Artificial Intelligence

Md. Miju Ahmed
ID : 2010676104

December 2024

Assignment 1

1. Build a fully connected neural network (FCNN) and a convolutional neural network (CNN) for classifying 10 classes of images.

```python
from tensorflow.keras.layers import Input, Flatten,Dense,
    Conv2D, MaxPooling2D
from tensorflow.keras.models import Model
import numpy as np
import cv2
```

Build an FCNN for classifying 10 classes

```python
inputs = Input((256,256,3))
x = Flatten()(inputs)
x = Dense(10, activation="relu")(x)
x = Dense(12,activation="sigmoid")(x)
x = Dense(16,activation="relu")(x)
x = Dense(20,activation="relu")(x)
x = Dense(15,activation="relu")(x)
outputs = Dense(10,activation="sigmoid",name='OutputLayer')(x)
model = Model(inputs,outputs,name='FCNN')
model.summary()
```

The output of the upper code is given below.

Model: "FCNN"

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 256, 256, 3)	0
flatten_2 (Flatten)	(None, 196608)	0
dense_5 (Dense)	(None, 10)	1,966,090
dense_6 (Dense)	(None, 12)	132
dense_7 (Dense)	(None, 16)	208
dense_8 (Dense)	(None, 20)	340
dense_9 (Dense)	(None, 15)	315
OutputLayer (Dense)	(None, 10)	160

Total params: 1,967,245 (7.50 MB)

Trainable params: 1,967,245 (7.50 MB)

Non-trainable params: 0 (0.00 B)

Figure 1: FCNN model summary

Build a CNN for classifying 10 classes.

```
inputs = Input((256,256,3))
x = Conv2D(filters=5,kernel_size=(3,3),padding='same',activation='relu')(inputs)
x = Conv2D(filters=8,kernel_size=(5,5),padding='same',activation='relu')(x)
x = Conv2D(filters=6,kernel_size=(3,3),padding='same',activation='relu')(x)
x = Conv2D(filters=9,kernel_size=(3,3),padding='same',activation='relu')(x)
x = Conv2D(filters=7,kernel_size=(3,3),padding='same',activation='relu')(x)
x = Flatten()(x)
outputs = Dense(10,activation='sigmoid',name='OutputLayer')(x)
model = Model(inputs, outputs, name='CNN')
model.summary()
```

The output of the CNN is given below

Model: "CNN"

Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 256, 256, 3)	0
conv2d_5 (Conv2D)	(None, 256, 256, 5)	140
conv2d_6 (Conv2D)	(None, 256, 256, 8)	1,008
conv2d_7 (Conv2D)	(None, 256, 256, 6)	438
conv2d_8 (Conv2D)	(None, 256, 256, 9)	495
conv2d_9 (Conv2D)	(None, 256, 256, 7)	574
flatten_3 (Flatten)	(None, 458752)	0
OutputLayer (Dense)	(None, 10)	4,587,530

Total params: 4,590,185 (17.51 MB)

Trainable params: 4,590,185 (17.51 MB)

Non-trainable params: 0 (0.00 B)

Figure 2: CNN model summary

2. Train and test your FCNN and CNN by the Fashion dataset. Discuss your results by comparing performance between two types of networks.
Import the necessary package.

```
from tensorflow.keras.datasets.fashion_mnist import load_data
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Input,Flatten,Dense,Activation,Conv2D,MaxPooling2D
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt
import numpy as np
```

Display the image:

```python
def display_image(img_set,title_set):
    n=len(title_set)
    for i in range(n):
        plt.subplot(3,3,i+1)
        plt.imshow(img_set[i],cmap='gray')
        plt.title(title_set[i])
    plt.show()
    plt.close()
```

Train and testing the Fashion dataset

```python
(trainX,trainY),(testX,testY)=load_data()
print('trainX.shape: {}, trainY.shape: {}, testX.shape: {}, testY.shape: {})'
    .format(trainX.shape, trainY.shape, testX.shape, testY.shape))
print('trainX.dtype: {}, trainY.dtype: {}, testX.dtype: {}, testY.dtype: {}'
    .format(trainX.dtype, trainY.dtype, testX.dtype, testY.dtype))
print('trainX.Range: {} - {}, testX.Range: {} - {}'
    .format(trainX.max(), trainX.min(), testX.max(), testX.min()))
display_image(trainX[:9], trainY[:9])
```

The output of the Fashion dataset is given below

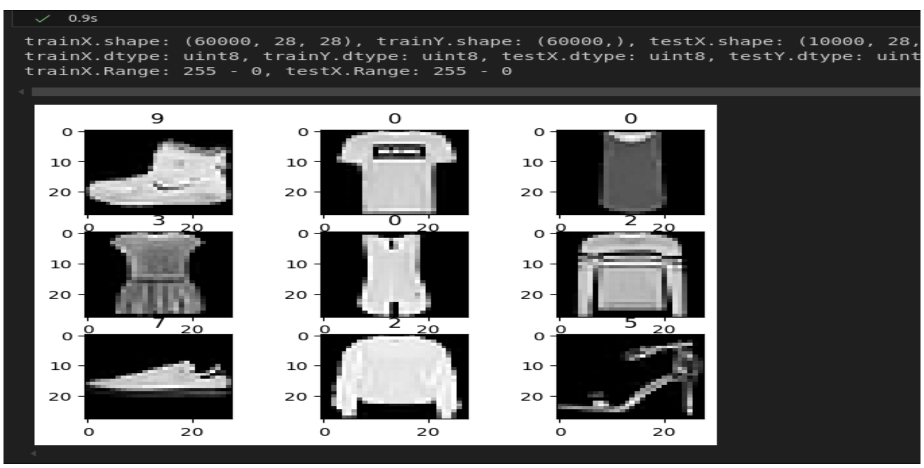

Figure 3: fashion_mnist

Prepared the dataset

```
trainX=trainX.reshape(-1,28,28,1)/256.0
testX=testX.reshape(-1,28,28,1)/256.0

print('trainX.shape: {}, testX.shape: {})'.format(trainX.shape, testX.shape))
print('trainX.dtype: {}, testX.dtype: {}'.format(trainX.dtype, testX.dtype))
print('trainX.Range: {} - {}, testX.Range: {} - {}'.format(trainX.max(), trainX.min(), testX.max(), t

trainY=to_categorical(trainY,num_classes=10)
testY=to_categorical(testY,num_classes=10)
print('trainY.shape: {}, testY.shape: {})'.format(trainY.shape, testY.shape))
print('trainY.dtype: {}, testX.dtype: {}'.format(trainY.dtype, testY.dtype))
print(trainY[:5])
```

