Machine Learning

CS-697AB

Assignment 3

Name:FNU Aishwarya Ajay Venkatesha

ID: E425P738

Procedure:

Step 1 : Importing required libraries.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense,Input, Flatten, Bidirectional, BatchNormalization , Dropout, Concatenate
tf.__version__
keras.__version__
```

```
'2.7.0'
```

Step 2 : Load the dataset Fashion_MNIST from keras and divide the test and train dataset from it.

```
[23] #Loading the FASHION_MNIST dataset
     fashion_mnist = keras.datasets.fashion_mnist
     (X_train_all, y_train_all), (X_test, y_test) = fashion_mnist.load_data()
```

Step 3 :

We know that the image data in x_train_all is from 0 to 255 , performing rescaling from 0 to 1.To achieve rescaling , the x_train_all is divided by 255 .
Note that the training set and the testing set should be preprocessed and reshaped in the same way. Mismatch in shape between training and test data leads to errors.

```
#Rescaling image data in x_train_all from (0 to 255) to (0 to 1)
X_train_all , X_test = X_train_all / 255.0 , X_test / 255.0

#Populating x_valid ,x_train, y_valid, y_train
X_valid, X_train = X_train_all[:5000], X_train_all[5000:]
y_valid, y_train = y_train_all[:5000], y_train_all[5000:]
```

Step 4 :

Building a neural network model with batch normalization and dropout layers.

The model has 3 hidden layers and 2 layers with batch normalization; these layers normalize its inputs (here normalizes the previous layers) . Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

Dropout is used to avoid overfitting of data.

```
#Building a neural network with batch normalization and dropout layers
input_data = Input(shape= X_train[0].shape)
hidden_Layer_1 = Dense(30 , activation='relu')(input_data)
model_modified = BatchNormalization()(hidden_Layer_1)
model_modified = Dropout(0.4)(model_modified)

hidden_Layer_2 = Dense(29, activation='relu')(model_modified)

hidden_Layer_3 = Dense(28, activation='relu')(hidden_Layer_2)
model_modified = BatchNormalization()(hidden_Layer_3)
model_modified = Dropout(0.4)(model_modified)

concat = Concatenate(axis=1)
model = concat([input_data, model_modified])
model = Flatten()(model)

output = Dense(10, activation='softmax')(model)
model = keras.models.Model(inputs = [input_data], outputs = [output])
```

Step 5 :

Compile the model

The model is configured before training. While configuring we use the following attributes to compile the model.

Loss function = "sparse_categorical_crossentropy" ; Measured the accuracy of the model while training.

Optimizer = “adam”; Updates based on data it perceives and loss function

Metrics = [“accuracy”]; Monitors the training and testing steps.

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

Step 6 :

Fitting the model with the data.

```
[28] history = model.fit(X_train,y_train, batch_size=72, epochs=15, validation_data=(X_valid, y_valid))

Epoch 1/15
764/764 [==============================] - 8s 9ms/step - loss: 0.6037 - accuracy: 0.7850 - val_loss: 0.3999 - val_accuracy: 0.8536
Epoch 2/15
764/764 [==============================] - 6s 8ms/step - loss: 0.4302 - accuracy: 0.8462 - val_loss: 0.3607 - val_accuracy: 0.8768
Epoch 3/15
764/764 [==============================] - 6s 8ms/step - loss: 0.3957 - accuracy: 0.8572 - val_loss: 0.3548 - val_accuracy: 0.8774
Epoch 4/15
764/764 [==============================] - 7s 9ms/step - loss: 0.3779 - accuracy: 0.8644 - val_loss: 0.3416 - val_accuracy: 0.8814
Epoch 5/15
764/764 [==============================] - 6s 8ms/step - loss: 0.3622 - accuracy: 0.8692 - val_loss: 0.3319 - val_accuracy: 0.8840
Epoch 6/15
764/764 [==============================] - 7s 9ms/step - loss: 0.3571 - accuracy: 0.8702 - val_loss: 0.3471 - val_accuracy: 0.8774
Epoch 7/15
```

Step 7:

Evaluating the model and predicting the accuracy.

