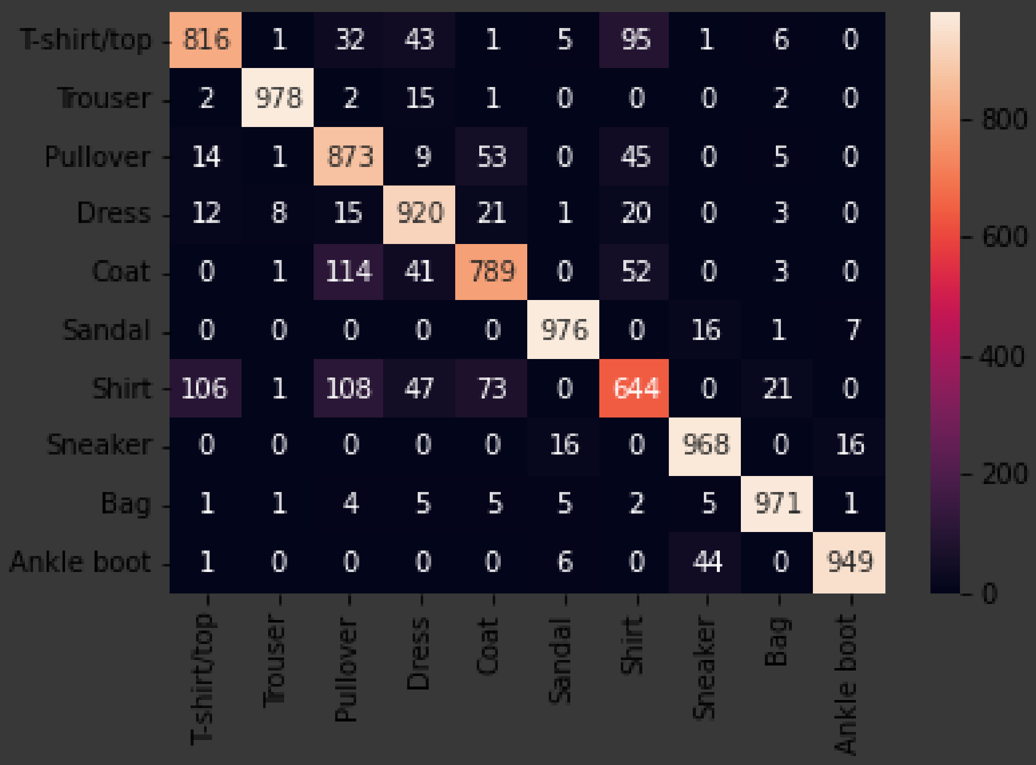
```
Epoch 17/20
94/94 [==============================] - 27s 291ms/step - loss: 0.2541
- accuracy: 0.9043 - val_loss: 0.3015 - val_accuracy: 0.8920
Epoch 18/20
94/94 [==============================] - 30s 321ms/step - loss: 0.2440
- accuracy: 0.9089 - val_loss: 0.3182 - val_accuracy: 0.8863
Epoch 19/20
94/94 [==============================] - 29s 307ms/step - loss: 0.2392
- accuracy: 0.9095 - val_loss: 0.3037 - val_accuracy: 0.8907
Epoch 20/20
94/94 [==============================] - 27s 288ms/step - loss: 0.2350
- accuracy: 0.9107 - val_loss: 0.3121 - val_accuracy: 0.8907
```

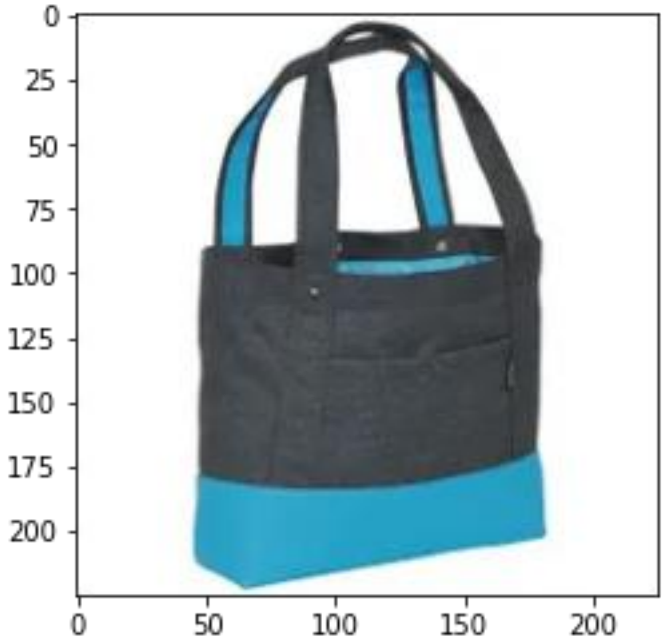
```
313/313 [==============================] - 4s 12ms/step - loss: 0.3261 - accuracy: 0.8884
[0.3261258602142334, 0.8884000182151794]
```