Create model using FCNN

```
inputs=Input((28,28,1),name='InputLayer')
x=Flatten()(inputs)
x=Dense(10,activation='relu')(x)
x=Dense(18,activation='relu')(x)
x=Dense(22,activation='relu')(x)
x=Dense(26,activation='relu')(x)
x=Dense(35,activation='relu')(x)
x=Dense(50,activation='relu')(x)
x=Dense(38,activation='relu')(x)
x=Dense(34,activation='relu')(x)
x=Dense(28,activation='relu')(x)
x=Dense(23,activation='relu')(x)

outputs=Dense(10,activation='softmax',name='OutputLayer')(x)
model=Model(inputs,outputs,name="FCNN")
model.summary()
```

The output for this FCNN

Model: "FCNN"

Layer (type)	Output Shape	Param #
InputLayer (InputLayer)	(None, 28, 28, 1)	0
flatten_9 (Flatten)	(None, 784)	0
dense_97 (Dense)	(None, 10)	7,850
dense_98 (Dense)	(None, 18)	198
dense_99 (Dense)	(None, 22)	418
dense_100 (Dense)	(None, 26)	598
dense_101 (Dense)	(None, 35)	945
dense_102 (Dense)	(None, 50)	1,800
dense_103 (Dense)	(None, 38)	1,938
dense_104 (Dense)	(None, 34)	1,326
dense_105 (Dense)	(None, 28)	980
dense_106 (Dense)	(None, 23)	667
OutputLayer (Dense)	(None, 10)	240

Total params: 16,960 (66.25 KB)

Trainable params: 16,960 (66.25 KB)

Non-trainable params: 0 (0.00 B)

Figure 4: Output for FCNN

Compile and fit the FCNN

```
model.compile(loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(trainX,trainY,batch_size=256,validation_split=0.1,epochs=20)
```

The result for this FCNN

```
Epoch 1/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8772 - loss: 0.3317 - val_accuracy: 0.8705 -
Epoch 2/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8823 - loss: 0.3170 - val_accuracy: 0.8688 -
Epoch 3/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8827 - loss: 0.3184 - val_accuracy: 0.8688 -
Epoch 4/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8850 - loss: 0.3113 - val_accuracy: 0.8758 -
Epoch 5/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8843 - loss: 0.3136 - val_accuracy: 0.8502 -
Epoch 6/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8868 - loss: 0.3108 - val_accuracy: 0.8548 -
Epoch 7/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8854 - loss: 0.3133 - val_accuracy: 0.8732 -
Epoch 8/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8862 - loss: 0.3042 - val_accuracy: 0.8722 -
Epoch 9/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8867 - loss: 0.3060 - val_accuracy: 0.8690 -
Epoch 10/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8859 - loss: 0.3084 - val_accuracy: 0.8722 -
Epoch 11/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8844 - loss: 0.3113 - val_accuracy: 0.8623 -
Epoch 12/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8871 - loss: 0.2999 - val_accuracy: 0.8683 -
Epoch 13/20
...
Epoch 19/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8911 - loss: 0.2947 - val_accuracy: 0.8718 -
Epoch 20/20
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8904 - loss: 0.2937 - val_accuracy: 0.8670 -
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

<keras.src.callbacks.history.History at 0x7b7a2bfbaf50>
```

Figure 5: Result for FCNN, Accuracy: 89.04%

Build the CNN

```
inputs=Input((28,28,1),name='InputLayer')
x=Conv2D(filters=8,kernel_size=(5,5),padding='same',activation='relu')(inputs)
x=Conv2D(filters=10,kernel_size=(3,3),padding='same',activation='relu')(x)
# x=Conv2D(filters=8,kernel_size=(7,7),padding='same',activation='relu')(x)
x=MaxPooling2D()(x)
x=Conv2D(filters=10,kernel_size=(5,5),padding='same',activation='relu')(x)
# x=Conv2D(filters=8,kernel_size=(5,5),padding='same',activation='relu')(x)
x=Conv2D(filters=30,kernel_size=(5,5),padding='same',activation='relu')(x)
# x=MaxPooling2D()(x)
x=Conv2D(filters=20,kernel_size=(5,5),padding='same',activation='relu')(x)
x=Conv2D(filters=15,kernel_size=(5,5),padding='same',activation='relu')(x)
x=Conv2D(filters=10,kernel_size=(5,5),padding='same',activation='relu')(x)
x=Flatten()(x)
x=Dense(8,activation='relu')(x)
outputs=Dense(10,activation='relu',name='OutputLayer')(x)
model=Model(inputs,outputs,name='CNN')
```

```
model.summary()
```

The output of the CNN

Model: "CNN"

Layer (type)	Output Shape	Param #
InputLayer (InputLayer)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 28, 28, 8)	208
max_pooling2d (MaxPooling2D)	(None, 14, 14, 8)	0
conv2d_1 (Conv2D)	(None, 14, 14, 10)	2,010
conv2d_2 (Conv2D)	(None, 14, 14, 15)	3,765
conv2d_3 (Conv2D)	(None, 14, 14, 10)	3,760
flatten (Flatten)	(None, 1960)	0
dense (Dense)	(None, 8)	15,688
OutputLayer (Dense)	(None, 10)	90

Total params: 25,521 (99.69 KB)

Trainable params: 25,521 (99.69 KB)

Non-trainable params: 0 (0.00 B)

