Knowledge test

ARCHANA G S
NSTI CALICUT

ACTIVITY 1

AIM :- Building a Simple Neural Network (10 marks) 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
2. Open vs code
3. Create a py file in that folder
4. Write the code in that file
5. Import Libraries:
   TensorFlow and Keras libraries are imported for building the model.
   ```
   import tensorflow as tf
   from tensorflow.keras import layers, models
   from tensorflow.keras.datasets import mnist
   ```
6. Load Dataset:
   The MNIST dataset is loaded and normalized to the range [0, 1].
   ```
   # 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  # Norm
   ```

7. Build Model:

   Flatten Layer: Converts the 28x28 images into a 1D array of 784 pixels.

   Dense Layer: A hidden layer with 128 neurons and ReLU activation function.

   Output Layer: A dense layer with 10 neurons (one for each digit) and softmax activation for multi-class classification.

```
# Build the model
model = models.Sequential()
model.add(layers.Flatten(input_shape=(28, 28)))  #
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

8. Compile Model:
   The model is compiled using the Adam optimizer and sparse categorical cross-entropy loss.

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

9. Train Model:
   The model is trained on the training data for 5 epochs.

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

10. Evaluate Model:
    The model's performance is evaluated on the test dataset, and the accuracy is printed.

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

Output

```
2024-08-02 17:05:58.612229: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is
ns in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlo
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8792 - loss: 0.4360
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9644 - loss: 0.1197
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9762 - loss: 0.0780
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9829 - loss: 0.0566
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9859 - loss: 0.0446
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9721 - loss: 0.0890
Test accuracy: 0.9761999845504761
PS C:\Users\User\Desktop\OJT\02.08.2024>
```

ACTIVITY 2

AIM :- Data Augmentation (5 marks) 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
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
6. Import Libraries:
   Necessary libraries for image processing and visualization are imported.

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

7. Check Image Path: The code checks if the specified image file exists.

```
# Define the path to your
image_path = 'images.jpg'
```

8. ImageDataGenerator:
   An instance is created with various augmentation techniques

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

# Create an instance of ImageDataGener
datagen = ImageDataGenerator(
    rotation_range=40,        # Random
    width_shift_range=0.2,    # Random
    height_shift_range=0.2,   # Random
    shear_range=0.2,          # Random
    zoom_range=0.2,           # Random
    horizontal_flip=True,     # Random
    fill_mode='nearest'       # Fill i
)
```

9. Load and Preprocess Image:
   The image is loaded and converted into a batch format.

```
# 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
```

10. Apply Augmentations: Augmented images are generated using the flow method.

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

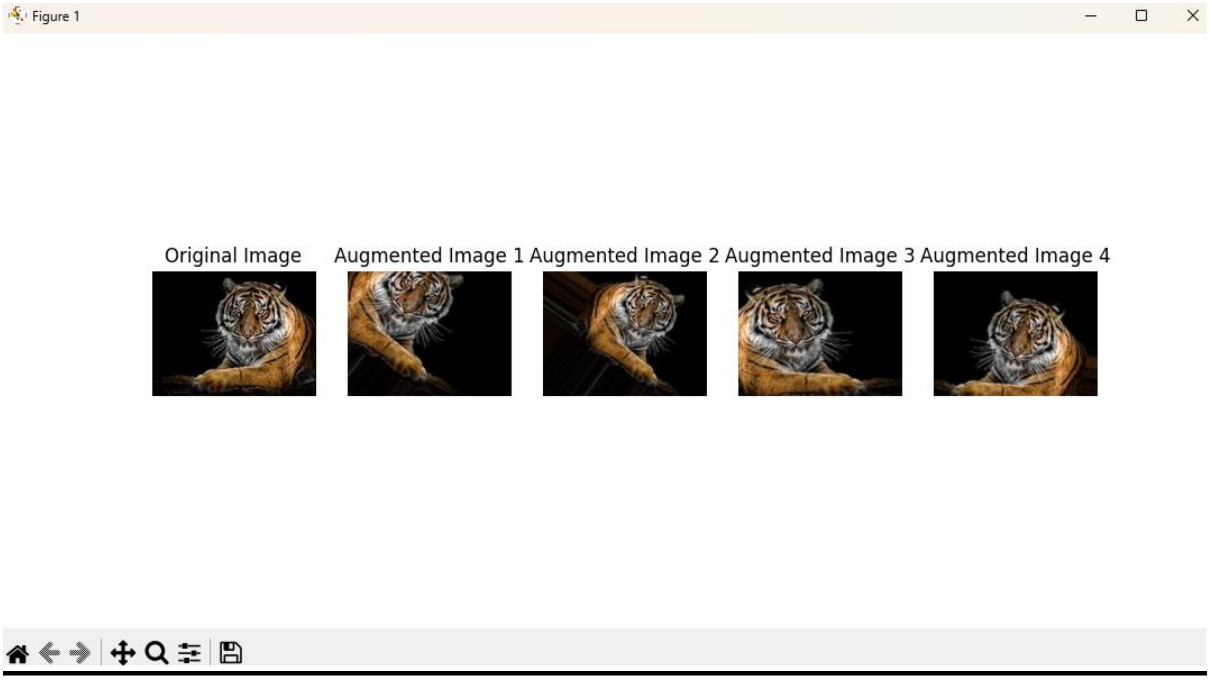
10.Plot Images:

The original and augmented images are displayed for comparison.

```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')
```

OUTPUT



ACTIVITY 3

AIM :- Custom Loss Function (5 marks) 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

2.Open vs code

3.Create a py file in that folder

4.Write the code in that file

5. Import Libraries:

TensorFlow and Keras libraries are imported.

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

2. Define Custom Loss Class: A class CustomLoss is defined, inheriting from Keras' Loss class.

```
# Define a custom loss function
Codeium: Refactor | Explain
class CustomLoss(Loss):
    Codeium: Refactor | Explain | Generate Docstring | ×
    def __init__(self, threshold=1.0, **kwargs):
        super().__init__(**kwargs)
        self.threshold = threshold

    Codeium: Refactor | Explain | Generate Docstring | ×
    def call(self, y_true, y_pred):
        # 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)
```

3. Model Definition:

A simple sequential model is created with a dense layer.

```
# 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))
```

4. Train Model: The model is trained for 5 epochs.

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

Output

```
2024-08-02 17:24:10.525392: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly di
e to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `
2024-08-02 17:24:13.874261: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to
ns in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the app
Epoch 1/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.4749
Epoch 2/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 0s/step - loss: 0.4751
Epoch 3/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.4325
Epoch 4/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.4205
Epoch 5/5
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 0s/step - loss: 0.4101
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': [0.493704617023468, 0.4624812602996826, 0.43306484818458557, 0.4153471291065216, 0.39721
PS C:\Users\User\Desktop\OJT\02.08.2024>
```

Battery saver

ACTIVITY 4

AIM :- Transfer Learning (5 marks) 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

5.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
```

6.give paths

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

7.create an imagedatagenerator for data augmentation

```python
# 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)
```

8.load data

```python
# 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'
)
```

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

```python
# 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)
```

✓ 1m 11.4s

11. **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
```

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

```
# 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)

✓ 0.0s
```

13. 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)

0.0s

model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

0.0s

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

8m 54.9s
```

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
```