Artificial Intelligence Lab

Md. Rayhanul Islam
Roll: 2010976154

December 21, 2024

0.1 Q1: Build a fully connected neural network (FCNN) and a convolutional neural network (CNN) for classifying 10 classes of images.

Ans:

Import necessary modules

```
from tensorflow.keras.layers import Input, Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.models import Model
```

Build FCNN

```
inputs = Input(shape=(28, 28, 1))
x = Flatten()(inputs)
x = Dense(8, activation = 'relu')(x)
x = Dense(16, activation = 'relu')(x)
x = Dense(32, activation = 'relu')(x)
x = Dense(64, activation = 'relu')(x)
x = Dense(128, activation = 'relu')(x)
x = Dense(64, activation = 'relu')(x)
outputs = Dense(10, name = 'outputLayers', activation = 'softmax')(x)

model = Model(inputs, outputs, name = 'FCNN')
model.summary()
```

Model: "FCNN"

```
Layer (type)                        Output Shape              Param #
==========================================================================
input_layer_8 (InputLayer)          (None, 28, 28, 1)         0
flatten_8 (Flatten)                 (None, 784)               0
dense_26 (Dense)                    (None, 8)                 6,280
dense_27 (Dense)                    (None, 16)                144
dense_28 (Dense)                    (None, 32)                544
dense_29 (Dense)                    (None, 64)                2,112
dense_30 (Dense)                    (None, 128)               8,320
dense_31 (Dense)                    (None, 64)                8,256
outputLayers (Dense)                (None, 10)                650
==========================================================================
Total params: 26,306 (102.76 KB)
Trainable params: 26,306 (102.76 KB)
Non-trainable params: 0 (0.00 B)
```

Build CNN

```
inputs = Input(shape=(28, 28, 1))
x = Conv2D(filters = 8, kernel_size = (3, 3), activation = 'relu')(inputs)
x = Conv2D(filters = 16, kernel_size = (3, 3), activation = 'relu')(x)
x = Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu')(x)
x = Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu')(x)
x = MaxPooling2D()(x)
x = Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu')(x)
x = Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu')(x)
x = Flatten()(inputs)
x = Dense(16, activation = 'relu')(x)
x = Dense(32, activation = 'relu')(x)
x = Dense(64, activation = 'relu')(x)
```

```
outputs = Dense(10, name = 'outputLayers', activation = 'softmax')(x)

model = Model(inputs, outputs, name = 'CNN')
model.summary()
```

```
Model: "CNN"
────────────────────────────────────────────────────────────────────────────────
 Layer (type)                           Output Shape                Param #
================================================================================
 input_layer_10 (InputLayer)            (None, 28, 28, 1)           0
 flatten_10 (Flatten)                   (None, 784)                 0
 dense_35 (Dense)                       (None, 16)                  12,560
 dense_36 (Dense)                       (None, 32)                  544
 dense_37 (Dense)                       (None, 64)                  2,112
 outputLayers (Dense)                   (None, 10)                  650
================================================================================
 Total params: 15,866 (61.98 KB)
 Trainable params: 15,866 (61.98 KB)
 Non-trainable params: 0 (0.00 B)
```

0.2 Q2: Train and test your FCNN and CNN by the Fashion dataset. Discuss your results by comparing performance between two types of networks.

Ans:

Import necessary modules

```
from tensorflow.keras.layers import Input, Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets.mnist import load_data
import numpy as np
```

Load Dataset

```
(trainX, trainY), (testX, testY) = load_data()
trainX = np.expand_dims(trainX, axis=-1)
testX = np.expand_dims(testX, axis=-1)

trainY = to_categorical(trainY)
testY = to_categorical(testY)

print(trainX.shape, trainY.shape, testX.shape, testY.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434    |==============>    0s 0us/step
(60000, 28, 28, 1) (60000, 10) (10000, 28, 28, 1) (10000, 10)
```

Build FCNN

```
inputs = Input(shape=(28, 28, 1))
x = Flatten()(inputs)
x = Dense(8, activation = 'relu')(x)
x = Dense(16, activation = 'relu')(x)
x = Dense(32, activation = 'relu')(x)
x = Dense(64, activation = 'relu')(x)
x = Dense(128, activation = 'relu')(x)
x = Dense(64, activation = 'relu')(x)
outputs = Dense(10, name = 'outputLayers', activation = 'softmax')(x)

model = Model(inputs, outputs, name = 'FCNN')
model.summary()
```

