8/2/2024

KNOWLEDGE TEST

AKSHAY T
NSTI CALICUT

QUESTIONS

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.

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.

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.

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.

ANSWERS

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

- Computer
- Vs code
- Network

Procedure

1. Create a folder name as exam
2. Open vs code
3. Create a py file in that folder
4. Write the code in that file

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

```
# 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

```
# 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

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

5. Train and evaluate the Model

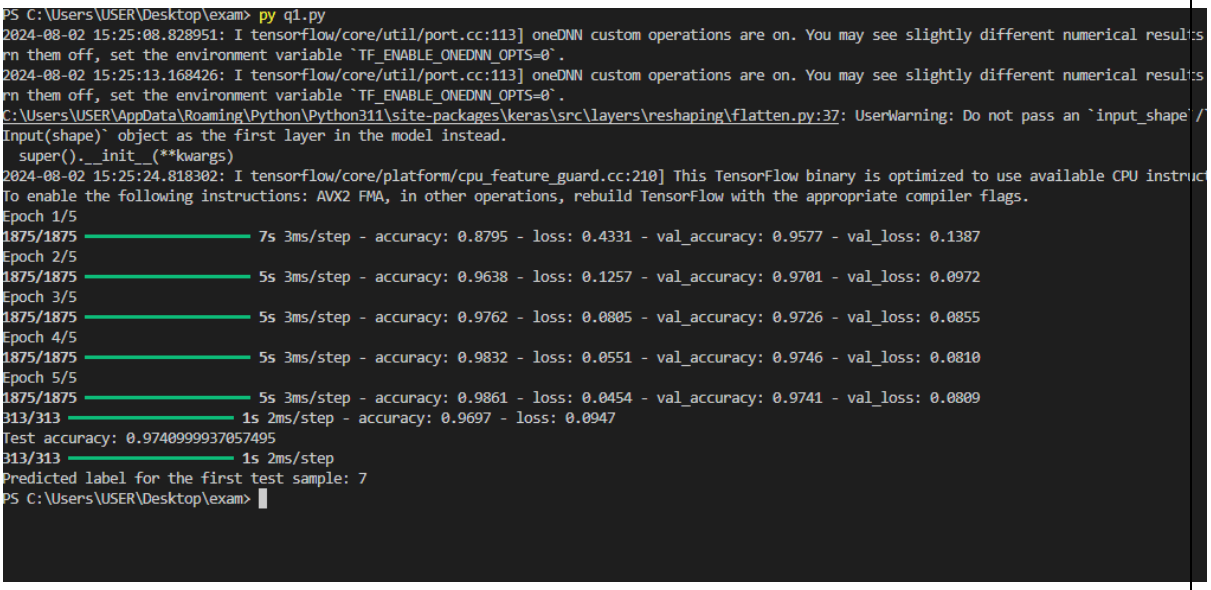
```
# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=32, validation_data=(x_test, y_test))

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

6. Make predictions

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

5.OutPut



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

- Computer
- Vs code
- Network

Procedure

1. Create a folder name as exam
2. Open vs code
3. Create a py file in that folder
4. Copy a image for data augmentaion
5. Write the code in that file

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 = r"C:\Users\user\OneDrive\Desktop\exam\car.jpg"  # Use raw string
```

3 . check the file

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

4.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
)
```

5.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
```

6.apply the argumentaion

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

7.plot the original and argumental images

```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

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

- Computer
- Vs code
- Network

Procedure

1.Create a folder name as exam

2.Open vs code

3.Create a py file in that folder

4.Write the code in that file

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.
```

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

- Computer
- Vs code
- Network

Procedure

1.Create a folder name as exam

2.Open vs code

3.Create a py file in that folder

4.Write the code in that file

1.Import Necessary Libraries

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
```

2.give paths

```
# Paths to your dataset directories
train_dir = 'C:/Users/USER/Desktop/exam/flower'
validation_dir = 'C:/Users/USER/Desktop/exam/flower'
```

3.create an imagedatagenerator for data augmentation

```
# Create an ImageDataGenerator for data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)
```

4.Load data

```
# Load data
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),  # Adjust based on model input size
    batch_size=32,
    class_mode='categorical'
)

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(224, 224),  # Adjust based on model input size
    batch_size=32,
    class_mode='categorical'
)
```

5.Train and Evaluate the model

```
    layer.trainable = False

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

# Step 4: Train the Model
history = model.fit(train_generator, epochs=5, validation_data=test_generator)

# Step 5: Unfreeze and Fine-Tune the Model
for layer in base_model.layers[-4:]:  # Unfreeze last 4 layers
    layer.trainable = True

# Recompile the model with a lower learning rate
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5), loss='categorical_crossentropy', metrics=['accuracy'])

# Fine-tune the model
history_fine_tune = model.fit(train_generator, epochs=5, validation_data=test_generator)

# Step 6: Evaluate the Model
eval_results = model.evaluate(test_generator)
print('Test Loss, Test Accuracy:', eval_results)

# Save the model
```

Output:

```
  self._warn_if_super_not_called()
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 352s 224ms/step - accuracy: 0.4064 - loss: 1.6585 - val_accuracy: 0.5402 - val_loss: 1.3129
Epoch 2/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 313s 200ms/step - accuracy: 0.5012 - loss: 1.4174 - val_accuracy: 0.5471 - val_loss: 1.2768
Epoch 3/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 320s 204ms/step - accuracy: 0.5162 - loss: 1.3680 - val_accuracy: 0.5747 - val_loss: 1.2094
Epoch 4/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 319s 204ms/step - accuracy: 0.5288 - loss: 1.3380 - val_accuracy: 0.5641 - val_loss: 1.2293
Epoch 5/5
```