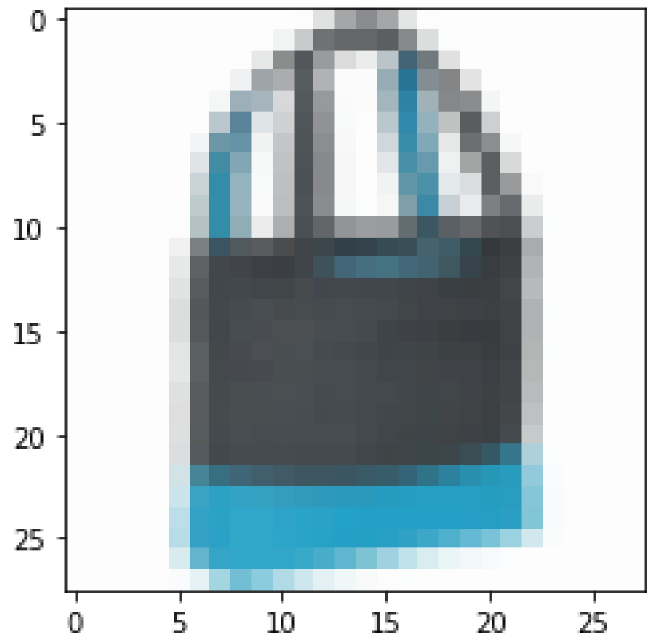

```
              precision    recall  f1-score   support

 T-shirt/top       0.86      0.82      0.84      1000
     Trouser       0.99      0.98      0.98      1000
    Pullover       0.76      0.87      0.81      1000
```

Dress	0.85	0.92	0.88	1000
Coat	0.84	0.79	0.81	1000
Sandal	0.97	0.98	0.97	1000
Shirt	0.75	0.64	0.69	1000
Sneaker	0.94	0.97	0.95	1000
Bag	0.96	0.97	0.97	1000
Ankle boot	0.98	0.95	0.96	1000
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
weighted avg	0.89	0.89	0.89	10000







Output For the 3 Complex CNN model:

