CSE4132: Artificial Intelligence Lab Assignment_18-12-2024

Nusrat Jannat

Roll: 2012176145

21 December,2024

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

Ans:1

FCNN for classifying 10 classes of images:

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Flatten
from tensorflow.keras.utils import to_categorical
num_classes = 10
inputs = Input((28,28, 1))
x = Flatten()(inputs)
x = Dense(512, activation='relu')(x)
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
outputs = Dense(num_classes, activation='softmax', name="OutputLayer")(x)
model = Model(inputs, outputs, name="FCNN_Classifier")
model.summary()
```

Output:

Model: "FCNN_Classifier"

Layer (type)	Output Shape	Param #
input_layer (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
OutputLayer (Dense)	(None, 10)	1,290

Total params: 567,434 (2.16 MB)
Trainable params: 567,434 (2.16 MB)
Non-trainable params: 0 (0.00 B)

CNN for classifying 10 classes of images:

```
import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Conv2D, Flatten, Dense

num_classes = 10

inputs = Input((28,28, 1))

x = Conv2D(32, kernel_size=(3, 3), padding = 'same', activation='relu')(inputs)  # 32 filters

x = Conv2D(64, kernel_size=(3, 3), padding = 'same', activation='relu')(x)  # 64 filters

x = Conv2D(128, kernel_size=(3, 3), padding = 'same', activation='relu')(x)  # 128 filters

x = Flatten()(x)

outputs = Dense(num_classes, activation='softmax', name="OutputLayer")(x)

model = Model(inputs, outputs, name="CNN_Classifier")

model.summary()
```

Output:

Model: "CNN_Classifier"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 28, 28, 1)	0
conv2d_3 (Conv2D)	(None, 28, 28, 32)	320
conv2d_4 (Conv2D)	(None, 28, 28, 64)	18,496
conv2d_5 (Conv2D)	(None, 28, 28, 128)	73,856
flatten_2 (Flatten)	(None, 100352)	0
OutputLayer (Dense)	(None, 10)	1,003,530

Total params: 1,096,202 (4.18 MB)
Trainable params: 1,096,202 (4.18 MB)
Non-trainable params: 0 (0.00 B)

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

Ans:2

Train and test FCNN:

```
from tensorflow.keras.datasets import fashion_mnist
```

```
import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.layers import Input, Flatten, Dense

from tensorflow.keras.models import Model


# Helper function to display images

def display_img(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()


# Load the Fashion-MNIST dataset

(trainX, trainY), (testX, testY) = fashion_mnist.load_data()


# Investigate loaded 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()))


# Class labels for Fashion-MNIST

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

            'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

```
# Display some loaded image data with labels
titles = [class_names[label] for label in trainY[:9]]
display_img(trainX[:9], titles)

# Expand dimensions for CNN input (28x28 grayscale to 28x28x1)
trainX = np.expand_dims(trainX, axis=-1)
testX = np.expand_dims(testX, axis=-1)

# Normalize the image data
trainX = trainX / 255.0
testX = testX / 255.0

# Investigate updated X
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(),
testX.min()))

# Turn Y into one-hot encoding correctly (num_classes=10)
trainY = to_categorical(trainY, num_classes=10)
testY = to_categorical(testY, num_classes=10)

# Investigate updated Y
print('trainY.shape: {}, testY.shape: {}'.format(trainY.shape, testY.shape))
print('trainY.dtype: {}, testY.dtype: {}'.format(trainY.dtype, testY.dtype))
print(trainY[:5])

# Build the fully connected neural network model
```

```
inputs = Input((28, 28, 1), name='InputLayer')
x = Flatten()(inputs)
x = Dense(512, activation='relu')(x)
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
outputs = Dense(10, activation='softmax', name='OutputLayer')(x)
model = Model(inputs, outputs, name='Fashion-Multi-Class-Classifier')
model.summary()

# Compile the model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(trainX, trainY, batch_size=32, validation_split=0.1, epochs=10)

# Evaluate model performance
model.evaluate(testX, testY)

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

# Print original and predicted Y values
print('OriginalY   PredictedY')
print('=========   ===========')
for i in range(10):
    print(np.argmax(testY[i]), '\t\t', np.argmax(predictY[i]))
```

