Md:Israil Hosen

Roll:2010876110

Artificial Intelligence Lab Assignment

Answer To The Question No: 1

Building a fully connected neural network(FCCN) and a convolutional neural network(CNN) for classifying 10 classes of image are given below:

The Fully Connected Neural Network(FCCN) for 10 classes is:

```
inputs = Input(shape=(28, 28, 1), name = 'InputLayer')
x = Flatten()(inputs)
x = Dense(512, activation = 'relu')(x)
Dropout(0.3)  # Dropout to prevent overfitting
x = Dense(256, activation = 'relu')(x)
Dropout(0.3)
x = Dense(128, activation = 'relu')(x)
Dropout(0.3)
x = Dense(64, activation = 'relu')(x)
Dropout(0.3)
x = Dense(32, activation = 'relu')(x)
Dropout(0.3)
x = Dense(16, activation = 'relu')(x)
Dropout(0.3)
outputs = Dense(10, name = 'OutputLayer', activation = 'softmax')(x
    )
model = Model(inputs, outputs, name = 'Multi-Class-Classifier')
model.summary()
```

Here is the model summary looks like:

Model: "Multi-Class-Classifier"

Layer (type)	Output Shape	Param #
InputLayer (InputLayer)	(None, 28, 28, 1)	0
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 512)	401,920
dense_1 (Dense)	(None, 256)	131,328
dense_2 (Dense)	(None, 128)	32,896
dense_3 (Dense)	(None, 64)	8,256
dense_4 (Dense)	(None, 32)	2,080
dense_5 (Dense)	(None, 16)	528
OutputLayer (Dense)	(None, 10)	170

Total params: 577,178 (2.20 MB)

Trainable params: 577,178 (2.20 MB)

Non-trainable params: 0 (0.00 B)

Figure 1: model summary of FCCN

The Convolutional Neural Network(CNN) for 10 classes is:

```
inputs = Input(shape=(28,28,1),name='InputLayer')

#first convolutional layer block
x = Conv2D(filters=32,kernel_size=(3,3),activation='relu')(inputs)
x = MaxPooling2D(pool_size=(2,2))(x)
x = Dropout(0.3)(x)

#second convolution block
x = Conv2D(filters=64,kernel_size=(3,3,),activation='relu')(x)
x = MaxPooling2D(pool_size=(2,2))(x)
x = Dropout(0.3)(x)

#third convolution block
x = Conv2D(filters=28,kernel_size=(3,3),activation='relu')(x)
x = MaxPooling2D(pool_size=(2,2))(x)
x = Dropout(0.3)(x)

#fully connected layers
x = Flatten()(x)
x = Dense(128,activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(64,activation='relu')(x)
x = Dropout(0.3)(x)

#output layer
outputs = Dense(10,activation='softmax',name='outputLayer')(x)
model = Model(inputs=inputs,outputs=outputs,name='CNN-Multi-
    Classifier')
model.summary()
```

Here is the model sumamry looks like:

Model: "CNN-Multi-Classifier"

Layer (type)	Output Shape	Param #
InputLayer (InputLayer)	(None, 28, 28, 1)	0
conv2d_9 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_8 (MaxPooling2D)	(None, 14, 14, 32)	0
dropout_11 (Dropout)	(None, 14, 14, 32)	0
conv2d_10 (Conv2D)	(None, 14, 14, 64)	18,496
max_pooling2d_9 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout_12 (Dropout)	(None, 7, 7, 64)	0
conv2d_11 (Conv2D)	(None, 7, 7, 28)	16,156
max_pooling2d_10 (MaxPooling2D)	(None, 3, 3, 28)	0
dropout_13 (Dropout)	(None, 3, 3, 28)	0
flatten_2 (Flatten)	(None, 252)	0
dense_3 (Dense)	(None, 128)	32,384
dropout_14 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 64)	8,256
dropout_15 (Dropout)	(None, 64)	0
outputLayer (Dense)	(None, 10)	650

Total params: 76,262 (297.90 KB)

Trainable params: 76,262 (297.90 KB)

Figure 2: model summary of CNN

Answer To The Question No: 2

After Training and Testing the FCNN and CNN by the Fashion Dataset both the model performance are given below:

At first we will show the test and train performance of FCCN model by the Fashion-Dataset which are:

