KNOWLEDGE TEST

ATHUL K

Knowledge Test

Practical Questions (25 marks)

Activity-1

Aim:

Build and compile a simple neural network using Keras to classify the MNIST dataset (handwritten digits). The model should include at least one hidden layer. Provide the code and briefly explain each step.

Requirements:

- Python
- TensorFlow and Keras libraries (included with TensorFlow)
- MNIST dataset (available directly through Keras)
- VS code

Procedure

Step-1

First, we need to import the required libraries. Keras is part of TensorFlow, and we will use it to build our neural network.

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist

```

Step-2

Next, we load the MNIST dataset and preprocess it. The dataset is divided into training and test sets. We need to normalize the pixel values and convert the labels to categorical format.

```
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize the images
```

Step-3

We define a simple neural network using the Sequential API from Keras. The network includes an input layer (flattening the 2D images), a hidden layer with 128 neurons, and an output layer with 10 neurons (one for each digit class).

```python
# Build the model
model = models.Sequential()
model.add(layers.Flatten(input_shape=(28, 28)))  # Flatten the input
model.add(layers.Dense(128, activation='relu'))    # Hidden layer with ReLU activation
model.add(layers.Dense(10, activation='softmax'))   # Output layer for 10 classes
```

Step-4

We compile the model by specifying the loss function, optimizer, and metrics to monitor. For classification, we use the categorical crossentropy loss function and the Adam optimizer.

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

Step-5

We train the model using the fit method, specifying the training data, validation data, batch size, and number of epochs.

```python
# Train the model
model.fit(x_train, y_train, epochs=5)
```

Step-6

Evaluate the Model: Assess the model's performance on the test data.

```python
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

Output

```
  super().__init__(**kwargs)
2024-08-02 16:51:17.658851: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in perfor
mance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8794 - loss: 0.4277
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.9628 - loss: 0.1256
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9756 - loss: 0.0837
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9830 - loss: 0.0572
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9873 - loss: 0.0432
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9736 - loss: 0.0868
Test accuracy: 0.9771999716758728
PS D:\Daily_tasks\02-08-2024>
```

Activity-2

Aim:

Implement data augmentation on a given image dataset using Keras. Show at least three different augmentation techniques and explain how they help improve model performance.

Requirements:

- Python
- TensorFlow and Keras libraries (included with TensorFlow)
- VS code

Procedure

Step-1

Import the necessary libraries for data augmentation.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import os
```

Step-2

Load and Preprocess Data:

- Load a sample image from your dataset. In practice, you would use a directory of images. Here, we use a single image for demonstration.

```
# Define the path to your image
image_path = 'D:/Daily_tasks/02-08-2024/images.jpg'  # Replace with your image path
```

Step-3

Define Augmentation Techniques: Set up the ImageDataGenerator with different augmentation parameters.

```
# Check if the file exists
if not os.path.isfile(image_path):
    raise FileNotFoundError(f"Image file not found at path: {image_path}")

# Create an instance of ImageDataGenerator with multiple augmentation techniques
datagen = ImageDataGenerator(
    rotation_range=40,       # Randomly rotate images by up to 40 degrees
    width_shift_range=0.2,   # Randomly shift images horizontally by up to 20% of the width
    height_shift_range=0.2,  # Randomly shift images vertically by up to 20% of the height
    shear_range=0.2,         # Randomly apply shearing transformations
    zoom_range=0.2,          # Randomly zoom into images by up to 20%
    horizontal_flip=True,    # Randomly flip images horizontally
    fill_mode='nearest'      # Fill in pixels that are newly created during transformations
)
```

Step-4

Apply Augmentations: Generate augmented images from the original image.

```python
# Load and preprocess the image
image = tf.keras.preprocessing.image.load_img(image_path)
image = tf.keras.preprocessing.image.img_to_array(image)
image = np.expand_dims(image, axis=0)  # Convert image to a batch of size 1

# Apply augmentations
augmented_images = datagen.flow(image, batch_size=1)

# Plot the original and augmented images
plt.figure(figsize=(15, 15))

# Plot the original image
plt.subplot(1, 5, 1)
```

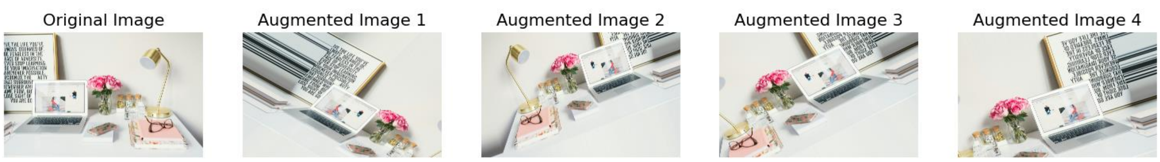
Step-5

Visualize Results:

- Plot the original image and several augmented images using matplotlib to visualize the effects of the applied augmentations.