Figure 6: CNN model summary

Compile the CNN

```
model.compile(loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(trainX,trainY,batch_size=512,validation_split=0.1,epochs=10)
```

The result of the CNN

```
Epoch 1/10
106/106 ━━━━━━━━━━━━━━━━━━━━ 17s 147ms/step - accuracy: 0.1802 - loss: 6.4776 - val_accuracy: 0.3977
Epoch 2/10
106/106 ━━━━━━━━━━━━━━━━━━━━ 18s 167ms/step - accuracy: 0.4329 - loss: nan - val_accuracy: 0.1050 -
Epoch 3/10
106/106 ━━━━━━━━━━━━━━━━━━━━ 18s 165ms/step - accuracy: 0.0990 - loss: nan - val_accuracy: 0.1050 -
Epoch 4/10
106/106 ━━━━━━━━━━━━━━━━━━━━ 18s 167ms/step - accuracy: 0.0995 - loss: nan - val_accuracy: 0.1050 -
Epoch 5/10
106/106 ━━━━━━━━━━━━━━━━━━━━ 18s 166ms/step - accuracy: 0.1004 - loss: nan - val_accuracy: 0.1050 -
Epoch 6/10
106/106 ━━━━━━━━━━━━━━━━━━━━ 18s 166ms/step - accuracy: 0.0998 - loss: nan - val_accuracy: 0.1050 -
Epoch 7/10
106/106 ━━━━━━━━━━━━━━━━━━━━ 19s 176ms/step - accuracy: 0.1008 - loss: nan - val_accuracy: 0.1050 -
Epoch 8/10
106/106 ━━━━━━━━━━━━━━━━━━━━ 19s 179ms/step - accuracy: 0.0979 - loss: nan - val_accuracy: 0.1050 -
Epoch 9/10
106/106 ━━━━━━━━━━━━━━━━━━━━ 18s 169ms/step - accuracy: 0.1006 - loss: nan - val_accuracy: 0.1050 -
Epoch 10/10
106/106 ━━━━━━━━━━━━━━━━━━━━ 18s 168ms/step - accuracy: 0.0982 - loss: nan - val_accuracy: 0.1050 -

<keras.src.callbacks.history.History at 0x7b7a035c7790>
```

Figure 7: Result for CNN, Accuracy: 43%

3. Build a CNN having a pre-trained MobileNet as backbone to classify 10 classes.
Imported necessary files.

```python
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.datasets.cifar10 import load_data
from tensorflow.keras.layers import Input,Dense,Flatten,Activation, Conv2D, MaxPool2D
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model
from tensorflow.keras.backend import clear_session
import matplotlib.pyplot as plt
import numpy as np
```

Display function:

```python
def display_image(img_set, title_set):
    n = len(title_set)
```

```python
    for i in range(n):
        plt.subplot(3,3,i+1)
        plt.imshow(img_set[i],cmap='gray')
        plt.title(title_set[i])
    plt.show()
    plt.close()
```

Train and test the dataset Fashion

```python
(trainX,trainY),(testX,testY) = load_data()
display_image(trainX[:9],trainY[:9])
```

Output of the Fashion dataset

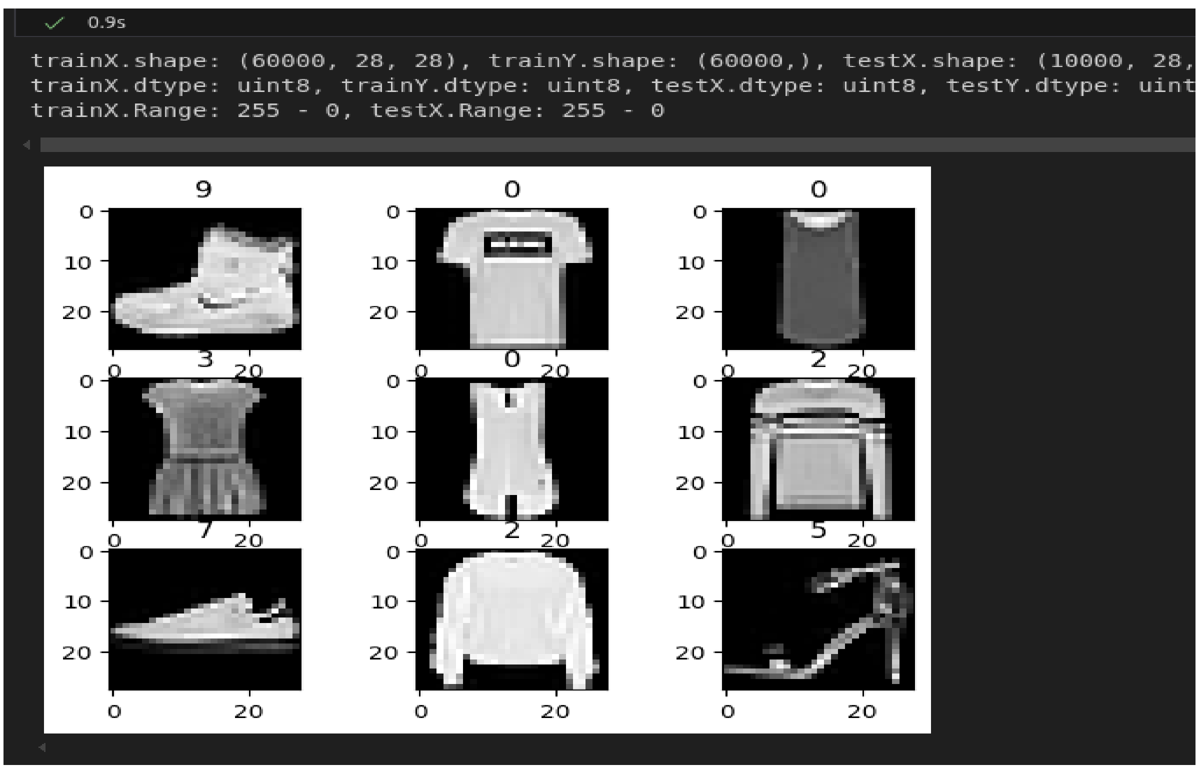

Figure 8: Output of the Fashion dataset

Load the MobileNet Model

```python
mobilenet_model = mobilenet.MobileNet(weights='imagenet', include_top=False, input_shape=(32,32, 3))
inputs = mobilenet_model.input
x = mobilenet_model.output

x = Conv2D(filters=8,kernel_size=(5,5),padding='same',activation='relu')(x)
x = Conv2D(filters=16,kernel_size=(5,5),padding='same',activation='relu')(x)
x = MaxPooling2D()(x)
x = Conv2D(filters=8,kernel_size=(5,5),padding='same',activation='relu')(x)

x = Flatten()(x)
x = Dense(15,activation='relu')(x)

output = Dense(10,activation='softmax')(x)
model = Model(inputs,output,name='MobileNet')
model.summary()
```