After Training the models

FCNN model

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
    metrics=['accuracy'])
history = model.fit(trainX, trainY, batch_size=64, epochs=10,
    validation_split=0.1)
```

Output after training the model is:

```
Epoch 1/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.7202 - loss: 0.8457 - val_accuracy: 0.8490 - val_loss: 0.4252
Epoch 2/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8602 - loss: 0.3880 - val_accuracy: 0.8627 - val_loss: 0.3822
Epoch 3/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8743 - loss: 0.3437 - val_accuracy: 0.8602 - val_loss: 0.3709
Epoch 4/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8820 - loss: 0.3205 - val_accuracy: 0.8787 - val_loss: 0.3530
Epoch 5/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8913 - loss: 0.2958 - val_accuracy: 0.8837 - val_loss: 0.3224
Epoch 6/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8949 - loss: 0.2855 - val_accuracy: 0.8795 - val_loss: 0.3377
Epoch 7/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9015 - loss: 0.2670 - val_accuracy: 0.8865 - val_loss: 0.3312
Epoch 8/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9040 - loss: 0.2565 - val_accuracy: 0.8837 - val_loss: 0.3261
Epoch 9/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9074 - loss: 0.2493 - val_accuracy: 0.8867 - val_loss: 0.3151
Epoch 10/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9116 - loss: 0.2364 - val_accuracy: 0.8843 - val_loss: 0.3247
```

Figure 3: training result of FCNN

CNN model

```
model.compile(optimizer='adam',loss='categorical_crossentropy',
    metrics=['accuracy'])
history = model.fit(trainX,trainY,batch_size=64,epochs=10,
    validation_split=0.1)
```

Output after training the model is:

```
Epoch 1/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 11s 12ms/step - accuracy: 0.5370 - loss: 1.2186 - val_accuracy: 0.8340 - val_loss: 0.4556
Epoch 2/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.8002 - loss: 0.5512 - val_accuracy: 0.8527 - val_loss: 0.3832
Epoch 3/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.8301 - loss: 0.4720 - val_accuracy: 0.8730 - val_loss: 0.3377
Epoch 4/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.8486 - loss: 0.4238 - val_accuracy: 0.8852 - val_loss: 0.3087
Epoch 5/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.8565 - loss: 0.3975 - val_accuracy: 0.8853 - val_loss: 0.2999
Epoch 6/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.8671 - loss: 0.3667 - val_accuracy: 0.8970 - val_loss: 0.2768
Epoch 7/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.8724 - loss: 0.3547 - val_accuracy: 0.9002 - val_loss: 0.2694
Epoch 8/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.8752 - loss: 0.3474 - val_accuracy: 0.9037 - val_loss: 0.2590
Epoch 9/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.8806 - loss: 0.3315 - val_accuracy: 0.9075 - val_loss: 0.2545
Epoch 10/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.8839 - loss: 0.3226 - val_accuracy: 0.9058 - val_loss: 0.2540
```

Figure 4: training result of CNN

After Testing the models

FCNN model

```
# Evaluate model performance
result = model.evaluate(testX, testY)
print("Test Loss:", result[0])
print("Test Accuracy:", result[1])

# Predict Y values
predictY = model.predict(testX)

print('OriginalY   PredictedY')
print('=========   ==========')
for i in range(10):
  print(np.argmax(testY[i]), '\t\t', np.argmax(predictY[i]))
```

Output after training the model is:

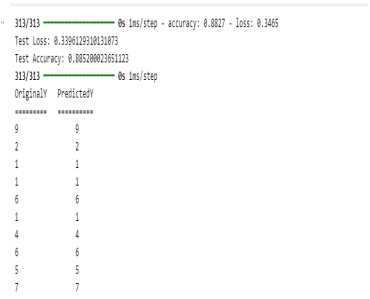

Figure 5: testing result of FCNN

CNN model

```
# Evaluate model performance
result = model.evaluate(testX, testY)
print("Test Loss:", result[0])
print("Test Accuracy:", result[1])