Model: "FCNN"

```
Layer (type)                          Output Shape              Param #
==========================================================================
input_layer_8 (InputLayer)            (None, 28, 28, 1)         0
flatten_8 (Flatten)                   (None, 784)               0
dense_26 (Dense)                      (None, 8)                 6,280
dense_27 (Dense)                      (None, 16)                144
dense_28 (Dense)                      (None, 32)                544
dense_29 (Dense)                      (None, 64)                2,112
dense_30 (Dense)                      (None, 128)               8,320
dense_31 (Dense)                      (None, 64)                8,256
outputLayers (Dense)                  (None, 10)                650
==========================================================================
Total params: 26,306 (102.76 KB)
Trainable params: 26,306 (102.76 KB)
Non-trainable params: 0 (0.00 B)
```

Train classifier without validation dataset.

```
model.compile(loss = 'categorical_crossentropy', metrics = ['accuracy'])
model.fit(trainX, trainY, epochs = 10, batch_size = 128)

print(f"Loss: {model.evaluate(testX, testY)[0]}")
print(f"Accuracy: {model.evaluate(testX, testY)[1]}")
```

```
  Epoch 1/10
469/469   |==============>    2s 2ms/step    - accuracy: 0.5808 - loss: 1.3776
Epoch 2/10
469/469   |==============>    1s 2ms/step    - accuracy: 0.8841 - loss: 0.4009
Epoch 3/10
469/469   |==============>    1s 3ms/step    - accuracy: 0.9083 - loss: 0.3146
Epoch 4/10
469/469   |==============>    2s 4ms/step    - accuracy: 0.9174 - loss: 0.2840
Epoch 5/10
469/469   |==============>    2s 3ms/step    - accuracy: 0.9249 - loss: 0.2600
Epoch 6/10
469/469   |==============>    1s 2ms/step    - accuracy: 0.9283 - loss: 0.2522
Epoch 7/10
469/469   |==============>    1s 2ms/step    - accuracy: 0.9302 - loss: 0.2413
Epoch 8/10
469/469   |==============>    1s 2ms/step    - accuracy: 0.9323 - loss: 0.2386
Epoch 9/10
469/469   |==============>    1s 2ms/step    - accuracy: 0.9354 - loss: 0.2266
Epoch 10/10
469/469   |==============>    1s 2ms/step    - accuracy: 0.9366 - loss: 0.2234
313/313   |==============>    1s 1ms/step    - accuracy: 0.9260 - loss: 0.2784
Loss: 0.25537511706352234
313/313   |==============>    0s 1ms/step    - accuracy: 0.9260 - loss: 0.2784
Accuracy: 0.9334999918937683
```

Test the performance of the model

```
model.evaluate(testX, testY)
predictY = model.predict(testX)

print('OriginalY  PredictedY')
for i in range(10):
    print(f'{np.argmax(testY[i])}          {np.argmax(predictY[i])}')
```

```
313/313    |==============>    1s 2ms/step    - accuracy: 0.9260 - loss: 0.2784
313/313    |==============>    1s 1ms/step
```

OriginalY	PredictedY
7	7
2	2
1	1
0	0
4	4
1	1
4	4
9	9
5	5
9	9

Training FCNN with Validation

```
model.compile(loss = 'categorical_crossentropy', metrics = ['accuracy'])
model.fit(trainX, trainY, epochs = 10, batch_size = 128, validation_split = 0.2)

print(f"Loss: {model.evaluate(testX, testY)[0]}")
print(f"Accuracy: {model.evaluate(testX, testY)[1]}")
```

```
Epoch 1/10
375/375    |==============>    2s 3ms/step    - accuracy: 0.9334 - loss: 0.2383 - val_accuracy: 0.9436 -
val_loss: 0.1943
Epoch 2/10
375/375    |==============>    2s 3ms/step    - accuracy: 0.9347 - loss: 0.2310 - val_accuracy: 0.9359 -
val_loss: 0.2363
Epoch 3/10
375/375    |==============>    1s 3ms/step    - accuracy: 0.9388 - loss: 0.2202 - val_accuracy: 0.9275 -
val_loss: 0.2703
Epoch 4/10
375/375    |==============>    1s 3ms/step    - accuracy: 0.9375 - loss: 0.2209 - val_accuracy: 0.9297 -
val_loss: 0.2651
Epoch 5/10
375/375    |==============>    1s 3ms/step    - accuracy: 0.9394 - loss: 0.2214 - val_accuracy: 0.9370 -
val_loss: 0.2256
Epoch 6/10
375/375    |==============>    2s 4ms/step    - accuracy: 0.9370 - loss: 0.2209 - val_accuracy: 0.9353 -
val_loss: 0.2282
Epoch 7/10
375/375    |==============>    2s 5ms/step    - accuracy: 0.9390 - loss: 0.2177 - val_accuracy: 0.9400 -
val_loss: 0.2214
Epoch 8/10
375/375    |==============>    2s 3ms/step    - accuracy: 0.9382 - loss: 0.2212 - val_accuracy: 0.9394 -
val_loss: 0.2222
Epoch 9/10
375/375    |==============>    1s 3ms/step    - accuracy: 0.9394 - loss: 0.2187 - val_accuracy: 0.9394 -
val_loss: 0.2355
Epoch 10/10
375/375    |==============>    1s 3ms/step    - accuracy: 0.9389 - loss: 0.2191 - val_accuracy: 0.9392 -
val_loss: 0.2345
313/313    |==============>    0s 1ms/step    - accuracy: 0.9312 - loss: 0.2854
Loss: 0.2612617611885071
313/313    |==============>    0s 1ms/step    - accuracy: 0.9312 - loss: 0.2854
Accuracy: 0.9379000067710876
```