Summary of the MobileNet Model

Model: "MobileNet"

Layer (type)	Output Shape	Param #
input_layer_34 (InputLayer)	(None, 32, 32, 3)	0
conv1 (Conv2D)	(None, 16, 16, 32)	864
conv1_bn (BatchNormalization)	(None, 16, 16, 32)	128
conv1_relu (ReLU)	(None, 16, 16, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, 16, 16, 32)	288
conv_dw_1_bn (BatchNormalization)	(None, 16, 16, 32)	128
conv_dw_1_relu (ReLU)	(None, 16, 16, 32)	0
conv_pw_1 (Conv2D)	(None, 16, 16, 64)	2,048
conv_pw_1_bn (BatchNormalization)	(None, 16, 16, 64)	256
conv_pw_1_relu (ReLU)	(None, 16, 16, 64)	0
conv_pad_2 (ZeroPadding2D)	(None, 17, 17, 64)	0
conv_dw_2 (DepthwiseConv2D)	(None, 8, 8, 64)	576
conv_dw_2_bn (BatchNormalization)	(None, 8, 8, 64)	256
conv_dw_2_relu (ReLU)	(None, 8, 8, 64)	0
conv_pw_2 (Conv2D)	(None, 8, 8, 128)	8,192
conv_pw_2_bn (BatchNormalization)	(None, 8, 8, 128)	512
conv_pw_2_relu (ReLU)	(None, 8, 8, 128)	0

Figure 9: MobileNet

conv_dw_4_relu (ReLU)	(None, 4, 4, 128)	0
conv_pw_4 (Conv2D)	(None, 4, 4, 256)	32,768
conv_pw_4_bn (BatchNormalization)	(None, 4, 4, 256)	1,024
conv_pw_4_relu (ReLU)	(None, 4, 4, 256)	0
conv_dw_5 (DepthwiseConv2D)	(None, 4, 4, 256)	2,304
conv_dw_5_bn (BatchNormalization)	(None, 4, 4, 256)	1,024
conv_dw_5_relu (ReLU)	(None, 4, 4, 256)	0
conv_pw_5 (Conv2D)	(None, 4, 4, 256)	65,536
conv_pw_5_bn (BatchNormalization)	(None, 4, 4, 256)	1,024
conv_pw_5_relu (ReLU)	(None, 4, 4, 256)	0
conv_pad_6 (ZeroPadding2D)	(None, 5, 5, 256)	0
conv_dw_6 (DepthwiseConv2D)	(None, 2, 2, 256)	2,304
conv_dw_6_bn (BatchNormalization)	(None, 2, 2, 256)	1,024
conv_dw_6_relu (ReLU)	(None, 2, 2, 256)	0
conv_pw_6 (Conv2D)	(None, 2, 2, 512)	131,072
conv_pw_6_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_pw_6_relu (ReLU)	(None, 2, 2, 512)	0
conv_dw_7 (DepthwiseConv2D)	(None, 2, 2, 512)	4,608

Figure 10: MobileNet

conv_pw_7 (Conv2D)	(None, 2, 2, 512)	262,144
conv_pw_7_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_pw_7_relu (ReLU)	(None, 2, 2, 512)	0
conv_dw_8 (DepthwiseConv2D)	(None, 2, 2, 512)	4,608
conv_dw_8_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_dw_8_relu (ReLU)	(None, 2, 2, 512)	0
conv_pw_8 (Conv2D)	(None, 2, 2, 512)	262,144
conv_pw_8_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_pw_8_relu (ReLU)	(None, 2, 2, 512)	0
conv_dw_9 (DepthwiseConv2D)	(None, 2, 2, 512)	4,608
conv_dw_9_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_dw_9_relu (ReLU)	(None, 2, 2, 512)	0
conv_pw_9 (Conv2D)	(None, 2, 2, 512)	262,144
conv_pw_9_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_pw_9_relu (ReLU)	(None, 2, 2, 512)	0
conv_dw_10 (DepthwiseConv2D)	(None, 2, 2, 512)	4,608
conv_dw_10_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_dw_10_relu (ReLU)	(None, 2, 2, 512)	0

Figure 11: MobileNet

conv_pw_11_relu (ReLU)	(None, 2, 2, 512)	0
conv_pad_12 (ZeroPadding2D)	(None, 3, 3, 512)	0
conv_dw_12 (DepthwiseConv2D)	(None, 1, 1, 512)	4,608
conv_dw_12_bn (BatchNormalization)	(None, 1, 1, 512)	2,048
conv_dw_12_relu (ReLU)	(None, 1, 1, 512)	0
conv_pw_12 (Conv2D)	(None, 1, 1, 1024)	524,288
conv_pw_12_bn (BatchNormalization)	(None, 1, 1, 1024)	4,096
conv_pw_12_relu (ReLU)	(None, 1, 1, 1024)	0
conv_dw_13 (DepthwiseConv2D)	(None, 1, 1, 1024)	9,216
conv_dw_13_bn (BatchNormalization)	(None, 1, 1, 1024)	4,096
conv_dw_13_relu (ReLU)	(None, 1, 1, 1024)	0
conv_pw_13 (Conv2D)	(None, 1, 1, 1024)	1,048,576
conv_pw_13_bn (BatchNormalization)	(None, 1, 1, 1024)	4,096
conv_pw_13_relu (ReLU)	(None, 1, 1, 1024)	0
flatten_19 (Flatten)	(None, 1024)	0
dense_35 (Dense)	(None, 15)	15,375
dense_36 (Dense)	(None, 10)	160

Total params: 3,244,399 (12.38 MB)
Trainable params: 3,222,511 (12.29 MB)
Non-trainable params: 21,888 (85.50 KB)

Figure 12: MobileNet

4. Train and test your CNN having a pre-trained MobileNet as backbone to classify images of the CIFAR-10 dataset. Discuss your results by comparing performance between transfer learning + fine-tuning and only transfer learning.

