Md. Rafat Hossain Reyal
Roll No: 1910576122
Session:2018-19
Year: 4th year
Semester:Even Semester
Department:Computer Science
Engineering

Question 1

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

Answer

Fully Connected Neural Network (FCNN)

Code

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

# Define FCNN Model
inputs = Input((28, 28, 1), name='InputLayer')  # Input layer with image shape (28, 28, 1)
x = Flatten()(inputs)  # Flatten the 2D image into a 1D vector
x = Dense(2, activation='relu')(x)  # Dense layer with 2 neurons and ReLU activation
x = Dense(4, activation='relu')(x)  # Dense layer with 4 neurons and ReLU activation
x = Dense(8, activation='relu')(x)  # Dense layer with 8 neurons and ReLU activation
x = Dense(16, activation='relu')(x)  # Dense layer with 16 neurons and ReLU activation
x = Dense(8, activation='relu')(x)  # Dense layer with 8 neurons and ReLU activation
x = Dense(4, activation='relu')(x)  # Dense layer with 4 neurons and ReLU activation
outputs = Dense(10, name='OutputLayer', activation='softmax')(x)  # Output layer with 10 neurons and softmax activation

# Compile and Summarize FCNN Model
fcnn_model = Model(inputs, outputs, name='Multi-Class-Classifier')
fcnn_model.summary()
```

Listing 1: FCNN Code

Explanation

- `Input((28, 28, 1), name='InputLayer')`: Defines the input layer for the model, expecting grayscale images of shape $(28, 28, 1)$.
- `Flatten()(inputs)`: Reshapes the input from a 2D matrix $(28, 28)$ into a flat 1D array of length $28 \times 28 = 784$.

- `Dense(2, activation='relu')(x)`: A fully connected layer with 2 neurons and ReLU activation ($f(x) = \max(0, x)$).
- **Additional Dense Layers**: Gradually increase and decrease the number of neurons to help learn complex patterns while avoiding overfitting.
- `Dense(10, activation='softmax')`: The output layer with 10 neurons, producing probabilities for each of the 10 classes.

Convolutional Neural Network (CNN)

Code

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

# Define CNN Model
cnn_inputs = Input((28, 28, 1), name='InputLayer')  # Input layer for images

x = Conv2D(filters=8, kernel_size=(5, 5), padding='same', activation='relu')(cnn_inputs)  # Conv layer with 8 filters
x = Conv2D(filters=8, kernel_size=(5, 5), padding='same', activation='relu')(x)  # Another Conv layer with 8 filters
x = MaxPooling2D()(x)  # MaxPooling layer to downsample feature maps

x = Conv2D(filters=16, kernel_size=(5, 5), activation='relu')(x)  # Conv layer with 16 filters
x = Conv2D(filters=16, kernel_size=(5, 5), activation='relu')(x)  # Another Conv layer with 16 filters
x = Conv2D(filters=16, kernel_size=(5, 5), strides=(2, 2), activation='relu')(x)  # Downsampling Conv layer

x = Flatten()(x)  # Flatten the feature maps to a 1D vector
x = Dense(8, activation='relu')(x)  # Fully connected layer with 8 neurons

cnn_outputs = Dense(10, name='OutputLayer', activation='softmax')(x)  # Output layer with softmax activation

# Compile and Summarize CNN Model
cnn_model = Model(cnn_inputs, cnn_outputs, name='CNN')
cnn_model.summary()
```

Listing 2: CNN Code

Explanation

- `Input((28, 28, 1), name='InputLayer')`: Similar to FCNN, the input layer expects grayscale images of shape $(28, 28, 1)$.
- `Conv2D(filters=8, kernel`$_{s}ize = (5,5), padding =' same', activation =' relu') : Aconvolutionallayerwith8filters, kernelsize(5,5)$`, and ReLU activation.`
- `MaxPooling2D(): A pooling layer that reduces the size of feature maps, summarizing information and improving efficiency.`
- `Additional Conv2D Layers: Use 16 filters to extract higher-level features such as corners or complex patterns.`
- `strides=(2, 2) in Conv2D: Downsamples the feature map by moving the filter 2 pixels horizontally and vertically.`
- `Flatten(): Transforms the 2D feature map into a flat vector for dense layers.`
- `Dense(8, activation='relu'): A dense layer processes the flattened features.`
- `Dense(10, activation='softmax'): Produces probabilities for each of the 10 classes.`

