**Exercise:7**

import tensorflow as tf  
from tensorflow.keras import layers, models  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
from tensorflow.keras.applications import VGG16  
import numpy as np  
import matplotlib.pyplot as plt  
from tensorflow.keras.preprocessing import image  
  
# Define the class names  
class\_names = ['Cats', 'Dogs'] # Update with your actual class names  
  
# Define the directory containing training images  
train\_dir = r'C:\Users\LENOVO\PycharmProjects\nn\train'  
  
# Define data augmentation parameters for training data  
train\_datagen = ImageDataGenerator(  
 rescale=1./255, # Normalize pixel values to [0, 1]  
 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'  
)  
  
# Load and augment training data  
train\_generator = train\_datagen.flow\_from\_directory(  
 train\_dir,  
 target\_size=(150, 150),  
 batch\_size=32,  
 class\_mode='categorical'  
)  
  
# Load the pre-trained VGG16 model (excluding the top layer)  
base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(150, 150, 3))  
  
# Freeze some layers of the base model (optional)  
for layer in base\_model.layers:  
 layer.trainable = False  
  
# Add custom classification layers  
x = layers.Flatten()(base\_model.output)  
x = layers.Dense(256, activation='relu')(x)  
x = layers.Dropout(0.5)(x)  
predictions = layers.Dense(len(class\_names), activation='softmax')(x)  
  
# Create the transfer learning model  
transfer\_model = models.Model(inputs=base\_model.input, outputs=predictions)  
  
# Compile the model  
transfer\_model.compile(optimizer='adam',  
 loss='categorical\_crossentropy',  
 metrics=['accuracy'])  
  
# Print the summary of the model  
transfer\_model.summary()  
  
# Train the model  
print("Training started...")  
history = transfer\_model.fit(train\_generator, epochs=10)  
print("Training completed.")  
  
# Save the trained model  
print("Saving the model...")  
transfer\_model.save(r'C:\Users\LENOVO\PycharmProjects\nn\transfer\_learning\_model.h5')  
print("Model saved successfully.")  
  
# Use the model for predictions  
print("Making predictions...")  
# Load an image for classification  
img\_path = r'C:\Users\LENOVO\PycharmProjects\nn\pet.jpg' # Update with the path to the image you want to classify  
img = image.load\_img(img\_path, target\_size=(150, 150)) # Resize images to match the input size expected by the model  
  
# Preprocess the image  
img\_array = image.img\_to\_array(img)  
img\_array = np.expand\_dims(img\_array, axis=0)  
img\_array /= 255.0 # Normalize pixel values to [0, 1]  
  
# Make predictions  
predictions = transfer\_model.predict(img\_array)  
predicted\_class = np.argmax(predictions[0])  
predicted\_class\_name = class\_names[predicted\_class]  
  
# Visualize the result  
plt.imshow(img)  
plt.axis('off')  
plt.title('Predicted Class: {}'.format(predicted\_class\_name)) # Corrected this line  
plt.show()  
print("Prediction completed.")

Output:



C:\Users\LENOVO\PycharmProjects\nn\venv\Scripts\python.exe C:\Users\LENOVO\PycharmProjects\nn\n.py

2024-03-21 23:22:39.027148: 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 computation orders. To turn them off, set the environment variable `TF\_ENABLE\_ONEDNN\_OPTS=0`.

2024-03-21 23:22:39.635038: 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 computation orders. To turn them off, set the environment variable `TF\_ENABLE\_ONEDNN\_OPTS=0`.

Found 10 images belonging to 2 classes.

2024-03-21 23:22:41.213367: 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 AVX512F AVX512\_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.

Model: "functional\_1"

┌─────────────────────────────────┬────────────────────────┬───────────────┐

│ Layer (type) │ Output Shape │ Param # │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ input\_layer (InputLayer) │ (None, 150, 150, 3) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block1\_conv1 (Conv2D) │ (None, 150, 150, 64) │ 1,792 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block1\_conv2 (Conv2D) │ (None, 150, 150, 64) │ 36,928 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block1\_pool (MaxPooling2D) │ (None, 75, 75, 64) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block2\_conv1 (Conv2D) │ (None, 75, 75, 128) │ 73,856 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block2\_conv2 (Conv2D) │ (None, 75, 75, 128) │ 147,584 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block2\_pool (MaxPooling2D) │ (None, 37, 37, 128) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block3\_conv1 (Conv2D) │ (None, 37, 37, 256) │ 295,168 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block3\_conv2 (Conv2D) │ (None, 37, 37, 256) │ 590,080 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block3\_conv3 (Conv2D) │ (None, 37, 37, 256) │ 590,080 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block3\_pool (MaxPooling2D) │ (None, 18, 18, 256) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block4\_conv1 (Conv2D) │ (None, 18, 18, 512) │ 1,180,160 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block4\_conv2 (Conv2D) │ (None, 18, 18, 512) │ 2,359,808 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block4\_conv3 (Conv2D) │ (None, 18, 18, 512) │ 2,359,808 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block4\_pool (MaxPooling2D) │ (None, 9, 9, 512) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block5\_conv1 (Conv2D) │ (None, 9, 9, 512) │ 2,359,808 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block5\_conv2 (Conv2D) │ (None, 9, 9, 512) │ 2,359,808 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block5\_conv3 (Conv2D) │ (None, 9, 9, 512) │ 2,359,808 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ block5\_pool (MaxPooling2D) │ (None, 4, 4, 512) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ flatten (Flatten) │ (None, 8192) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense (Dense) │ (None, 256) │ 2,097,408 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dropout (Dropout) │ (None, 256) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_1 (Dense) │ (None, 2) │ 514 │

