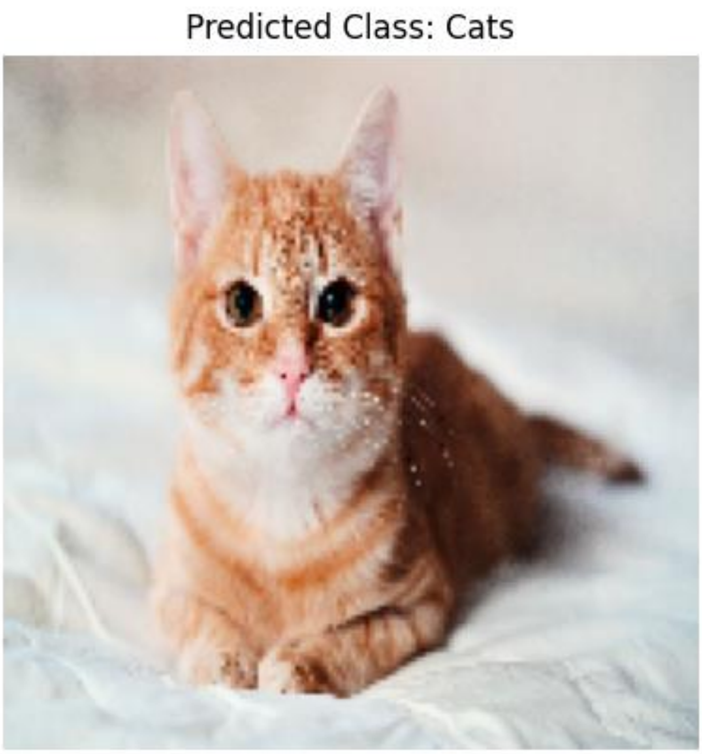
Exercise:7

```
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()
```