Build CNN

```
inputs = Input(shape=(28, 28, 1))
x = Conv2D(filters = 8, kernel_size = (3, 3), activation = 'relu')(inputs)
x = Conv2D(filters = 16, kernel_size = (3, 3), activation = 'relu')(x)
x = Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu')(x)
x = Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu')(x)
x = MaxPooling2D()(x)
x = Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu')(x)
x = Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu')(x)
x = Flatten()(inputs)
x = Dense(16, activation = 'relu')(x)
x = Dense(32, activation = 'relu')(x)
x = Dense(64, activation = 'relu')(x)
outputs = Dense(10, name = 'outputLayers', activation = 'softmax')(x)

model = Model(inputs, outputs, name = 'CNN')
model.summary()
```

```
Model: "CNN"
 Layer (type)                      Output Shape              Param #
 ==================================================================
 input_layer_10 (InputLayer)       (None, 28, 28, 1)         0
 flatten_10 (Flatten)              (None, 784)               0
 dense_35 (Dense)                  (None, 16)                12,560
 dense_36 (Dense)                  (None, 32)                544
 dense_37 (Dense)                  (None, 64)                2,112
 outputLayers (Dense)              (None, 10)                650
 ==================================================================
 Total params: 15,866 (61.98 KB)
 Trainable params: 15,866 (61.98 KB)
 Non-trainable params: 0 (0.00 B)
```

Train classifier without validation dataset.

```
model.compile(loss = 'categorical_crossentropy', metrics = ['accuracy'])
model.fit(trainX, trainY, epochs = 10, batch_size = 128)

print(f"Loss: {model.evaluate(testX, testY)[0]}")
print(f"Accuracy: {model.evaluate(testX, testY)[1]}")
```

```
   Epoch 1/10
469/469   |==============>   2s 2ms/step   - accuracy: 0.4938 - loss: 3.3086
Epoch 2/10
469/469   |==============>   2s 3ms/step   - accuracy: 0.8395 - loss: 0.5652
Epoch 3/10
469/469   |==============>   2s 2ms/step   - accuracy: 0.8861 - loss: 0.4308
Epoch 4/10
469/469   |==============>   1s 2ms/step   - accuracy: 0.9030 - loss: 0.3722
Epoch 5/10
469/469   |==============>   1s 2ms/step   - accuracy: 0.9082 - loss: 0.3440
Epoch 6/10
469/469   |==============>   1s 2ms/step   - accuracy: 0.9185 - loss: 0.3092
Epoch 7/10
469/469   |==============>   1s 2ms/step   - accuracy: 0.9209 - loss: 0.3011
Epoch 8/10
469/469   |==============>   1s 2ms/step   - accuracy: 0.9234 - loss: 0.2902
Epoch 9/10
469/469   |==============>   1s 2ms/step   - accuracy: 0.9271 - loss: 0.2760
Epoch 10/10
469/469   |==============>   1s 2ms/step   - accuracy: 0.9279 - loss: 0.2734
313/313   |==============>   0s 1ms/step   - accuracy: 0.9220 - loss: 0.3012
```

```
Loss: 0.27396932244430084
313/313      |==============>      1s 2ms/step      - accuracy: 0.9220 - loss: 0.3012
Accuracy: 0.9297999739646912
```

Test the performance of the model

```python
model.evaluate(testX, testY)
predictY = model.predict(testX)

print('OriginalY  PredictedY')
for i in range(10):
    print(f'{np.argmax(testY[i])}          {np.argmax(predictY[i])}')
```

```
313/313      |==============>      1s 2ms/step      - accuracy: 0.9260 - loss: 0.2784
313/313      |==============>      1s 1ms/step
```