```
score = model.evaluate(X_test,y_test, verbose=0)
print('Test Loss: {:.4f}%'.format(score[0]*100))
print('Test Accuracy : {:.4f}%'.format(score[1]*100))
```

```
Test Loss: 32.0652%
Test Accuracy : 89.0400%
```

Step 6 and 7 repeated for epoch values = 20,10,5

epochs=20

```
[39] history = model.fit(X_train,y_train, batch_size=72, epochs=20, validation_data=(X_valid, y_valid))
```

```
Epoch 1/20
764/764 [==============================] - 6s 8ms/step - loss: 0.2712 - accuracy: 0.9008 - val_los
Epoch 2/20
764/764 [==============================] - 6s 8ms/step - loss: 0.2736 - accuracy: 0.9000 - val_los
Epoch 3/20
```

```
[24] score = model.evaluate(X_test,y_test, verbose=0)
     #Results
     print('Test Loss: {:.2f}%'.format(score[0]*100))
     print('Test Accuracy : {:.4f}%'.format(score[1]*100))
```

```
Test Loss: 32.54%
Test Accuracy : 88.6500%
```

epochs = 10

```
[30] history = model.fit(X_train,y_train, batch_size=72, epochs=10, validation_data=(X_valid, y_valid))
```

```
Epoch 1/10
764/764 [==============================] - 7s 9ms/step - loss: 0.3072 - accuracy: 0.8882 - val_loss
Epoch 2/10
764/764 [==============================] - 7s 9ms/step - loss: 0.3024 - accuracy: 0.8883 - val_loss
```

```
[20] score = model.evaluate(X_test,y_test, verbose=0)

     print('Test Loss: {:.2f}%'.format(score[0]*100))
     print('Test Accuracy : {:.4f}%'.format(score[1]*100))
```

```
Test Loss: 33.03%
Test Accuracy : 88.9400%
```

Epochs = 5

```
history = model.fit(X_train,y_train, batch_size=72, epochs=5, validation_data=(X_valid, y_valid))
```

```
score = model.evaluate(X_test,y_test, verbose=0)
print(score[0],score[1])
print('Test Loss: {:.2f}%'.format(score[0]*100))
print('Test Accuracy : {:.4f}%'.format(score[1]*100))
```

```
0.32903847098350525 0.8881000280380249
Test Loss: 32.90%
Test Accuracy : 88.8100%
```

Step 8:

Model Summary is as follows.

```
[15] model.summary()

Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 28, 28)]     0           []

 dense (Dense)                  (None, 28, 30)       870         ['input_1[0][0]']

 batch_normalization (BatchNorm  (None, 28, 30)      120         ['dense[0][0]']
 alization)

 dropout (Dropout)              (None, 28, 30)       0           ['batch_normalization[0][0]']

 dense_1 (Dense)                (None, 28, 29)       899         ['dropout[0][0]']

 dense_2 (Dense)                (None, 28, 28)       840         ['dense_1[0][0]']

 batch_normalization_1 (BatchNo  (None, 28, 28)      112         ['dense_2[0][0]']
 rmalization)

 dropout_1 (Dropout)            (None, 28, 28)       0           ['batch_normalization_1[0][0]'

 concatenate (Concatenate)      (None, 56, 28)       0           ['input_1[0][0]',
                                                                  'dropout_1[0][0]']

 flatten (Flatten)              (None, 1568)         0           ['concatenate[0][0]']

 dense_3 (Dense)                (None, 10)           15690       ['flatten[0][0]']

==================================================================================================
Total params: 18,531
Trainable params: 18,415
Non-trainable params: 116
```

Result Discussion: -

- Batch Normalization and Dropout functions reduce overfitting.
- I did try using different batch sizes for fitting the model but it did not affect the accuracy. Batch size affects the speed/performance depending on memory in the processing unit . Greater the memory , greater can be the batch size and thus the training will be faster .
- With increase in epoch size, the accuracy increases up to a certain threshold beyond which the data can get overfitted . Noticeably in the above executions , epoch size increased from 5 1o 10 , accuracy increased noticeably but at epoch value = 20, accuracy dropped. If we go lower than 5 , it causes underfitting. Below table represents the values achieved for the above example.

Epoch value	Accuracy (with test data)
5	88.8100%
10	88.9400%
15	89.0400%
20	88.6500%