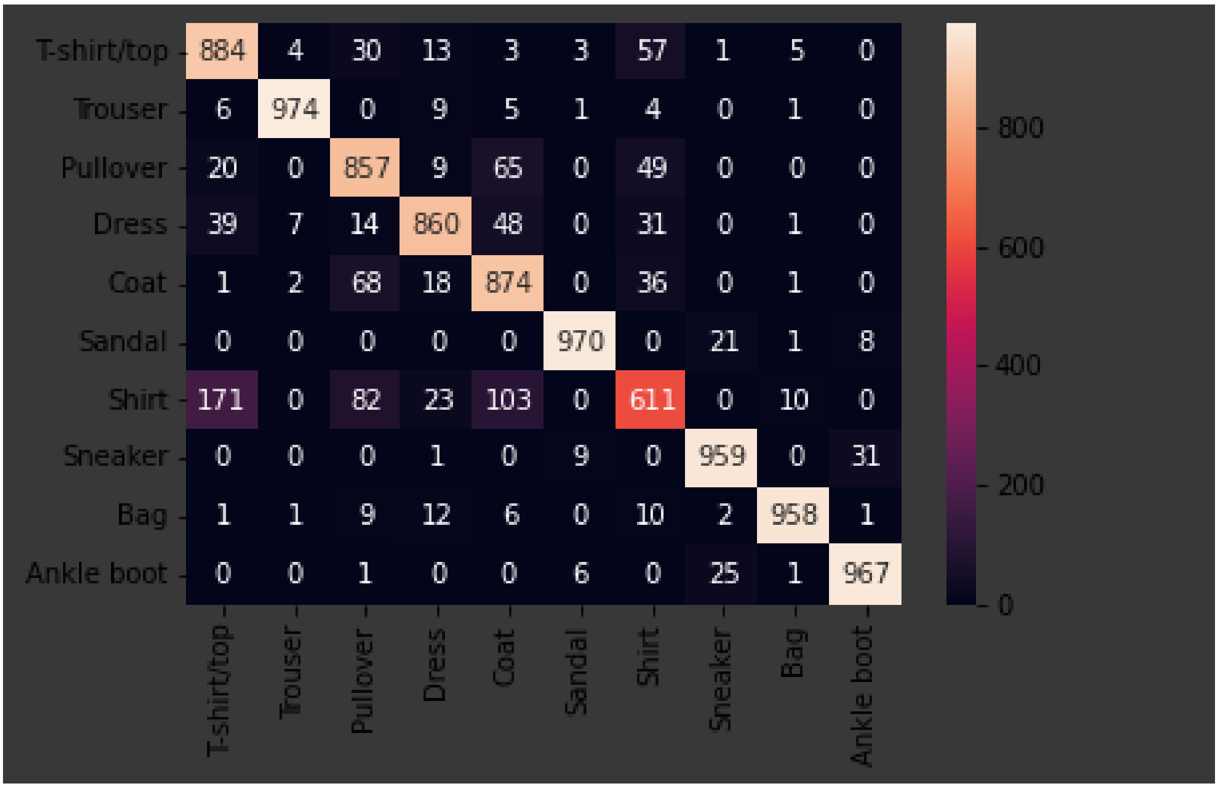
```
Epoch 1/50
94/94 [==============================] - 70s 740ms/step - loss: 1.3332 - accuracy: 0.5359 -
val_loss: 0.5893 - val_accuracy: 0.7717
Epoch 2/50
94/94 [==============================] - 63s 666ms/step - loss: 0.5877 - accuracy: 0.7823 -
val_loss: 0.4630 - val_accuracy: 0.8238
Epoch 3/50
94/94 [==============================] - 65s 687ms/step - loss: 0.4869 - accuracy: 0.8239 -
val_loss: 0.3964 - val_accuracy: 0.8536
```

```
Epoch 4/50
94/94 [==============================] - 62s 654ms/step - loss: 0.4180 - accuracy: 0.8521 -
val_loss: 0.3763 - val_accuracy: 0.8664
Epoch 5/50
94/94 [==============================] - 62s 662ms/step - loss: 0.3721 - accuracy: 0.8698 -
val_loss: 0.3557 - val_accuracy: 0.8758
Epoch 6/50
94/94 [==============================] - 64s 681ms/step - loss: 0.3414 - accuracy: 0.8795 -
val_loss: 0.3333 - val_accuracy: 0.8832
Epoch 7/50
94/94 [==============================] - 62s 657ms/step - loss: 0.3163 - accuracy: 0.8884 -
val_loss: 0.3261 - val_accuracy: 0.8848
Epoch 8/50
94/94 [==============================] - 63s 669ms/step - loss: 0.2991 - accuracy: 0.8944 -
val_loss: 0.3405 - val_accuracy: 0.8779
Epoch 9/50
94/94 [==============================] - 63s 671ms/step - loss: 0.2823 - accuracy: 0.9009 -
val_loss: 0.3158 - val_accuracy: 0.8903
Epoch 10/50
94/94 [==============================] - 62s 655ms/step - loss: 0.2708 - accuracy: 0.9035 -
val_loss: 0.3301 - val_accuracy: 0.8867
Epoch 11/50
94/94 [==============================] - 64s 679ms/step - loss: 0.2602 - accuracy: 0.9084 -
val_loss: 0.3052 - val_accuracy: 0.8945
Epoch 12/50
94/94 [==============================] - 62s 658ms/step - loss: 0.2445 - accuracy: 0.9126 -
val_loss: 0.3114 - val_accuracy: 0.8893
Epoch 13/50
94/94 [==============================] - 63s 672ms/step - loss: 0.2360 - accuracy: 0.9159 -
val_loss: 0.2988 - val_accuracy: 0.8967
Epoch 14/50
94/94 [==============================] - 62s 661ms/step - loss: 0.2232 - accuracy: 0.9199 -
val_loss: 0.3158 - val_accuracy: 0.8923
Epoch 15/50
94/94 [==============================] - 63s 668ms/step - loss: 0.2163 - accuracy: 0.9217 -
val_loss: 0.3087 - val_accuracy: 0.8953
Epoch 16/50
94/94 [==============================] - 63s 668ms/step - loss: 0.2121 - accuracy: 0.9226 -
val_loss: 0.3238 - val_accuracy: 0.8877
Epoch 17/50
94/94 [==============================] - 63s 666ms/step - loss: 0.2082 - accuracy: 0.9254 -
val_loss: 0.3116 - val_accuracy: 0.8931
Epoch 18/50
94/94 [==============================] - 63s 664ms/step - loss: 0.1978 - accuracy: 0.9290 -
val_loss: 0.3113 - val_accuracy: 0.8915
Epoch 19/50
94/94 [==============================] - 62s 658ms/step - loss: 0.1853 - accuracy: 0.9326 -
val_loss: 0.3159 - val_accuracy: 0.8972
Epoch 20/50
94/94 [==============================] - 64s 685ms/step - loss: 0.1796 - accuracy: 0.9355 -
val_loss: 0.3329 - val_accuracy: 0.8987
Epoch 21/50
94/94 [==============================] - 61s 654ms/step - loss: 0.1845 - accuracy: 0.9328 -
val_loss: 0.3316 - val_accuracy: 0.8910
Epoch 22/50
94/94 [==============================] - 63s 672ms/step - loss: 0.1694 - accuracy: 0.9380 -
val_loss: 0.3372 - val_accuracy: 0.8970
Epoch 23/50
```