Below given the code.

Imported necessary files.

```
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.datasets.cifar10 import load_data
from tensorflow.keras.layers import Input,Dense,Flatten,Activation, Conv2D, MaxPool2D
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model
from tensorflow.keras.backend import clear_session
import matplotlib.pyplot as plt
import numpy as np
```

Display function:

```
def display_image(img_set, title_set):
    n = len(title_set)
    for i in range(n):
        plt.subplot(3,3,i+1)
        plt.imshow(img_set[i],cmap='gray')
        plt.title(title_set[i])
    plt.show()
    plt.close()
```

Train and test the dataset CIFAR10

```
(trainX,trainY),(testX,testY) = load_data()
display_image(trainX[:9],trainY[:9])
```

Output of the CIFAR10 dataset

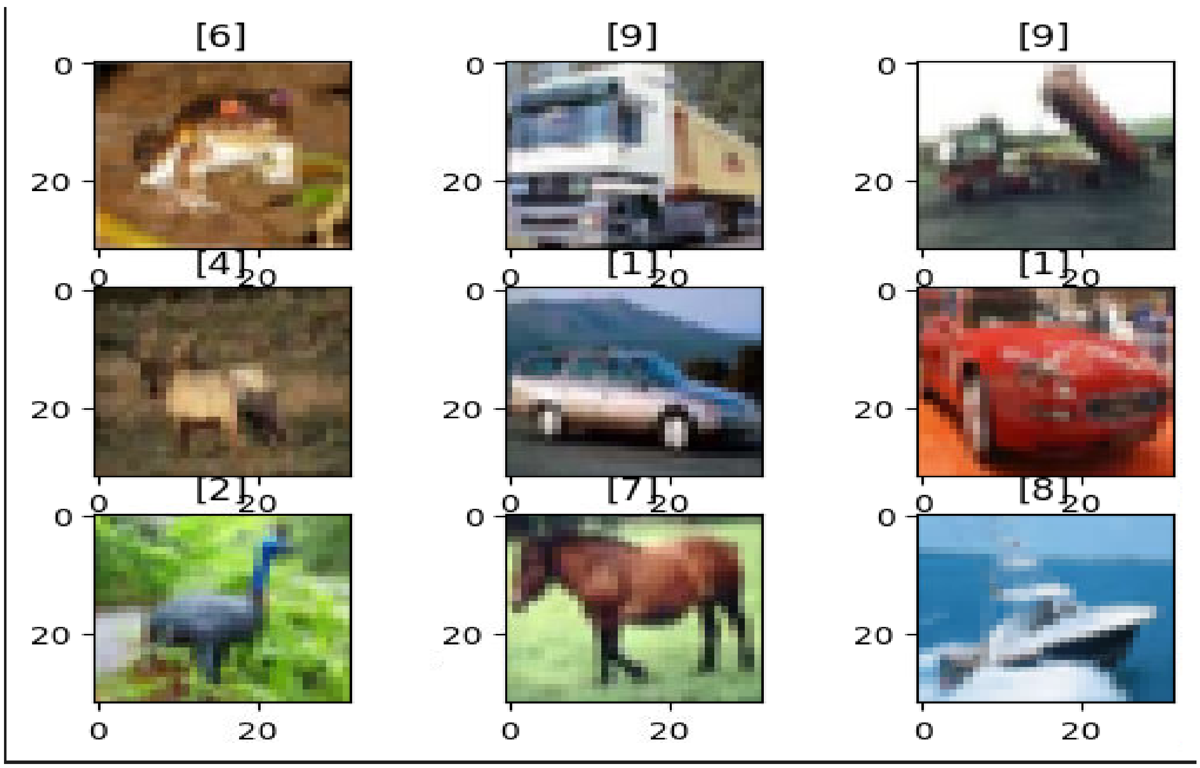

Figure 13: Output of the CIFAR10 dataset

Prepared the CIFAR10 dataset

```
# trainX=np.expand_dims(trainX,axis=-1)
# testX = np.expand_dims(testX,axis=-1)
trainX = trainX/255.0
testX = testX/255.0

trainX = np.squeeze(trainX)
testX = np.squeeze(testX)
print("Train shape:", trainX.shape)
print("Test shape:", testX.shape)

trainY = to_categorical(trainY,num_classes=10)
testY = to_categorical(testY,num_classes=10)
```

Load the MobileNet Model and create the CNN

```python
mobilenet_model = MobileNet(weights=None, include_top=False, input_shape=(32,32,3))
inputs = mobilenet_model.input
x = mobilenet_model.output

x = Conv2D(filters=8,kernel_size=(5,5),padding='same',activation='relu')(x)
x = Conv2D(filters=16,kernel_size=(5,5),padding='same',activation='relu')(x)

x = Flatten()(x)
x = Dense(16,activation='relu')(x)
outputs = Dense(10,activation='softmax',name='OutputLayer')(x)
model = Model(inputs,outputs,name='mobilenet_cifar10')
# model.summary()
```