Question 2

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

Answer:

Step 1: Import Libraries

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

Listing 3: Import Libraries

Step 2: Define a Function to Display Sample 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()
```

Listing 4: Display Sample Images

Step 3: Load the Fashion MNIST Dataset

```
# Load dataset
(trainX, trainY), (testX, testY) = load_data()

# Investigate loaded data
print(f'trainX.shape: {trainX.shape}, trainY.shape: {trainY.shape
   }')
print(f'trainX.Range: {trainX.min()} to {trainX.max()}')

# Display sample images
display_img(trainX[:9], trainY[:9])
```

Listing 5: Load and Investigate Data

Step 4: Preprocess the Dataset

```
# Expand dimensions for grayscale images
trainX = np.expand_dims(trainX, axis=-1)
testX = np.expand_dims(testX, axis=-1)

# One-hot encode labels
trainY = to_categorical(trainY, num_classes=10)
testY = to_categorical(testY, num_classes=10)

# Normalize pixel values to the range [0, 1]
trainX = trainX.astype('float32') / 255
testX = testX.astype('float32') / 255
```

Listing 6: Preprocess the Data

Step 5: Define the Fully Connected Neural Network (FCNN)

```
fc_inputs = Input((28, 28, 1), name='FC_InputLayer')
x = Flatten()(fc_inputs)
x = Dense(512, activation='relu')(x)
x = Dense(4, activation='relu')(x)
x = Dense(8, activation='relu')(x)
x = Dense(16, activation='relu')(x)
x = Dense(8, activation='relu')(x)
x = Dense(4, activation='relu')(x)
```

```
fc_outputs = Dense(10, name='FC_OutputLayer', activation='softmax
    ')(x)
fc_model = Model(fc_inputs, fc_outputs, name='Fully-Connected-
    Classifier')
fc_model.summary()
```

Listing 7: Define FCNN

Step 6: Compile and Train FCNN

```
# Compile FCNN model
fc_model.compile(optimizer="rmsprop", loss='
    categorical_crossentropy', metrics=['accuracy'])

# Train FCNN model
fc_model.fit(trainX, trainY, batch_size=128, validation_split
    =0.1, epochs=10)
```

Listing 8: Compile and Train FCNN

Step 7: Evaluate FCNN

```
# Evaluate FCNN model
fc_score = fc_model.evaluate(testX, testY)

# Predict using FCNN
fc_predictions = fc_model.predict(testX)

# Display first 10 predictions
print('OriginalY   FCNN_PredictedY')
for i in range(10):
    true_label = np.argmax(testY[i])
    fc_pred = np.argmax(fc_predictions[i])
    print(f'{true_label:<10} {fc_pred}')
```

Listing 9: Evaluate FCNN

Step 8: Define the Convolutional Neural Network (CNN)

```
cnn_inputs = Input((28, 28, 1), name='InputLayer')

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

x = Conv2D(filters=16, kernel_size=(5, 5), activation='relu')(x)
x = Conv2D(filters=16, kernel_size=(5, 5), activation='relu')(x)
```

```python
x = Conv2D(filters=16, kernel_size=(5, 5), strides=(2, 2),
    activation='relu')(x)

x = Flatten()(x)
x = Dense(8, activation='relu')(x)

cnn_outputs = Dense(10, name='OutputLayer', activation='softmax')
    (x)
cnn_model = Model(cnn_inputs, cnn_outputs, name='CNN')
cnn_model.summary()
```

Listing 10: Define CNN

Step 9: Compile and Train CNN

```python
# Compile CNN model
cnn_model.compile(optimizer="adam", loss='
    categorical_crossentropy', metrics=['accuracy'])

# Train CNN model
cnn_model.fit(trainX, trainY, batch_size=128, validation_split
    =0.1, epochs=10)
```

Listing 11: Compile and Train CNN

Step 10: Evaluate CNN

```python
# Evaluate CNN model
cnn_score = cnn_model.evaluate(testX, testY)