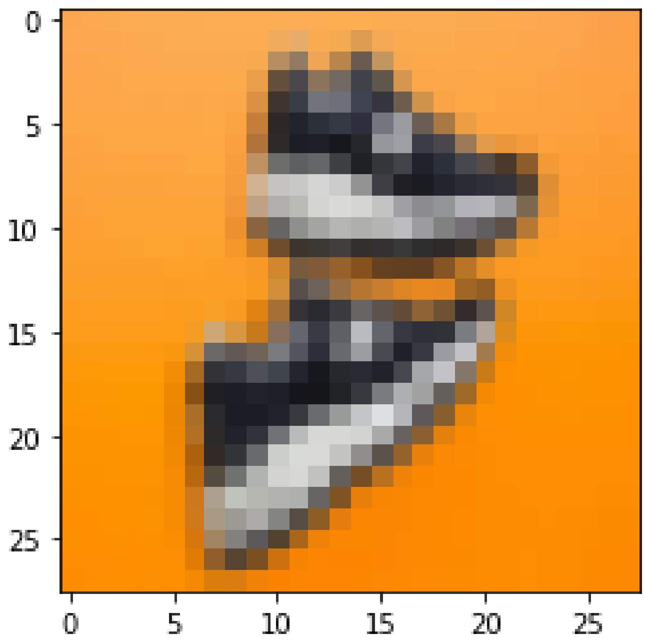
```
94/94 [==============================] - 63s 672ms/step - loss: 0.1640 - accuracy: 0.9413 -
val_loss: 0.3438 - val_accuracy: 0.8969
Epoch 24/50
94/94 [==============================] - 62s 659ms/step - loss: 0.1648 - accuracy: 0.9417 -
val_loss: 0.3252 - val_accuracy: 0.9009
Epoch 25/50
94/94 [==============================] - 64s 683ms/step - loss: 0.1528 - accuracy: 0.9440 -
val_loss: 0.3548 - val_accuracy: 0.8976
Epoch 26/50
94/94 [==============================] - 61s 653ms/step - loss: 0.1456 - accuracy: 0.9482 -
val_loss: 0.3519 - val_accuracy: 0.8999
Epoch 27/50
94/94 [==============================] - 63s 675ms/step - loss: 0.1467 - accuracy: 0.9484 -
val_loss: 0.3565 - val_accuracy: 0.8983
Epoch 28/50
94/94 [==============================] - 63s 668ms/step - loss: 0.1378 - accuracy: 0.9511 -
val_loss: 0.3495 - val_accuracy: 0.8978
Epoch 29/50
94/94 [==============================] - 62s 663ms/step - loss: 0.1372 - accuracy: 0.9507 -
val_loss: 0.3660 - val_accuracy: 0.8987
Epoch 30/50
94/94 [==============================] - 61s 651ms/step - loss: 0.1345 - accuracy: 0.9525 -
val_loss: 0.3611 - val_accuracy: 0.8912
Epoch 31/50
94/94 [==============================] - 62s 664ms/step - loss: 0.1328 - accuracy: 0.9522 -
val_loss: 0.3598 - val_accuracy: 0.8957
Epoch 32/50
94/94 [==============================] - 62s 664ms/step - loss: 0.1325 - accuracy: 0.9530 -
val_loss: 0.3515 - val_accuracy: 0.8970
Epoch 33/50
94/94 [==============================] - 62s 658ms/step - loss: 0.1164 - accuracy: 0.9590 -
val_loss: 0.3830 - val_accuracy: 0.8953
Epoch 34/50
94/94 [==============================] - 63s 667ms/step - loss: 0.1182 - accuracy: 0.9582 -
val_loss: 0.3756 - val_accuracy: 0.9013
Epoch 35/50
94/94 [==============================] - 60s 643ms/step - loss: 0.1129 - accuracy: 0.9596 -
val_loss: 0.4064 - val_accuracy: 0.8992
Epoch 36/50
94/94 [==============================] - 63s 667ms/step - loss: 0.1096 - accuracy: 0.9617 -
val_loss: 0.3911 - val_accuracy: 0.8977
Epoch 37/50
94/94 [==============================] - 62s 665ms/step - loss: 0.1090 - accuracy: 0.9617 -
val_loss: 0.3893 - val_accuracy: 0.8948
Epoch 38/50
94/94 [==============================] - 61s 645ms/step - loss: 0.1111 - accuracy: 0.9611 -
val_loss: 0.3842 - val_accuracy: 0.8956
Epoch 39/50
94/94 [==============================] - 63s 666ms/step - loss: 0.0991 - accuracy: 0.9645 -
val_loss: 0.4147 - val_accuracy: 0.8975
Epoch 40/50
94/94 [==============================] - 62s 662ms/step - loss: 0.1006 - accuracy: 0.9641 -
val_loss: 0.4232 - val_accuracy: 0.8922
Epoch 41/50
94/94 [==============================] - 62s 666ms/step - loss: 0.0937 - accuracy: 0.9668 -
val_loss: 0.4531 - val_accuracy: 0.8948
Epoch 42/50
94/94 [==============================] - 63s 675ms/step - loss: 0.1027 - accuracy: 0.9644 -
val_loss: 0.4250 - val_accuracy: 0.8949
```

```
Epoch 43/50
94/94 [==============================] - 62s 664ms/step - loss: 0.0933 - accuracy: 0.9680 - val_loss: 0.4280 - val_accuracy: 0.8938
Epoch 44/50
94/94 [==============================] - 61s 647ms/step - loss: 0.0940 - accuracy: 0.9667 - val_loss: 0.4213 - val_accuracy: 0.8997
Epoch 45/50
94/94 [==============================] - 63s 668ms/step - loss: 0.0971 - accuracy: 0.9668 - val_loss: 0.4390 - val_accuracy: 0.8937
Epoch 46/50
94/94 [==============================] - 62s 659ms/step - loss: 0.0848 - accuracy: 0.9708 - val_loss: 0.4365 - val_accuracy: 0.8978
Epoch 47/50
94/94 [==============================] - 60s 643ms/step - loss: 0.0845 - accuracy: 0.9699 - val_loss: 0.4532 - val_accuracy: 0.8979
Epoch 48/50
94/94 [==============================] - 64s 679ms/step - loss: 0.0783 - accuracy: 0.9731 - val_loss: 0.4504 - val_accuracy: 0.8984
Epoch 49/50
94/94 [==============================] - 60s 643ms/step - loss: 0.0831 - accuracy: 0.9710 - val_loss: 0.4720 - val_accuracy: 0.9010
Epoch 50/50
94/94 [==============================] - 62s 658ms/step - loss: 0.0786 - accuracy: 0.9733 - val_loss: 0.4479 - val_accuracy: 0.8985
313/313 [==============================] - 5s 16ms/step - loss: 0.5008 - accuracy: 0.8914
[0.5007569193840027, 0.8913999795913696]
```