Summary of the MobileNet Model

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 32, 32, 3)	0
conv1 (Conv2D)	(None, 16, 16, 32)	864
conv1_bn (BatchNormalization)	(None, 16, 16, 32)	128
conv1_relu (ReLU)	(None, 16, 16, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, 16, 16, 32)	288
conv_dw_1_bn (BatchNormalization)	(None, 16, 16, 32)	128
conv_dw_1_relu (ReLU)	(None, 16, 16, 32)	0
conv_pw_1 (Conv2D)	(None, 16, 16, 64)	2,048
conv_pw_1_bn (BatchNormalization)	(None, 16, 16, 64)	256
conv_pw_1_relu (ReLU)	(None, 16, 16, 64)	0
conv_pad_2 (ZeroPadding2D)	(None, 17, 17, 64)	0
conv_dw_2 (DepthwiseConv2D)	(None, 8, 8, 64)	576
conv_dw_2_bn (BatchNormalization)	(None, 8, 8, 64)	256
conv_dw_2_relu (ReLU)	(None, 8, 8, 64)	0
conv_pw_2 (Conv2D)	(None, 8, 8, 128)	8,192
conv_pw_2_bn (BatchNormalization)	(None, 8, 8, 128)	512
conv_pw_2_relu (ReLU)	(None, 8, 8, 128)	0
conv_dw_3 (DepthwiseConv2D)	(None, 8, 8, 128)	1,152
conv_dw_3_bn (BatchNormalization)	(None, 8, 8, 128)	512
conv_dw_3_relu (ReLU)	(None, 8, 8, 128)	0
conv_pw_3 (Conv2D)	(None, 8, 8, 128)	16,384

Figure 14: Model summary

conv_pw_2_bn (BatchNormalization)	(None, 8, 8, 128)	512
conv_pw_2_relu (ReLU)	(None, 8, 8, 128)	0
conv_dw_3 (DepthwiseConv2D)	(None, 8, 8, 128)	1,152
conv_dw_3_bn (BatchNormalization)	(None, 8, 8, 128)	512
conv_dw_3_relu (ReLU)	(None, 8, 8, 128)	0
conv_pw_3 (Conv2D)	(None, 8, 8, 128)	16,384
conv_pw_3_bn (BatchNormalization)	(None, 8, 8, 128)	512
conv_pw_3_relu (ReLU)	(None, 8, 8, 128)	0
conv_pad_4 (ZeroPadding2D)	(None, 9, 9, 128)	0
conv_dw_4 (DepthwiseConv2D)	(None, 4, 4, 128)	1,152
conv_dw_4_bn (BatchNormalization)	(None, 4, 4, 128)	512
conv_dw_4_relu (ReLU)	(None, 4, 4, 128)	0
conv_pw_4 (Conv2D)	(None, 4, 4, 256)	32,768
conv_pw_4_bn (BatchNormalization)	(None, 4, 4, 256)	1,024
conv_pw_4_relu (ReLU)	(None, 4, 4, 256)	0
conv_dw_5 (DepthwiseConv2D)	(None, 4, 4, 256)	2,304
conv_dw_5_bn (BatchNormalization)	(None, 4, 4, 256)	1,024
conv_dw_5_relu (ReLU)	(None, 4, 4, 256)	0
conv_pw_5 (Conv2D)	(None, 4, 4, 256)	65,536
conv_pw_5_bn (BatchNormalization)	(None, 4, 4, 256)	1,024
conv_pw_5_relu (ReLU)	(None, 4, 4, 256)	0
conv_pad_6 (ZeroPadding2D)	(None, 5, 5, 256)	0

Figure 15: Model summary

conv_pad_6 (ZeroPadding2D)	(None, 5, 5, 256)	0
conv_dw_6 (DepthwiseConv2D)	(None, 2, 2, 256)	2,304
conv_dw_6_bn (BatchNormalization)	(None, 2, 2, 256)	1,024
conv_dw_6_relu (ReLU)	(None, 2, 2, 256)	0
conv_pw_6 (Conv2D)	(None, 2, 2, 512)	131,072
conv_pw_6_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_pw_6_relu (ReLU)	(None, 2, 2, 512)	0
conv_dw_7 (DepthwiseConv2D)	(None, 2, 2, 512)	4,608
conv_dw_7_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_dw_7_relu (ReLU)	(None, 2, 2, 512)	0
conv_pw_7 (Conv2D)	(None, 2, 2, 512)	262,144
conv_pw_7_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_pw_7_relu (ReLU)	(None, 2, 2, 512)	0
conv_dw_8 (DepthwiseConv2D)	(None, 2, 2, 512)	4,608
conv_dw_8_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_dw_8_relu (ReLU)	(None, 2, 2, 512)	0
conv_pw_8 (Conv2D)	(None, 2, 2, 512)	262,144
conv_pw_8_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_pw_8_relu (ReLU)	(None, 2, 2, 512)	0
conv_dw_9 (DepthwiseConv2D)	(None, 2, 2, 512)	4,608
conv_dw_9_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_dw_9_relu (ReLU)	(None, 2, 2, 512)	0

Figure 16: Model summary

conv_pw_9 (Conv2D)	(None, 2, 2, 512)	262,144
conv_pw_9_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_pw_9_relu (ReLU)	(None, 2, 2, 512)	0
conv_dw_10 (DepthwiseConv2D)	(None, 2, 2, 512)	4,608
conv_dw_10_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_dw_10_relu (ReLU)	(None, 2, 2, 512)	0
conv_pw_10 (Conv2D)	(None, 2, 2, 512)	262,144
conv_pw_10_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_pw_10_relu (ReLU)	(None, 2, 2, 512)	0
conv_dw_11 (DepthwiseConv2D)	(None, 2, 2, 512)	4,608
conv_dw_11_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_dw_11_relu (ReLU)	(None, 2, 2, 512)	0
conv_pw_11 (Conv2D)	(None, 2, 2, 512)	262,144
conv_pw_11_bn (BatchNormalization)	(None, 2, 2, 512)	2,048
conv_pw_11_relu (ReLU)	(None, 2, 2, 512)	0
conv_pad_12 (ZeroPadding2D)	(None, 3, 3, 512)	0
conv_dw_12 (DepthwiseConv2D)	(None, 1, 1, 512)	4,608
conv_dw_12_bn (BatchNormalization)	(None, 1, 1, 512)	2,048
conv_dw_12_relu (ReLU)	(None, 1, 1, 512)	0
conv_pw_12 (Conv2D)	(None, 1, 1, 1024)	524,288
conv_pw_12_bn (BatchNormalization)	(None, 1, 1, 1024)	4,096
conv_pw_12_relu (ReLU)	(None, 1, 1, 1024)	0