# Predict using CNN
cnn_predictions = cnn_model.predict(testX)

# Display first 10 predictions
print('OriginalY   CNN_PredictedY')
for i in range(10):
    true_label = np.argmax(testY[i])
    cnn_pred = np.argmax(cnn_predictions[i])
    print(f'{true_label:<10} {cnn_pred}')
```

Listing 12: Evaluate CNN

Comparison of FCNN and CNN Results

1. Model Architectures

Fully Connected Neural Network (FCNN):

- Input Layer: Flattened 28×28 grayscale image.

- Dense layers with neuron counts of 512, 4, 8, 16, 8, and 4.
- Softmax output layer with 10 neurons for classification.
- **Total Parameters:** 404,378.

Convolutional Neural Network (CNN):

- Input Layer: $28 \times 28 \times 1$ grayscale image.
- Convolutional layers with 8 and 16 filters, kernel size 5×5, with max-pooling and downsampling strides.
- Dense layer with 8 neurons, followed by a softmax output layer with 10 neurons.
- **Total Parameters:** 18,090.

2. Training and Validation Performance

Fully Connected Neural Network (FCNN):

- Epoch 10 training accuracy: 87.59%.
- Validation accuracy: 86.82%.

Convolutional Neural Network (CNN):

- Epoch 10 training accuracy: 88.35%.
- Validation accuracy: 87.30%.

3. Testing Performance

Fully Connected Neural Network (FCNN):

- Test Accuracy: 86.59%.
- Test Loss: 0.4272.

Convolutional Neural Network (CNN):

- Test Accuracy: 86.69%.
- Test Loss: 0.3928.

4. Observations and Insights

- **Accuracy:** Both models achieved comparable performance in terms of test accuracy, with the CNN performing slightly better.
- **Loss:** The CNN demonstrated a marginally lower loss, indicating better confidence in its predictions.
- **Efficiency:** The CNN used significantly fewer parameters (4.5% of the FCNN's parameters) to achieve similar performance, making it more computationally efficient.
- **Architecture Fit:** CNNs are better suited for image data due to their ability to capture spatial relationships and features, as seen in their better validation loss and accuracy.

Question 3:

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

Answer:

Here is how you can do it:

```
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
# Load MobileNet as the base model
mobilenet_model = MobileNet(input_shape=(224, 224, 3),
    include_top=False, weights='imagenet')
Close
# Add custom layers for classification
inputs = mobilenet_model.input
x = mobilenet_model.output
x = Flatten(name='Flatten')(x)
x = Dense(256, activation='relu', name='DenseLayer1')(x)
outputs = Dense(10, activation='softmax', name='OutputLayer')(x)
# Create the final model
model = Model(inputs=inputs, outputs=outputs, name='
    MobileNet_Classifier')
# Display the model architecture
model.summary()
```

Question 4

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

Answer

To classify CIFAR-10 images using a CNN with a pre-trained MobileNet backbone, the workflow involves:

1. **Transfer Learning**: Using MobileNet as a feature extractor with frozen weights.
2. **Fine-Tuning**: Unfreezing some deeper layers of MobileNet to further improve performance.

Below is the implementation, broken into steps for clarity.

Step 1: Import Libraries and Load the Dataset

```
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.layers import Flatten, Dense, Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import cifar10
import tensorflow as tf

# Load CIFAR-10 dataset
(trainX, trainY), (testX, testY) = cifar10.load_data()

# Normalize pixel values to the range [0, 1]
trainX = trainX.astype('float32') / 255.0
testX = testX.astype('float32') / 255.0

# One-hot encode labels
trainY = to_categorical(trainY, num_classes=10)
testY = to_categorical(testY, num_classes=10)

print(f"Train shape: {trainX.shape}, Test shape: {testX.shape}")
```

Step 2: Build the Model with Pre-trained MobileNet

```
def build_mobilenet_cnn(input_shape, num_classes):
    # Load MobileNet as the backbone (pretrained on ImageNet)
    mobilenet = MobileNet(input_shape=input_shape, include_top=
    False, weights='imagenet')

    # Freeze all layers of the backbone for transfer learning
    for layer in mobilenet.layers:
        layer.trainable = False

