ASSIGNMENT – 4

[3]:
```python
# STEP 1: LOAD THE DATA
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
(X_train, y_train), (X_test, y_test) = mnist.load_data()  # loading
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

[4]:
```python
#STEP 2 : DESIGN ND CREATE MODEL
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, BatchNormalization

def create_model():
    model = Sequential([
        Flatten(input_shape=(28, 28)),
        Dense(300, activation='relu', kernel_initializer='he_normal'),
        BatchNormalization(),
        Dense(100, activation='relu', kernel_initializer='he_normal'),
        BatchNormalization(),
        Dense(10, activation='softmax')
    ])
    return model

model = create_model()


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

[5]:
```python
#STEP 3:TRAIN THE MODEL.
# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8879 - loss: 0.3654 - val_accuracy: 0.9599 - val_loss: 0.1329
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9656 - loss: 0.1124 - val_accuracy: 0.9686 - val_loss: 0.1002
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9747 - loss: 0.0843 - val_accuracy: 0.9704 - val_loss: 0.1019
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9796 - loss: 0.0638 - val_accuracy: 0.9722 - val_loss: 0.0896
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9823 - loss: 0.0546 - val_accuracy: 0.9732 - val_loss: 0.0968
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9837 - loss: 0.0483 - val_accuracy: 0.9755 - val_loss: 0.0845
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9875 - loss: 0.0386 - val_accuracy: 0.9768 - val_loss: 0.0786
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9883 - loss: 0.0340 - val_accuracy: 0.9787 - val_loss: 0.0830
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9893 - loss: 0.0309 - val_accuracy: 0.9787 - val_loss: 0.0828
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9894 - loss: 0.0311 - val_accuracy: 0.9759 - val_loss: 0.0895
```

[6]:
```python
#STEP 4:EVALUATE THE MODEL
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_accuracy)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9753 - loss: 0.0937
Test Accuracy: 0.9775999784469604
```

[7]:
```python
#Step 5: Extend the Model to Have 3 Hidden Layers
def create_extended_model():
    model = Sequential([
        Flatten(input_shape=(28, 28)),
        Dense(300, activation='relu', kernel_initializer='he_normal'),
        BatchNormalization(),
        Dense(200, activation='relu', kernel_initializer='he_normal'),
        BatchNormalization(),
        Dense(100, activation='relu', kernel_initializer='he_normal'),
        BatchNormalization(),
        Dense(10, activation='softmax')
    ])
    return model
extended_model = create_extended_model()
extended_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Train the extended model
history = extended_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
# Evaluate the extended model
test_loss, test_accuracy = extended_model.evaluate(X_test, y_test)
print("Test Accuracy:", test_accuracy)    # print the accuracy
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.8786 - loss: 0.3900 - val_accuracy: 0.9624 - val_loss: 0.1209
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9623 - loss: 0.1186 - val_accuracy: 0.9657 - val_loss: 0.1139
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9711 - loss: 0.0934 - val_accuracy: 0.9732 - val_loss: 0.0942
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9761 - loss: 0.0760 - val_accuracy: 0.9715 - val_loss: 0.0940
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9798 - loss: 0.0634 - val_accuracy: 0.9765 - val_loss: 0.0871
Epoch 6/10
```


```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.8786 - loss: 0.3900 - val_accuracy: 0.9624 - val_loss: 0.1209
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9623 - loss: 0.1186 - val_accuracy: 0.9657 - val_loss: 0.1139
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9711 - loss: 0.0934 - val_accuracy: 0.9732 - val_loss: 0.0942
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9761 - loss: 0.0760 - val_accuracy: 0.9715 - val_loss: 0.0940
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9798 - loss: 0.0634 - val_accuracy: 0.9765 - val_loss: 0.0871
Epoch 6/10
```


```
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9830 - loss: 0.0556 - val_accuracy: 0.9723 - val_loss: 0.0963
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9848 - loss: 0.0452 - val_accuracy: 0.9752 - val_loss: 0.0935
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9872 - loss: 0.0402 - val_accuracy: 0.9758 - val_loss: 0.0967
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9852 - loss: 0.0433 - val_accuracy: 0.9745 - val_loss: 0.0944
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9894 - loss: 0.0321 - val_accuracy: 0.9759 - val_loss: 0.0920
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9708 - loss: 0.1034
Test Accuracy: 0.9768000245094299
```