Knowledge Test

NAVANI P

Activity 1

Building a Simple Neural Network

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

- Personal computer/laptop
- Google Collab

Procedure

1. **Import Necessary Libraries**

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
```

2. **Load and Preprocess the Data**

```python
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize the pixel values (0-255) to the range (0-1)
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

3. **Build the Neural Network Model**

```python
# Build the neural network model
model = Sequential([
    Flatten(input_shape=(28, 28)),   # Flattens the 2D image array into a 1D array
    Dense(128, activation='relu'),   # Hidden layer with 128 neurons and ReLU activation
    Dense(10, activation='softmax')  # Output layer with 10 neurons and softmax activation
])
```

4. **Compile the Model**

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

5. **Train and evaluate the Model**

```python
# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=32, validation_data=(x_test, y_test))

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

6. **Make predictions**

```python
# Make predictions (optional)
predictions = model.predict(x_test)
print(f'Predicted label for the first test sample: {np.argmax(predictions[0])}')
```

OUTPUT

```
PS C:\Users\USER\Desktop\exam> py q1.py
2024-08-02 15:25:08.828951: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results
rn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-08-02 15:25:13.168426: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results
rn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
C:\Users\USER\AppData\Roaming\Python\Python311\site-packages\keras\src\layers\reshaping\flatten.py:37: UserWarning: Do not pass an `input_shape`/`
Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
2024-08-02 15:25:24.818302: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instruct
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.8795 - loss: 0.4331 - val_accuracy: 0.9577 - val_loss: 0.1387
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9638 - loss: 0.1257 - val_accuracy: 0.9701 - val_loss: 0.0972
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9762 - loss: 0.0805 - val_accuracy: 0.9726 - val_loss: 0.0855
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9832 - loss: 0.0551 - val_accuracy: 0.9746 - val_loss: 0.0810
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9861 - loss: 0.0454 - val_accuracy: 0.9741 - val_loss: 0.0809
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9697 - loss: 0.0947
Test accuracy: 0.9740999937057495
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Predicted label for the first test sample: 7
PS C:\Users\USER\Desktop\exam>
```

Activity 2

Data Augmentation

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

- ➢ Personal computer/laptop
- ➢ Google Collab

Procedure

1. **Import Necessary Libraries**

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

2. **Define path**

```
# Define the path to your image
image_path = 'C:/Users/USER/Desktop/exam/images.jpg'  # Replace with your image path
```

3. **check the file**

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

3. **Create an instance of imageDataGenerator with multiple augmentation**

```python
# 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
)
```

4. load and preprocess the 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
```

5. apply the argumentaion

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

6. <u>plot the original and argumental images</u>

```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()
```

OUTPUT

Activity 3

Custom Loss Function

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

- Personal computer/laptop
- Google Collab

Procedure

1. Import Necessary Libraries

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

2. Function

```python
class CustomLoss(Loss):
    def __init__(self, alpha=0.1, **kwargs):
        super().__init__(**kwargs)
        self.alpha = alpha  # Regularization strength

    def call(self, y_true, y_pred):
        # Mean Squared Error
        mse = tf.reduce_mean(tf.square(y_true - y_pred))

        # Regularization Term: Penalizes predictions deviating from the mean of y_true
        y_true_mean = tf.reduce_mean(y_true)
        regularization_term = tf.reduce_mean(tf.square(y_pred - y_true_mean))

        # Combine MSE with the regularization term
        loss = mse + self.alpha * regularization_term
        return loss
```

3. example usage

```python
# Example Usage
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, activation='relu', input_shape=(5,)),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss=CustomLoss(alpha=0.5))
```

Output

```
PS C:\Users\USER\Desktop\exam> py q3.py
2024-08-02 16:09:36.752481: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computat
ion orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-08-02 16:09:38.485330: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computat
ion orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
C:\Users\USER\AppData\Roaming\Python\Python311\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer
using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2024-08-02 16:09:42.181044: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

ACTIVITY 4

Transfer Learning.

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

- ➢ Personal computer/laptop
- ➢ Google Collab

Procedure

1.Import 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
```

2

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

# 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)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 54s 0us/step
```

3.

```
# Load the VGG16 model without the top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

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

4.

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

# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)
```

5.

```
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))
```

6.

```
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 62s 39ms/step - accuracy: 0.4787 - loss: 1.4747 - val_accuracy: 0.5644 - val_loss: 1.2234
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 60s 38ms/step - accuracy: 0.5887 - loss: 1.1592 - val_accuracy: 0.5896 - val_loss: 1.1645
Epoch 3/10
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

<keras.src.callbacks.history.History at 0x2dd66d9a930>
```

7.

```
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 62s 39ms/step - accuracy: 0.4787 - loss: 1.4747 - val_accuracy: 0.5644 - val_loss: 1.2234
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 60s 38ms/step - accuracy: 0.5887 - loss: 1.1592 - val_accuracy: 0.5896 - val_loss: 1.1645
Epoch 3/10
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

<keras.src.callbacks.history.History at 0x2dd66d9a930>
```

OUTPUT.

```
loss, accuracy = model.evaluate(x_test, y_test)
print(f'Test accuracy: {accuracy}')
```

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