# Predict Y values
predictY = model.predict(testX)

print('OriginalY    PredictedY')
print('=========    ==========')
for i in range(10):
  print(np.argmax(testY[i]), '\t\t', np.argmax(predictY[i]))
```

Output after training the model is:

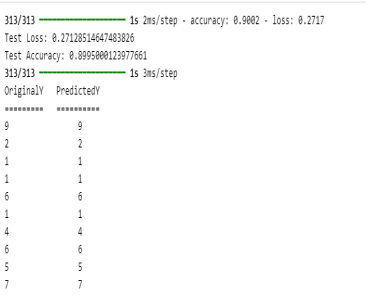

Figure 6: testing result of CNN

Here is the comparision between the FCNN and CNN model are given in tabular form:

Parameter	FCNN Model	CNN Model
Optimizer	adam	adam
Loss function	categorical_crossentropy	categorical_crossentropy
Metrics	['accuracy']	['accuracy']
Batch_Size	64	64
Epochs	10	10
Validation_Split	0.1	0.1
Test Accuracy	88.52%	89.95%
Test Loss	33.96%	27.12%
Validation_Accuracy	88.67%(max) (this result obtain the 10/10 epoch)	90.58%(max) (this result obtain the 10/10 epochs)
Validation_Loss	45.52%(max) (this result obtain the 1/10 epoch)	45.56%(max)(this result obtain the 1/10 epochs)

Table 1: FCNN and CNN comparision with the Fashion Dataset

The FCNN and CNN model with Fashion dataset the model accuracy and model loss graph are given in below:

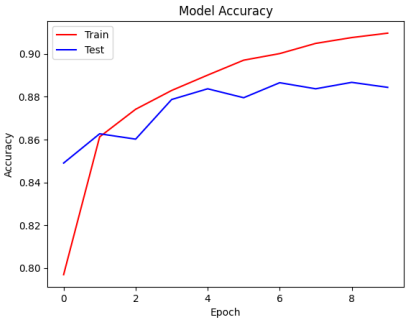

Figure 7: Accuracy

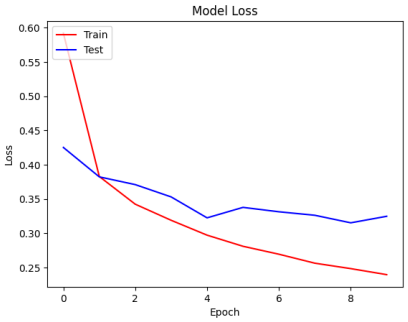

Figure 8: Loss

Figure 9: FCNN Fashion dataset model accuracy and loss

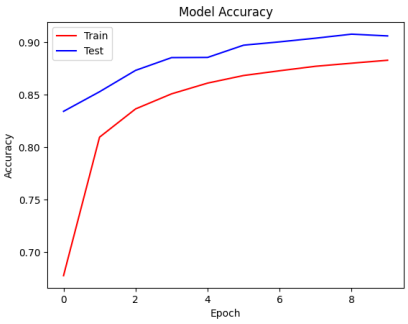

Figure 10: Accuracy

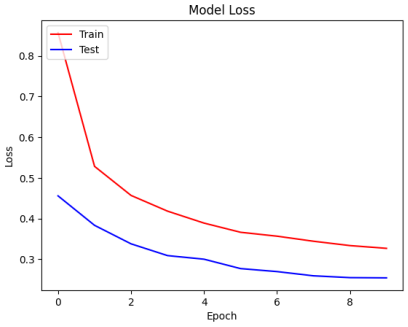

Figure 11: Loss

Figure 12: CNN Fashion dataset model accuracy and loss

Answer To The Question No: 3

Building a CNN having a pre-trained MobileNet model as backbone to classify 10 classes is given below:

```
#load pre-train MobileNet model as backbone
mobilenet_model = mobilenet.MobileNet(weights='imagenet',
    include_top=False,input_shape=(244,244,3))
mobilenet_model.summary()

mobilenet.trainable = False
#build a new model based on pre-trained MobileNet
inputs = mobilenet_model.input
x = mobilenet_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
outputs = Dense(10, activation='softmax')(x)
#combine MobileNet model and custom head into a new model
model = Model(inputs,outputs)
model.summary()
```

The parameter list:

```
Total params: 3,361,354 (12.82 MB)
Trainable params: 132,490 (517.54 KB)
Non-trainable params: 3,228,864 (12.32 MB)
```

Answer To The Question No: 4

After Traning and Testing the CNN having a pre-trained MobileNet with the CIFAR-10 dataset then the performance of the transfer learning only and transfer learning + fine tuning are given below:

After training the CNN model

Transfer Learning only