Output:

```
Epoch 1/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 17s 9ms/step - accuracy: 0.7620 - loss: 0.6571 - val_accuracy: 0.8458 - val_loss: 0.4220
Epoch 2/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 19s 9ms/step - accuracy: 0.8517 - loss: 0.4140 - val_accuracy: 0.8573 - val_loss: 0.4155
Epoch 3/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 20s 9ms/step - accuracy: 0.8654 - loss: 0.3833 - val_accuracy: 0.8600 - val_loss: 0.4198
Epoch 4/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 21s 9ms/step - accuracy: 0.8705 - loss: 0.3727 - val_accuracy: 0.8072 - val_loss: 0.6268
Epoch 5/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 21s 9ms/step - accuracy: 0.8769 - loss: 0.3638 - val_accuracy: 0.8728 - val_loss: 0.4060
Epoch 6/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 19s 8ms/step - accuracy: 0.8788 - loss: 0.3606 - val_accuracy: 0.8680 - val_loss: 0.4469
Epoch 7/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.8818 - loss: 0.3499 - val_accuracy: 0.8680 - val_loss: 0.4378
Epoch 8/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 22s 9ms/step - accuracy: 0.8826 - loss: 0.3534 - val_accuracy: 0.8512 - val_loss: 0.5155
Epoch 9/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 19s 8ms/step - accuracy: 0.8826 - loss: 0.3526 - val_accuracy: 0.8705 - val_loss: 0.5172
Epoch 10/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.8848 - loss: 0.3491 - val_accuracy: 0.8435 - val_loss: 0.5342
```

Train and test CNN:

```
from tensorflow.keras.datasets import fashion_mnist

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.layers import Input, Flatten, Conv2D

from tensorflow.keras.models import Model


# Helper function to display images

def display_img(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()


# Load the Fashion-MNIST dataset
```

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


# Investigate loaded 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()))


# Class labels for Fashion-MNIST
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']


# Display some loaded image data with labels
titles = [class_names[label] for label in trainY[:9]]
display_img(trainX[:9], titles)


# Expand dimensions for CNN input (28x28 grayscale to 28x28x1)
trainX = np.expand_dims(trainX, axis=-1)
testX = np.expand_dims(testX, axis=-1)


# Normalize the image data
trainX = trainX / 255.0
testX = testX / 255.0


# Investigate updated X
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(), testX.min()))


# Turn Y into one-hot encoding correctly (num_classes=10)
trainY = to_categorical(trainY, num_classes=10)
testY = to_categorical(testY, num_classes=10)


# Investigate updated Y
print('trainY.shape: {}, testY.shape: {}'.format(trainY.shape, testY.shape))
print('trainY.dtype: {}, testY.dtype: {}'.format(trainY.dtype, testY.dtype))
print(trainY[:5])


# Build the convolutional neural network model
num_classes = 10


inputs = Input((28, 28, 1))
x = Conv2D(32, kernel_size=(3, 3), padding = 'same', activation='relu')(inputs)  # 32 filters
x = Conv2D(64, kernel_size=(3, 3), padding = 'same', activation='relu')(x)  # 64 filters
x = Conv2D(128, kernel_size=(3, 3), padding = 'same', activation='relu')(x)  # 128 filters
x = Flatten()(x)


outputs = Dense(num_classes, activation='softmax', name="OutputLayer")(x)
model = Model(inputs, outputs, name="CNN_Classifier")
model.summary()


# Compile the model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'])


# Train the model
```

```
model.fit(trainX, trainY, batch_size=32, validation_split=0.1, epochs=10)


# Evaluate model performance

model.evaluate(testX, testY)


# Predict Y values

predictY = model.predict(testX)


# Print original and predicted Y values

print('OriginalY   PredictedY')

print('=========   ===========')

for i in range(10):

    print(np.argmax(testY[i]), '\t\t', np.argmax(predictY[i]))
```

Output:

