8/2/2024

Knowledge Test

TIDKE ASHOK TATERAO
NSTI CALICUT

Knowledge Test

AIM:

Implement various deep learning tasks using Keras and TensorFlow, including building a neural network, data augmentation, custom loss functions, and transfer learning.

Requirements:

- Pc/Laptop
- VS Code
- Chrome
- Python installed with required libraries (Keras, TensorFlow, numpy, matplotlib)

Learning Outcome:

Apply and understand the steps involved in building neural networks, augmenting data, implementing custom loss functions, and using transfer learning for image classification tasks.

Procedure:

1. Building a Simple Neural Network

Step-1: Import necessary packages

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

Step-2: Load and preprocess the MNIST dataset

```
# 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  # Normaliz
```

Step-3: Build the neural network model

```
# 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: Compile and train the model

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

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

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

Output:

- Trained neural network on MNIST dataset with training and validation accuracy.

```
sing an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
2024-08-02 16:15:19.915221: I tensorflow/core/platform/cpu_feature_guard.cc:210] This Tens
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow wi
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8780 - loss: 0.4253
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9640 - loss: 0.1192
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9775 - loss: 0.0762
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9833 - loss: 0.0558
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9871 - loss: 0.0425
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 978us/step - accuracy: 0.9718 - loss: 0.0869
Test accuracy: 0.9760000109672546
PS D:\OJT\OJT_Practical exam\Knowledge Test\week 12>
```

2. Data Augmentation

Step-1: Import necessary packages for 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: Define the path to your image

```
# Define the path to your image
image_path = 'D:\OJT\OJT_Practical exam\Knowledge Test\week 12\data\doctors-1.jpg'  # Replace with your image path
```

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

Step-3: Create an instance of ImageDataGenerator with multiple augmentation techniques

```
# 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
)
```

```
# 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)
```

Step-4:plot for image

```
# 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:

- Augmented images demonstrating various augmentation techniques.

3. Custom Loss Function

Step-1: Define the custom loss function

```python
# custom_loss.py
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Define a custom loss function
def custom_loss(y_true, y_pred):
    return tf.reduce_mean(tf.square(y_true - y_pred))
```

Step-2: Build a simple model using the custom loss function

```python
# Build a simple model
model = Sequential([
    Dense(10, activation='relu', input_shape=(10,)),
    Dense(1)
])
```

Step-3: saved the model

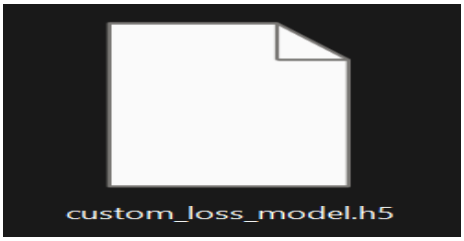
```python
# Compile the model with custom loss
model.compile(optimizer=Adam(), loss=custom_loss)

# Save the model
model.save('custom_loss_model.h5')
```

Output:

- Simple model using custom loss function with Mean Absolute Error (MAE) as a metric.

```
PS D:\OJT\OJT_Practical exam\Knowledge Test\week 12> py Custom_Loss_Function.py
2024-08-02 16:26:03.819372: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may
tation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-08-02 16:26:04.743095: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may
tation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
C:\Users\ashok\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\dense.py
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 16:26:07.224937: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropr
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mo
. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
PS D:\OJT\OJT_Practical exam\Knowledge Test\week 12>
```



4. Transfer Learning:

Step-1: Imported nececary libraries

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Step-2: Load and Preprocess CIFAR-10 Dataset

```
# Step 1: Load and Preprocess CIFAR-10 Dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize the images
x_train, x_test = x_train / 255.0, x_test / 255.0

# Convert labels to one-hot encoding
y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test, 10)
```

Step-3: Create ImageDataGenerators for training and testing

```
# Create ImageDataGenerators for training and testing
train_datagen = ImageDataGenerator(
    rotation_range=20,
    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()

train_generator = train_datagen.flow(x_train, y_train, batch_size=32)
test_generator = test_datagen.flow(x_test, y_test, batch_size=32)
```

Step-4: Load and Prepare the Pre-trained VGG16 Model

```
# Step 2: Load and Prepare the Pre-trained VGG16 Model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Add custom layers on top of the VGG16 base model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)  # 10 classes for CIFAR-10
```

Step-5: Define the complete model

```
# Define the complete model
model = Model(inputs=base_model.input, outputs=predictions)

# Step 3: Freeze the Pre-trained Layers
for layer in base_model.layers:
    layer.trainable = False
```

Step-6: Compile the model

```
# 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
```

Step-6: Recompile the model with a lower learning rate

```
# 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)
```

Step-7: Save the model

```
# Save the model
model.save('fine_tuned_vgg16_cifar10.h5')
```

Outputs:

- Fine-tuned pre-trained model for a new dataset with training and validation accuracy.

```
  self._warn_if_super_not_called()
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 233s 148ms/step - accuracy: 0.4116 - loss: 1.6526 - val_accuracy: 0.5409 - val_loss: 1.2997
Epoch 2/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 245s 157ms/step - accuracy: 0.4944 - loss: 1.4211 - val_accuracy: 0.5429 - val_loss: 1.2982
Epoch 3/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 238s 152ms/step - accuracy: 0.5199 - loss: 1.3660 - val_accuracy: 0.5353 - val_loss: 1.2951
Epoch 4/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 227s 145ms/step - accuracy: 0.5253 - loss: 1.3350 - val_accuracy: 0.5650 - val_loss: 1.2260
Epoch 5/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 270s 173ms/step - accuracy: 0.5309 - loss: 1.3176 - val_accuracy: 0.5735 - val_loss: 1.2169
Epoch 1/5
```