```
from tensorflow.keras.optimizers import Adam
model.compile(optimizer=Adam(learning_rate=0.001), loss='
    categorical_crossentropy', metrics=['accuracy'])
history = model.fit(trainX, trainY, batch_size=64, epochs=15,
    validation_split=0.2)
```

Transfer learning with fine tuning

```
# Unfreeze the last 50 layers
for layer in mobilenet_model.layers[-50:]:
    layer.trainable = True
model.compile(optimizer='adam', loss='categorical_crossentropy',
    metrics=['accuracy'])
history_fine_tuning = model.fit(trainX, trainY, batch_size=64,
    epochs=15, validation_split=0.2)
```

After testing the CNN model

Transfer Learning only

```
import numpy as np
# Evaluate model performance
result = model.evaluate(testX, testY)
print("Test Loss:", result[0])
print("Test Accuracy:", result[1])

# Predict Y values
predictY = model.predict(testX)

print('OriginalY   PredictedY')
print('=========   ==========')
for i in range(10):
  print(np.argmax(testY[i]), '\t\t', np.argmax(predictY[i]))
#The np.argmax() function in NumPy is used to find the index of the
     maximum value
```

Output of the transfer learning only

```
313/313 ---------------4s 12ms/step - accuracy: 0.2029 - loss:
    2.1485
Test Loss: 2.139378786087036
Test Accuracy: 0.20819999277591705
313/313 ----------------5s 13ms/step
OriginalY   PredictedY
=========   ==========
3           8
8           9
8           1
0           8
6           6
6           6
1           4
6           6
3           9
1           7
```

Transfer learning with fine tuning

```
import numpy as np
# Evaluate model performance
result = model.evaluate(testX, testY)
print("Test Loss:", result[0])
print("Test Accuracy:", result[1])

# Predict Y values
predictY = model.predict(testX)

print('OriginalY   PredictedY')
print('=========   ==========')
for i in range(10):
  print(np.argmax(testY[i]), '\t\t', np.argmax(predictY[i]))
#The np.argmax() function in NumPy is used to find the index of the
     maximum value
```

Output of the transfer leraning with fine tuning