```python

# Plot the original and augmented images
plt.figure(figsize=(15, 15))

# Plot the original image
plt.subplot(1, 5, 1)
plt.imshow(image[0].astype('uint8'))
plt.title('Original Image')
plt.axis('off')

# Plot a few augmented images
for i in range(4):
    plt.subplot(1, 5, i + 2)
    batch = next(augmented_images)  # Use next() to get the next batch
    augmented_image = batch[0].astype('uint8')
    plt.imshow(augmented_image)
    plt.title(f'Augmented Image {i+1}')
    plt.axis('off')

plt.show()
```

Out put



Activity-3

Aim:

Implement a custom loss function in TensorFlow/Keras. Explain the purpose of the loss function and provide an example scenario where it would be useful.

Requirements:

- Python
- TensorFlow and Keras libraries (included with TensorFlow)
- VS code

Procedure

Step-1

Import Libraries: Import TensorFlow and other necessary libraries.

```
import tensorflow as tf
from tensorflow.keras.losses import Loss
```

Step-2

Define the Custom Loss Function: Create a custom loss function by subclassing tf.keras.losses.Loss.

```
# Define a custom loss function
class CustomLoss(Loss):
    def __init__(self, threshold=1.0, **kwargs):
        super().__init__(**kwargs)
        self.threshold = threshold

    def call(self, y_true, y_pred):
```

Step-3

Create and Compile the Model: Define a simple neural network model and compile it using the custom loss function.

```
# Compute the absolute error
absolute_error = tf.abs(y_true - y_pred)
# Apply custom penalty: increase loss if error exceeds threshold
custom_loss = tf.where(absolute_error > self.threshold,
                       2 * absolute_error,
                       absolute_error)
return tf.reduce_mean(custom_loss)
```

Step-4

Generate Example Data: Create synthetic data for training the model.

```
# Example data
import numpy as np
X_train = np.random.rand(100, 10)
y_train = np.random.rand(100, 1)
```

Step-5

Train the Model: Train the model using the example data and the custom loss function.

```
# Train the model
history = model.fit(X_train, y_train, epochs=5)
```

Step-6

Review Results: Print the model summary and training history.

```
# Print the model summary
model.summary()

# Print the training history
print("Training history:")
print(history.history)
```

Output

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.2968
Epoch 3/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2828
Epoch 4/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.2665
Epoch 5/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2833
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                   │ (None, 10)             │           110 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 1)              │            11 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 365 (1.43 KB)
 Trainable params: 121 (484.00 B)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 244 (980.00 B)
Training history:
{'loss': [0.28788039088249207, 0.2846781313419342, 0.2824252247810364, 0.28025153279304504, 0.27975377440452576]}
PS D:\Daily_tasks\02-08-2024>
```

Activity-4

Aim:

Use a pre-trained model (such as VGG16 or ResNet) available in Keras for a simple image classification task. Fine-tune the model for a new dataset and describe the steps taken**Requirements:**

- Python
- TensorFlow and Keras libraries (included with TensorFlow)
- VS code

Procedure

Step-1

Import Libraries: Import TensorFlow and other necessary libraries.

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
# Load CIFAR-10 dataset
```

Step-2

Load and Prepare the Dataset: Load and preprocess the new dataset.

```
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

Step-3

Load the Pre-Trained Model: Load the pre-trained VGG16 model without the top layers.

```
# Normalize the images to the range [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Convert class vectors to binary class matrices
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
# Load the VGG16 model without the top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
```

Step-4

Freeze the Pre-Trained Layers: Prevent the pre-trained layers from being updated during training.

```
# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False
```

Step-5

Add Custom Layers: Add new layers to adapt the model to the new dataset

```
    layer.trainable = False
# Add custom layers
x = base_model.output
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
```

Step-6

Compile the Model: Define the optimizer, loss function, and evaluation metrics. Train the model on the new dataset. Assess the performance of the model on the test set.

```
# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
loss, accuracy = model.evaluate(x_test, y_test)
print(f'Test accuracy: {accuracy}')
```

Output

```
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 60s 38ms/step - accuracy: 0.6213 - loss: 1.0790 - val_accuracy: 0.5890 - val_loss: 1.1612
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 59s 38ms/step - accuracy: 0.6411 - loss: 1.0208 - val_accuracy: 0.6081 - val_loss: 1.1174
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 64s 41ms/step - accuracy: 0.6605 - loss: 0.9668 - val_accuracy: 0.6172 - val_loss: 1.0930
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 58s 37ms/step - accuracy: 0.6718 - loss: 0.9255 - val_accuracy: 0.6195 - val_loss: 1.1011
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 62s 40ms/step - accuracy: 0.6946 - loss: 0.8747 - val_accuracy: 0.6088 - val_loss: 1.1394
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 60s 38ms/step - accuracy: 0.7045 - loss: 0.8397 - val_accuracy: 0.6104 - val_loss: 1.1395
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 64s 41ms/step - accuracy: 0.7225 - loss: 0.7831 - val_accuracy: 0.6172 - val_loss: 1.1522
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 59s 38ms/step - accuracy: 0.7366 - loss: 0.7540 - val_accuracy: 0.6186 - val_loss: 1.1569
```

```
... 313/313 ━━━━━━━━━━━━━━━━━━━━ 10s 31ms/step - accuracy: 0.6166 - loss: 1.1521
    Test accuracy: 0.6186000108718872
```