OriginalY	PredictedY
7	7
2	2
1	1
0	0
4	4
1	1
4	4
9	9
5	6
9	9

Training FCNN with Validation

```python
model.compile(loss = 'categorical_crossentropy', metrics = ['accuracy'])
model.fit(trainX, trainY, epochs = 10, batch_size = 128, validation_split = 0.2)

print(f"Loss: {model.evaluate(testX, testY)[0]}")
print(f"Accuracy: {model.evaluate(testX, testY)[1]}")
```

```
Epoch 1/10
375/375      |==============>      2s 3ms/step      - accuracy: 0.9270 - loss: 0.2741 - val_accuracy: 0.9363 -
val_loss: 0.2422
Epoch 2/10
375/375      |==============>      1s 2ms/step      - accuracy: 0.9308 - loss: 0.2513 - val_accuracy: 0.9366 -
val_loss: 0.2345
Epoch 3/10
375/375      |==============>      1s 2ms/step      - accuracy: 0.9291 - loss: 0.2547 - val_accuracy: 0.9333 -
val_loss: 0.2450
Epoch 4/10
375/375      |==============>      1s 2ms/step      - accuracy: 0.9325 - loss: 0.2434 - val_accuracy: 0.9281 -
val_loss: 0.2662
Epoch 5/10
375/375      |==============>      1s 2ms/step      - accuracy: 0.9329 - loss: 0.2379 - val_accuracy: 0.9331 -
val_loss: 0.2541
Epoch 6/10
375/375      |==============>      2s 4ms/step      - accuracy: 0.9368 - loss: 0.2316 - val_accuracy: 0.9324 -
val_loss: 0.2533
Epoch 7/10
375/375      |==============>      2s 4ms/step      - accuracy: 0.9363 - loss: 0.2271 - val_accuracy: 0.9342 -
val_loss: 0.2489
Epoch 8/10
375/375      |==============>      2s 2ms/step      - accuracy: 0.9391 - loss: 0.2168 - val_accuracy: 0.9324 -
val_loss: 0.2598
Epoch 9/10
```

```
375/375      |==============>      1s 2ms/step      - accuracy: 0.9399 - loss: 0.2154 - val_accuracy: 0.9333 -
val_loss: 0.2548
Epoch 10/10
375/375      |==============>      1s 2ms/step      - accuracy: 0.9380 - loss: 0.2185 - val_accuracy: 0.9360 -
val_loss: 0.2474
313/313      |==============>      0s 1ms/step      - accuracy: 0.9244 - loss: 0.3027
Loss: 0.27270984649658203
313/313      |==============>      0s 1ms/step      - accuracy: 0.9244 - loss: 0.3027
Accuracy: 0.9326000213623047
```

0.3 Q3:Build a CNN having a pre-trained MobileNet as backbone to classify 10 classes.

Ans:

Import necessary modules

```
from tensorflow.keras.layers import Input, Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets.cifar10 import load_data
from tensorflow.keras.applications import mobilenet
import numpy as np
import tensorflow as tf
```

Load Dataset

```
(trainX, trainY), (testX, testY) = load_data()

trainX = tf.image.resize(trainX, (224, 224))
testX = tf.image.resize(testX, (224, 224))

trainY = to_categorical(trainY)
testY = to_categorical(testY)

print(f"Train: X={trainX.shape}, Y={trainY.shape}")
print(f"Test: X={testX.shape}, Y={testY.shape}")
```

Build CNN

```
base_model = mobilenet.MobileNet(weights=’imagenet’, include_top=False, input_shape=(224,
     224, 3))
base_model.trainable = False
x = base_model.output
x = Flatten()(x)
x = Dense(16, activation = ’relu’)(x)
x = Dense(32, activation = ’relu’)(x)
x = Dense(64, activation = ’relu’)(x)
outputs = Dense(10, activation = ’softmax’)(x)
model = Model(inputs = base_model.input, outputs = outputs)
model.summary()
```

Train classifier without validation dataset.

```
model.compile(loss = ’categorical_crossentropy’, metrics = [’accuracy’])
model.fit(trainX, trainY, epochs = 10, batch_size = 256)

print(f"Loss: {model.evaluate(testX, testY)[0]}")
print(f"Accuracy: {model.evaluate(testX, testY)[1]}")
```

Test the performance of the model

```
model.evaluate(testX, testY)
predictY = model.predict(testX)
```

```
print('OriginalY  PredictedY')
for i in range(10):
    print(f'{np.argmax(testY[i])}          {np.argmax(predictY[i])}')
```

Training CNN with Validation

```
model.compile(loss = 'categorical_crossentropy', metrics = ['accuracy'])
model.fit(trainX, trainY, epochs = 10, batch_size = 128, validation_split = 0.2)


print(f"Loss: {model.evaluate(testX, testY)[0]}")
print(f"Accuracy: {model.evaluate(testX, testY)[1]}")
```