Figure 17: Model summary

conv_pw_12_bn (BatchNormalization)	(None, 1, 1, 1024)	4,096
conv_pw_12_relu (ReLU)	(None, 1, 1, 1024)	0
conv_dw_13 (DepthwiseConv2D)	(None, 1, 1, 1024)	9,216
conv_dw_13_bn (BatchNormalization)	(None, 1, 1, 1024)	4,096
conv_dw_13_relu (ReLU)	(None, 1, 1, 1024)	0
conv_pw_13 (Conv2D)	(None, 1, 1, 1024)	1,048,576
conv_pw_13_bn (BatchNormalization)	(None, 1, 1, 1024)	4,096
conv_pw_13_relu (ReLU)	(None, 1, 1, 1024)	0
conv2d_6 (Conv2D)	(None, 1, 1, 8)	204,808
conv2d_7 (Conv2D)	(None, 1, 1, 16)	3,216
flatten_3 (Flatten)	(None, 16)	0
dense_3 (Dense)	(None, 16)	272
OutputLayer (Dense)	(None, 10)	170

Total params: 3,437,330 (13.11 MB)

Trainable params: 3,415,442 (13.03 MB)

Non-trainable params: 21,888 (85.50 KB)

Figure 18: Model summary

Compile and fit the model

```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(trainX,trainY,batch_size=128,validation_split=0.1,validation_data=(testX,testY),epochs=10)
```

Output of the model

```
Epoch 1/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 177s 429ms/step - accuracy: 0.6965 - loss: 0.8713 - val_accuracy: 0.6094 - val_loss: 1.3134
Epoch 2/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 169s 432ms/step - accuracy: 0.7266 - loss: 0.8065 - val_accuracy: 0.6137 - val_loss: 1.2325
Epoch 3/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 174s 444ms/step - accuracy: 0.7403 - loss: 0.7661 - val_accuracy: 0.6233 - val_loss: 1.3571
Epoch 4/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 161s 412ms/step - accuracy: 0.7567 - loss: 0.7221 - val_accuracy: 0.6380 - val_loss: 1.3567
Epoch 5/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 169s 433ms/step - accuracy: 0.7748 - loss: 0.6731 - val_accuracy: 0.6615 - val_loss: 1.1323
Epoch 6/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 172s 440ms/step - accuracy: 0.7863 - loss: 0.6414 - val_accuracy: 0.6459 - val_loss: 1.1945
Epoch 7/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 157s 400ms/step - accuracy: 0.7955 - loss: 0.6149 - val_accuracy: 0.6340 - val_loss: 1.2985
Epoch 8/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 156s 399ms/step - accuracy: 0.7992 - loss: 0.6119 - val_accuracy: 0.6423 - val_loss: 1.3242
Epoch 9/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 161s 410ms/step - accuracy: 0.8121 - loss: 0.5771 - val_accuracy: 0.6689 - val_loss: 1.2140
Epoch 10/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 173s 442ms/step - accuracy: 0.8296 - loss: 0.5222 - val_accuracy: 0.6474 - val_loss: 1.3892

<keras.src.callbacks.history.History at 0x760ef6563a00>
```

Figure 19: Result of the model, Accuracy : 83.96%

Freezing

```
for layer in model.layers[:-4]:
  layer.trainable = False
model.summary(show_trainable = True)
```

Output of freezing

conv_pw_13 (Conv2D)	(None, 1, 1, 1024)	1,048,576	N
conv_pw_13_bn (BatchNormalization)	(None, 1, 1, 1024)	4,096	N
conv_pw_13_relu (ReLU)	(None, 1, 1, 1024)	0	-
conv2d_6 (Conv2D)	(None, 1, 1, 8)	204,808	N
conv2d_7 (Conv2D)	(None, 1, 1, 16)	3,216	Y
flatten_3 (Flatten)	(None, 16)	0	-
dense_3 (Dense)	(None, 16)	272	Y
OutputLayer (Dense)	(None, 10)	170	Y

Total params: 3,437,330 (13.11 MB)

Trainable params: 3,658 (14.29 KB)

Non-trainable params: 3,433,672 (13.10 MB)

Figure 20: Freez

Transfer leaning - 1

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(trainX, trainY, validation_split=0.1, epochs=10)
```

Output of transfer learning - 1

```
Epoch 1/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 21s 14ms/step - accuracy: 0.0995 - loss: 2.3027 - val_accuracy: 0.0986 - val_loss: 2.3028
Epoch 2/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.0992 - loss: 2.3028 - val_accuracy: 0.0950 - val_loss: 2.3028
Epoch 3/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 20s 14ms/step - accuracy: 0.0991 - loss: 2.3027 - val_accuracy: 0.0970 - val_loss: 2.3027
Epoch 4/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.0988 - loss: 2.3027 - val_accuracy: 0.0976 - val_loss: 2.3028
Epoch 5/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 19s 14ms/step - accuracy: 0.0996 - loss: 2.3027 - val_accuracy: 0.1058 - val_loss: 2.3027
Epoch 6/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.0995 - loss: 2.3027 - val_accuracy: 0.0970 - val_loss: 2.3027
Epoch 7/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 18s 12ms/step - accuracy: 0.0996 - loss: 2.3027 - val_accuracy: 0.0950 - val_loss: 2.3028
Epoch 8/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.1014 - loss: 2.3027 - val_accuracy: 0.0970 - val_loss: 2.3029
Epoch 9/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.1001 - loss: 2.3027 - val_accuracy: 0.0976 - val_loss: 2.3029
Epoch 10/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.1006 - loss: 2.3027 - val_accuracy: 0.0970 - val_loss: 2.3031
```

Figure 21: TL-1, Accuracy : 10.06%

Transfer learning - 2