```
Epoch 1/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 609s 360ms/step - accuracy: 0.9144 - loss: 2.6795 - val_accuracy: 0.9773 - val_loss: 0.0872
Epoch 2/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 608s 360ms/step - accuracy: 0.9849 - loss: 0.0510 - val_accuracy: 0.9882 - val_loss: 0.0498
Epoch 3/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 609s 361ms/step - accuracy: 0.9918 - loss: 0.0315 - val_accuracy: 0.9852 - val_loss: 0.0739
Epoch 4/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 619s 359ms/step - accuracy: 0.9937 - loss: 0.0247 - val_accuracy: 0.9862 - val_loss: 0.0920
Epoch 5/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 628s 363ms/step - accuracy: 0.9955 - loss: 0.0195 - val_accuracy: 0.9850 - val_loss: 0.1413
Epoch 6/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 610s 362ms/step - accuracy: 0.9972 - loss: 0.0148 - val_accuracy: 0.9862 - val_loss: 0.1722
Epoch 7/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 620s 361ms/step - accuracy: 0.9981 - loss: 0.0137 - val_accuracy: 0.9865 - val_loss: 0.2237
Epoch 8/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 621s 360ms/step - accuracy: 0.9982 - loss: 0.0117 - val_accuracy: 0.9855 - val_loss: 0.2581
Epoch 9/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 622s 360ms/step - accuracy: 0.9978 - loss: 0.0167 - val_accuracy: 0.9860 - val_loss: 0.3638
Epoch 10/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 623s 361ms/step - accuracy: 0.9981 - loss: 0.0172 - val_accuracy: 0.9897 - val_loss: 0.3447
<keras.src.callbacks.history.History at 0x7a6f65e15ba0>
```

Explanation:

From FCNN and CNN we can observe that the performance of CNN is better than FCNN as the accuracy for CNN is 99%(training level) and 98%(validation level) whereas for FCNN the accuracy is 88%%(training level) and 84%(validation level).

Ques:3 Build a CNN having a pre-trained MobileNet as backbone to classify 10 classes.

Ans:3

```
from tensorflow.keras.applications import MobileNet

from tensorflow.keras.layers import Flatten, Dense

from tensorflow.keras.models import Model

mobilenet_model = mobilenet.MobileNet()

mobilenet_model.summary()

# Load mobilenet with pretrained weights

mobilenet_model = mobilenet.MobileNet(input_shape = (224, 224, 3), weights = 'imagenet',
include_top = False)


# Build a new model based on pre-trained mobilenet

inputs = mobilenet_model.inputs

x = mobilenet_model.output

x = Flatten()(x)

x = Dense(256, activation = 'relu')(x)

outputs = Dense(10, activation = 'softmax')(x)

model = Model(inputs, outputs, name = 'NewModel')

model.summary()
```

Ques: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.

Ans:4

To train and test a CNN with a pre-trained MobileNet model on the CIFAR-10 dataset, we need to make a few adjustments. CIFAR-10 consists of 32x32 RGB images, but MobileNet expects 224x224 RGB images. We will need to resize the images to match MobileNet's expected input size. Additionally, we

will use the pre-trained MobileNet model as the feature extractor and build a new classification head to suit CIFAR-10's 10 classes.
The code is:

```
import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.applications import MobileNet

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.preprocessing.image import img_to_array, array_to_img


# Load CIFAR-10 dataset

(trainX, trainY), (testX, testY) = cifar10.load_data()


# Normalize the image data to the range [0, 1]

trainX = trainX.astype('float32') / 255.0

testX = testX.astype('float32') / 255.0


# Resize the images to 224x224 (MobileNet input size)

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

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


# Convert labels to one-hot encoding

trainY = to_categorical(trainY, num_classes=10)

testY = to_categorical(testY, num_classes=10)


# Load the pre-trained MobileNet model without the top layer

mobilenet_model = MobileNet(input_shape=(224, 224, 3), weights='imagenet', include_top=False)


# Freeze the layers of MobileNet
```

```
for layer in mobilenet_model.layers:
    layer.trainable = False

# Build a new model based on the pre-trained MobileNet
inputs = mobilenet_model.inputs
x = mobilenet_model.output
x = layers.GlobalAveragePooling2D()(x)  # Global Average Pooling layer to reduce spatial dimensions
x = layers.Dense(256, activation='relu')(x)  # Fully connected layer
outputs = layers.Dense(10, activation='softmax')(x)  # 10 classes for CIFAR-10
model = models.Model(inputs, outputs)

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

# Train the model
model.fit(trainX_resized, trainY, batch_size=64, epochs=10, validation_split=0.1)

# Evaluate the model on the test dataset
test_loss, test_acc = model.evaluate(testX_resized, testY)
print(f"Test accuracy: {test_acc:.4f}")

# Make predictions on the test set
predictions = model.predict(testX_resized)

# Display the first 5 predictions and actual labels
for i in range(5):
    print(f"Predicted: {predictions[i].argmax()}, Actual: {testY[i].argmax()}")
```