└─────────────────────────────────┴────────────────────────┴───────────────┘

Total params: 16,812,610 (64.14 MB)

Trainable params: 2,097,922 (8.00 MB)

Non-trainable params: 14,714,688 (56.13 MB)

Training started...

Epoch 1/10

C:\Users\LENOVO\PycharmProjects\nn\venv\lib\site-packages\keras\src\trainers\data\_adapters\py\_dataset\_adapter.py:120: UserWarning: Your `PyDataset` class should call `super().\_\_init\_\_(\*\*kwargs)` in its constructor. `\*\*kwargs` can include `workers`, `use\_multiprocessing`, `max\_queue\_size`. Do not pass these arguments to `fit()`, as they will be ignored.

self.\_warn\_if\_super\_not\_called()

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.3000 - loss: 1.0154

Epoch 2/10

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 372ms/step - accuracy: 0.6000 - loss: 0.7410

Epoch 3/10

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 365ms/step - accuracy: 0.5000 - loss: 1.7815

Epoch 4/10

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 368ms/step - accuracy: 0.6000 - loss: 1.2147

Epoch 5/10

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 361ms/step - accuracy: 0.8000 - loss: 0.4705

Epoch 6/10

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 386ms/step - accuracy: 0.5000 - loss: 1.3349

Epoch 7/10

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 363ms/step - accuracy: 0.6000 - loss: 1.8989

Epoch 8/10

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 375ms/step - accuracy: 1.0000 - loss: 0.1202

Epoch 9/10

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 363ms/step - accuracy: 1.0000 - loss: 0.0697

Epoch 10/10

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 372ms/step - accuracy: 0.8000 - loss: 0.7076

Training completed.

Saving the model...

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save\_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')` or `keras.saving.save\_model(model, 'my\_model.keras')`.

Model saved successfully.

Making predictions...

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step

Prediction completed.

Process finished with exit code 0

**Exercise:8**

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import VGG16

from tensorflow.keras.preprocessing import image

import numpy as np

import matplotlib.pyplot as plt

# Define the class names

class\_names = ['Cats', 'Dogs'] # Update with your actual class names

# Define the directory containing training images

train\_dir = r'C:\Users\LENOVO\PycharmProjects\nn\train'

# Define data augmentation parameters for training data

train\_datagen = ImageDataGenerator(

rescale=1./255, # Normalize pixel values to [0, 1]

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'

)

# Load and augment training data

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(150, 150),

batch\_size=32,

class\_mode='binary' # Use 'binary' for binary classification

)

# Load the pre-trained VGG16 model (excluding the top layer)

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(150, 150, 3))

# Freeze some layers of the base model (optional)

for layer in base\_model.layers:

layer.trainable = False

# Add custom classification layers

x = layers.Flatten()(base\_model.output)

x = layers.Dense(256, activation='relu')(x)

x = layers.Dropout(0.5)(x)

predictions = layers.Dense(len(class\_names), activation='softmax')(x)

# Create the transfer learning model

transfer\_model = models.Model(inputs=base\_model.input, outputs=predictions)

# Compile the model

transfer\_model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy', # Use 'sparse\_categorical\_crossentropy' for binary classification

metrics=['accuracy'])

# Print the summary of the model

transfer\_model.summary()

# Train the model

print("Training started...")

history = transfer\_model.fit(train\_generator, epochs=10)

print("Training completed.")

# Save the trained model

print("Saving the model...")

transfer\_model.save(r'C:\Users\LENOVO\PycharmProjects\nn\transfer\_learning\_model1.keras')

print("Model saved successfully.")

# Making predictions on new data

# Load an image for prediction

img\_path = r'C:\Users\LENOVO\PycharmProjects\nn\pet.jpg'

img = image.load\_img(img\_path, target\_size=(150, 150))

# Preprocess the image

img\_array = image.img\_to\_array(img)

img\_array = np.expand\_dims(img\_array, axis=0)

img\_array /= 255.0 # Normalize pixel values to [0, 1]

# Make predictions

print("Making predictions...")

predictions = transfer\_model.predict(img\_array)

# Interpret the predictions

predicted\_class = np.argmax(predictions[0])

predicted\_class\_name = class\_names[predicted\_class]

# Visualize the result

plt.imshow(img)

plt.axis('off')

plt.title('Predicted Class: {}'.format(predicted\_class\_name))

plt.show()