    # Add custom classification head
    inputs = mobilenet.input
    x = mobilenet.output
    x = Flatten(name='Flatten')(x)
    x = Dense(256, activation='relu', name='DenseLayer1')(x)
    outputs = Dense(num_classes, activation='softmax', name='
    OutputLayer')(x)

    # Build the model
    model = Model(inputs, outputs, name='MobileNet_CNN')
    model.summary()
    return model
```

Step 3: Train the Model (Transfer Learning Only)

```
# Build and compile the model
```

```
mobilenet_cnn = build_mobilenet_cnn(input_shape=(32, 32, 3), num_classes=10)
mobilenet_cnn.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

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

# Evaluate the model
test_loss_transfer, test_acc_transfer = mobilenet_cnn.evaluate(testX, testY)
print(f"Test Accuracy (Transfer Learning Only): {test_acc_transfer}")
```

Step 4: Fine-Tune the Model

```
# Unfreeze deeper layers of the backbone for fine-tuning
for layer in mobilenet_cnn.layers[-20:]:
    layer.trainable = True

# Compile the model with a lower learning rate for fine-tuning
mobilenet_cnn.compile(optimizer=Adam(learning_rate=1e-4), loss='categorical_crossentropy', metrics=['accuracy'])

# Fine-tune the model
history_finetune = mobilenet_cnn.fit(trainX, trainY, validation_split=0.1, epochs=10, batch_size=32)

# Evaluate the fine-tuned model
test_loss_finetune, test_acc_finetune = mobilenet_cnn.evaluate(testX, testY)
print(f"Test Accuracy (Transfer Learning + Fine-Tuning): {test_acc_finetune}")
```

Performance Comparison: Transfer Learning vs Fine-Tuning

Transfer Learning Only

- **Validation Accuracy:** Starts at a low value and does not significantly improve over epochs.
- **Test Accuracy:** 23.47%.

Transfer Learning + Fine-Tuning

- **Validation Accuracy:** Improves consistently across epochs, reaching approximately 81.72%.
- **Test Accuracy:** 80.60%.

Observations

- **Transfer Learning Alone:** Performance is constrained by freezing the pretrained layers, limiting the model's ability to adapt to the CIFAR-10 dataset.
- **Fine-Tuning:** Unfreezing deeper layers and training them with a lower learning rate significantly enhances performance.

Accuracy Plot

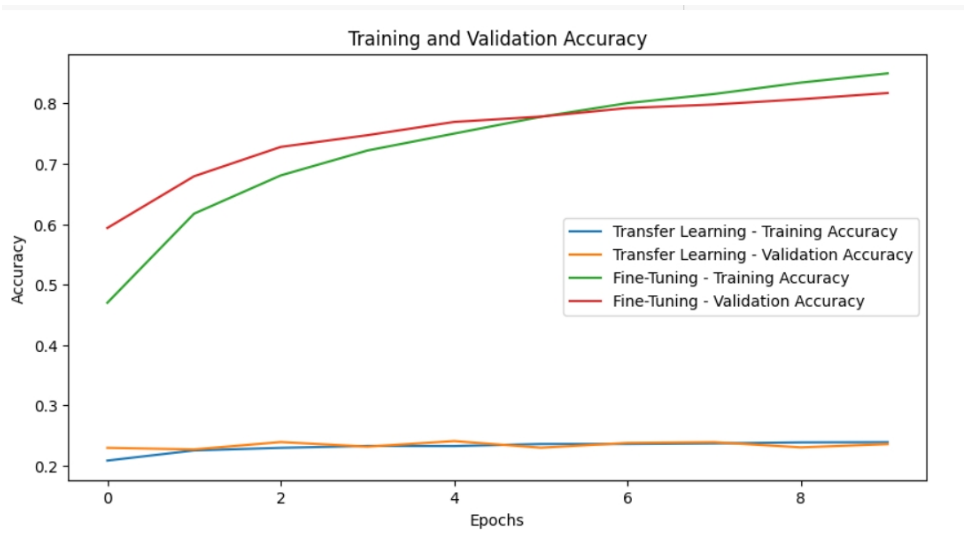

Figure 1: Training and Validation Accuracy for Transfer Learning and Fine-Tuning