	precision	recall	f1-score	support
T-shirt/top	0.79	0.88	0.83	1000
Trouser	0.99	0.97	0.98	1000
Pullover	0.81	0.86	0.83	1000
Dress	0.91	0.86	0.88	1000
Coat	0.79	0.87	0.83	1000
Sandal	0.98	0.97	0.98	1000
Shirt	0.77	0.61	0.68	1000
Sneaker	0.95	0.96	0.96	1000
Bag	0.98	0.96	0.97	1000
Ankle boot	0.96	0.97	0.96	1000
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
weighted avg	0.89	0.89	0.89	10000


T-shirt/top 884 4 30 13 3 3 57 1 5 0
Trouser 6 974 0 9 5 1 4 0 1 0
Pullover 20 0 857 9 65 0 49 0 0 0
Dress 39 7 14 860 48 0 31 0 1 0
Coat 1 2 68 18 874 0 36 0 1 0
Sandal 0 0 0 0 0 970 0 21 1 8
Shirt 171 0 82 23 103 0 611 0 10 0
Sneaker 0 0 0 1 0 9 0 959 0 31
Bag 1 1 9 12 6 0 10 2 958 1
Ankle boot 0 0 1 0 0 6 0 25 1 967
T-shirt/top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot
800
600
400
200
0


0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300
350
400



Sneakers

Conclusion:

The CNN model with complex model gives more accuracy & less loss. The above model proves that compare to basic CNN model Complex model gives accurate result. CNN is widely used for object recognition.