```python
for layer in model.layers[-5:-1]:
    layer.trainable = True
# model.summary(show_trainable=True)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(trainX, trainY, validation_split=0.1, epochs=10)
```

Output of transfer learning - 2

```
Epoch 1/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 38s 25ms/step - accuracy: 0.0995 - loss: 2.3027 - val_accuracy: 0.0970 - val_loss: 2.3029
Epoch 2/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 36s 25ms/step - accuracy: 0.0990 - loss: 2.3027 - val_accuracy: 0.0976 - val_loss: 2.3027
Epoch 3/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 36s 25ms/step - accuracy: 0.0968 - loss: 2.3027 - val_accuracy: 0.1024 - val_loss: 2.3032
Epoch 4/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 36s 25ms/step - accuracy: 0.0968 - loss: 2.3028 - val_accuracy: 0.0976 - val_loss: 2.3029
Epoch 5/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 36s 25ms/step - accuracy: 0.1001 - loss: 2.3027 - val_accuracy: 0.0950 - val_loss: 2.3030
Epoch 6/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 35s 25ms/step - accuracy: 0.0995 - loss: 2.3027 - val_accuracy: 0.0976 - val_loss: 2.3027
Epoch 7/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 34s 24ms/step - accuracy: 0.1011 - loss: 2.3027 - val_accuracy: 0.0986 - val_loss: 2.3029
Epoch 8/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 35s 25ms/step - accuracy: 0.0988 - loss: 2.3027 - val_accuracy: 0.1038 - val_loss: 2.3026
Epoch 9/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 35s 25ms/step - accuracy: 0.0981 - loss: 2.3027 - val_accuracy: 0.0976 - val_loss: 2.3027
Epoch 10/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 35s 25ms/step - accuracy: 0.1012 - loss: 2.3027 - val_accuracy: 0.0986 - val_loss: 2.3028
```

Figure 22: TL-2, Accuracy : 10.12%

Transfer learning - 3

```
for layer in model.layers[-10:-4]:
    layer.trainable = True
# model.summary(show_trainable=True)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(trainX, trainY, validation_split=0.1, epochs=10)
```

Output of transfer learning - 3

```
Epoch 1/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 110s 76ms/step - accuracy: 0.0993 - loss: 2.3027 - val_accuracy: 0.1024 - val_loss: 2.3029
Epoch 2/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 102s 73ms/step - accuracy: 0.1003 - loss: 2.3027 - val_accuracy: 0.0958 - val_loss: 2.3029
Epoch 3/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 141s 72ms/step - accuracy: 0.0974 - loss: 2.3028 - val_accuracy: 0.0970 - val_loss: 2.3027
Epoch 4/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 102s 72ms/step - accuracy: 0.0999 - loss: 2.3028 - val_accuracy: 0.0958 - val_loss: 2.3030
Epoch 5/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 102s 72ms/step - accuracy: 0.0990 - loss: 2.3027 - val_accuracy: 0.1038 - val_loss: 2.3028
Epoch 6/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 102s 72ms/step - accuracy: 0.0997 - loss: 2.3027 - val_accuracy: 0.0970 - val_loss: 2.3029
Epoch 7/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 101s 72ms/step - accuracy: 0.0991 - loss: 2.3027 - val_accuracy: 0.1038 - val_loss: 2.3028
Epoch 8/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 102s 73ms/step - accuracy: 0.0981 - loss: 2.3027 - val_accuracy: 0.0950 - val_loss: 2.3028
Epoch 9/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 102s 73ms/step - accuracy: 0.0992 - loss: 2.3028 - val_accuracy: 0.0958 - val_loss: 2.3031
Epoch 10/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 102s 73ms/step - accuracy: 0.0984 - loss: 2.3027 - val_accuracy: 0.0976 - val_loss: 2.3029
```

Figure 23: TL-3, Accuracy : 10.03%

Transfer learning - 4

```python
for layer in model.layers[:-8]:
    layer.trainable = True
# model.summary(show_trainable=True)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(trainX, trainY, validation_split=0.1, epochs=10)
```

Output of transfer learning - 4

```
Epoch 1/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 196s 132ms/step - accuracy: 0.1592 - loss: 2.1234 - val_accuracy: 0.1472 - val_loss: 4.2380
Epoch 2/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 186s 132ms/step - accuracy: 0.2428 - loss: 1.8828 - val_accuracy: 0.2476 - val_loss: 1.9850
Epoch 3/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 186s 132ms/step - accuracy: 0.2962 - loss: 1.7749 - val_accuracy: 0.3652 - val_loss: 1.7146
Epoch 4/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 187s 133ms/step - accuracy: 0.4091 - loss: 1.5441 - val_accuracy: 0.4138 - val_loss: 1.5209
Epoch 5/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 187s 133ms/step - accuracy: 0.4816 - loss: 1.4000 - val_accuracy: 0.4888 - val_loss: 1.4233
Epoch 6/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 186s 132ms/step - accuracy: 0.5294 - loss: 1.3001 - val_accuracy: 0.5492 - val_loss: 1.2873
Epoch 7/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 186s 132ms/step - accuracy: 0.5681 - loss: 1.2110 - val_accuracy: 0.5454 - val_loss: 1.2532
Epoch 8/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 189s 134ms/step - accuracy: 0.6005 - loss: 1.1293 - val_accuracy: 0.5484 - val_loss: 1.2441
Epoch 9/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 189s 134ms/step - accuracy: 0.6217 - loss: 1.0737 - val_accuracy: 0.5632 - val_loss: 1.2708
Epoch 10/10
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 180s 128ms/step - accuracy: 0.6445 - loss: 1.0268 - val_accuracy: 0.6222 - val_loss: 1.1193

<keras.src.callbacks.history.History at 0x760edac17010>
```

Figure 24: TL-4, Accuracy : 64.45%