```
313/313 ------------- 4s 13ms/step - accuracy: 0.5546 - loss:
    1.4877
Test Loss: 1.4942177534103394
Test Accuracy: 0.5568000078201294
313/313 ------------ 5s 14ms/step
OriginalY   PredictedY
=========   ==========
3           3
8           1
8           8
0           8
6           6
6           6
1           3
6           6
3           4
1           1
```

Here is teh comparision between the transfer learning and tranfer learning with fine tuining of CNN model are given in tabular form:

Parameter	Transfer learning	Transfer learning with Fine tuning
Optimizer	adam	adam
Loss function	categorical_crossentropy	categorical_crossentropy
Metrics	['accuracy']	['accuracy']
Batch_Size	64	64
Epochs	15	15
Validation_Split	0.2	0.2
Test Accuracy	0.2081	0.5568
Test Loss	2.1393	1.4942
Validation_Accuracy	0.2111(max) (this result obtain the 15/15 epoch)	0.5740%(max) (this result obtain the 12/15 epochs)
Validation_Loss	2.1572(max) (this result obtain the 1/15 epoch)	1.6444(max)(this result obtain the 1/15 epochs)

Table 2: Transfer-learning and Transfer-learning with fine tuning

The accuracy comparision graph is:

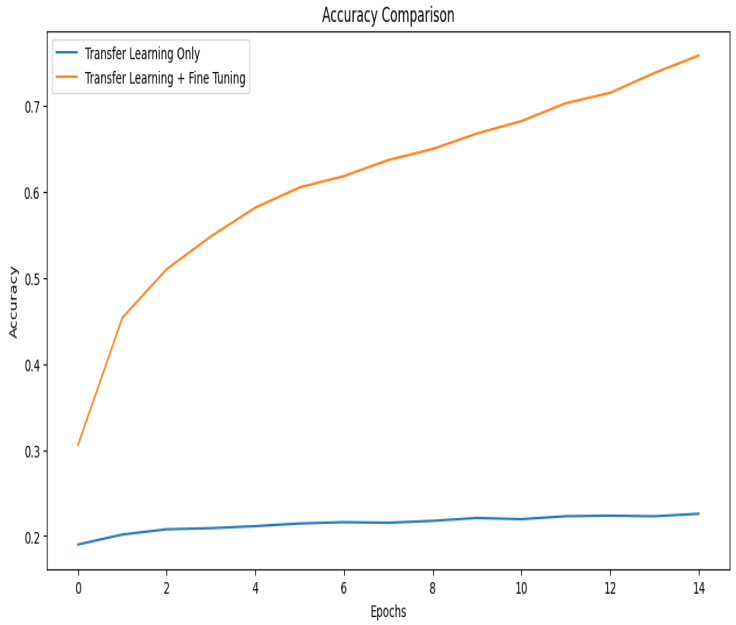

Figure 13: Accuracy of transfer-learning and transfer-learning+fine tuning

The loss comparision graph is: The validation accuracy comparision graph

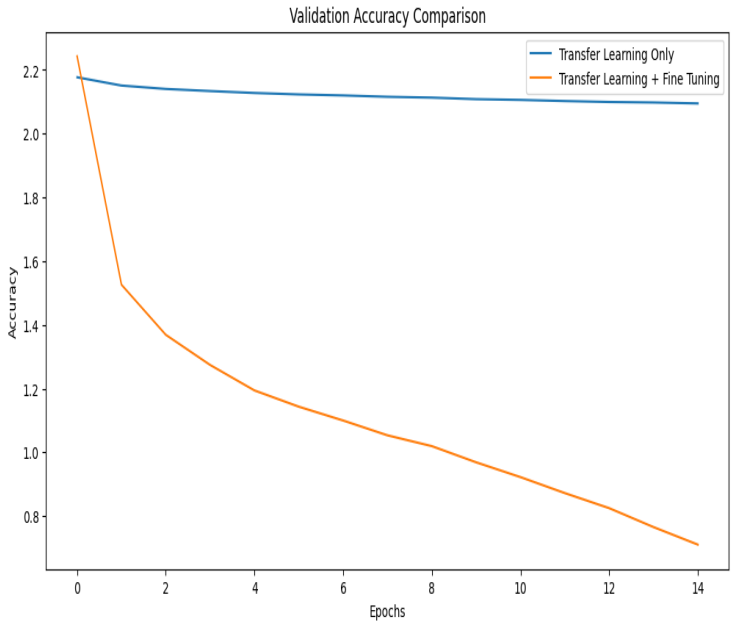

Figure 14: Loss of transfer-learning and transfer-learning+fine tuning

is:

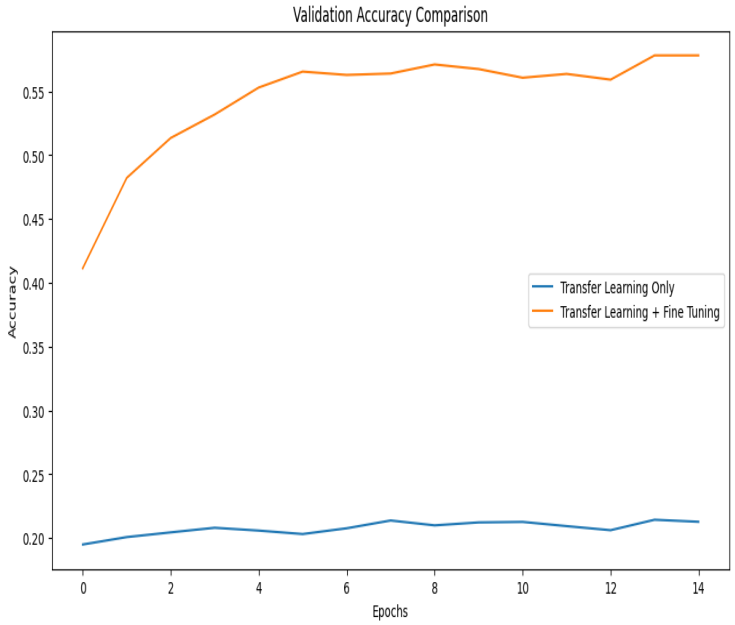

Figure 15: Validation-accuracy of transfer-learning and transfer-learning+fine tuning