```
# Import necessary libraries
import tensorflow as tf
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.utils import get_file
print("Demo by G.Pavan Sai(22a81a6119)")
```

```
Demo by G.Pavan Sai(22a81a6119)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os
import zipfile
#Download and save in google drive from
#URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
# Define the dataset path inside Google Drive
google_drive_path = "/content/drive/MyDrive/cats_and_dogs_filtered.zip"
```

```
extract_path = "/content/cats_and_dogs_filtered"
```

```
# Verify if the dataset exists
if not os.path.exists(google_drive_path):
    print("❌ Dataset file not found! Check the path in Google Drive.")
else:
    print("✅ Dataset found in Google Drive!")
```

```
✅ Dataset found in Google Drive!
```

```
if not os.path.exists(extract_path):
    print("🔄 Extracting dataset... Please wait.")
    with zipfile.ZipFile(google_drive_path, 'r') as zip_ref:
        zip_ref.extractall("/content")
    print("✅ Dataset extracted successfully!")
else:
    print("📂 Dataset already extracted.")
```

```
🔄 Extracting dataset... Please wait.
✅ Dataset extracted successfully!
```

```
train_dir = os.path.join(extract_path, 'train')
validation_dir = os.path.join(extract_path, 'validation')
```

```
if not os.path.exists(train_dir) or not os.path.exists(validation_dir):
    print("❌ Training or validation directories are missing!")
else:
    print("✅ Training and validation directories exist.")
    print("📂 Training folder contents:", os.listdir(train_dir))
    print("📂 Validation folder contents:", os.listdir(validation_dir))
```

```
✅ Training and validation directories exist.
📂 Training folder contents: ['cats', 'dogs']
📂 Validation folder contents: ['cats', 'dogs']
```

```
import cv2
import matplotlib.pyplot as plt

# Define a sample image path (change 'cats' to 'dogs' if needed)
sample_image_path = os.path.join(train_dir, 'cats', os.listdir(os.path.join(train_dir, 'cats'))[0])

# Load the image using OpenCV
img = cv2.imread(sample_image_path)

# Check if the image is loaded correctly
if img is None:
    print("❌ Image not loaded! Check the file path.")
else:
    print("✅ Image loaded successfully!")
    print("📏 Image Shape:", img.shape)  # (Height, Width, Channels)

    # Convert BGR to RGB (OpenCV loads images in BGR format)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Resize image to match VGG-16 input size (224x224)
    img_resized = cv2.resize(img_rgb, (224, 224))

    # Display the image using Matplotlib (since cv2.imshow() does not work in Colab)
    plt.imshow(img_resized)
    plt.axis("off")  # Hide axes
    plt.title("Sample Cat Image By Pavan(22a81a6119) (Resized to 224x224)")
    plt.show()
```

```
✅ Image loaded successfully!
📏 Image Shape: (499, 436, 3)
```

Sample Cat Image By Pavan(22a81a6119) (Resized to 224x224)

```
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1.0/255.0,  # Normalize pixel values (0-255 → 0-1)
    rotation_range=40,  # Rotate images randomly
    width_shift_range=0.2,  # Horizontal shift
    height_shift_range=0.2,  # Vertical shift
    shear_range=0.2,  # Shearing transformation
    zoom_range=0.2,  # Zoom in/out
    horizontal_flip=True,  # Flip images horizontally
    fill_mode='nearest'  # Fill missing pixels after transformation
)

# Validation dataset: Only rescale (no augmentation)
validation_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1.0/255.0)

# Load images from directories
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),  # Resize images to VGG-16 input size
    batch_size=32,
    class_mode='binary'  # Binary classification (cats vs dogs)
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

```
# Load VGG-16 without the top classification layer
base_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the convolutional base (prevents modification of pre-trained weights)
base_model.trainable = False

# Print the base model summary
base_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Model: "vgg16"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)             │ (None, 224, 224, 3)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block1_conv1 (Conv2D)                │ (None, 224, 224, 64)        │           1,792 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block1_conv2 (Conv2D)                │ (None, 224, 224, 64)        │          36,928 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block1_pool (MaxPooling2D)           │ (None, 112, 112, 64)        │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block2_conv1 (Conv2D)                │ (None, 112, 112, 128)       │          73,856 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block2_conv2 (Conv2D)                │ (None, 112, 112, 128)       │         147,584 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block2_pool (MaxPooling2D)           │ (None, 56, 56, 128)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block3_conv1 (Conv2D)                │ (None, 56, 56, 256)         │         295,168 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block3_conv2 (Conv2D)                │ (None, 56, 56, 256)         │         590,080 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block3_conv3 (Conv2D)                │ (None, 56, 56, 256)         │         590,080 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block3_pool (MaxPooling2D)           │ (None, 28, 28, 256)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block4_conv1 (Conv2D)                │ (None, 28, 28, 512)         │       1,180,160 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block4_conv2 (Conv2D)                │ (None, 28, 28, 512)         │       2,359,808 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block4_conv3 (Conv2D)                │ (None, 28, 28, 512)         │       2,359,808 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block4_pool (MaxPooling2D)           │ (None, 14, 14, 512)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block5_conv1 (Conv2D)                │ (None, 14, 14, 512)         │       2,359,808 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block5_conv2 (Conv2D)                │ (None, 14, 14, 512)         │       2,359,808 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block5_conv3 (Conv2D)                │ (None, 14, 14, 512)         │       2,359,808 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ block5_pool (MaxPooling2D)           │ (None, 7, 7, 512)           │               0 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
```

```
model = tf.keras.Sequential([
    base_model,  # Use VGG-16 as a feature extractor
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),  # Dropout to reduce overfitting
    tf.keras.layers.Dense(1, activation='sigmoid')  # Output layer for binary classification
])
```

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

# Print model summary
model.summary()
```

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ vgg16 (Functional)                   │ (None, 7, 7, 512)           │      14,714,688 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten (Flatten)                    │ (None, 25088)               │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense (Dense)                        │ (None, 512)                 │      12,845,568 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout (Dropout)                    │ (None, 512)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                      │ (None, 1)                   │             513 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 27,560,769 (105.14 MB)
```

```
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=2,
    verbose=1
)
```

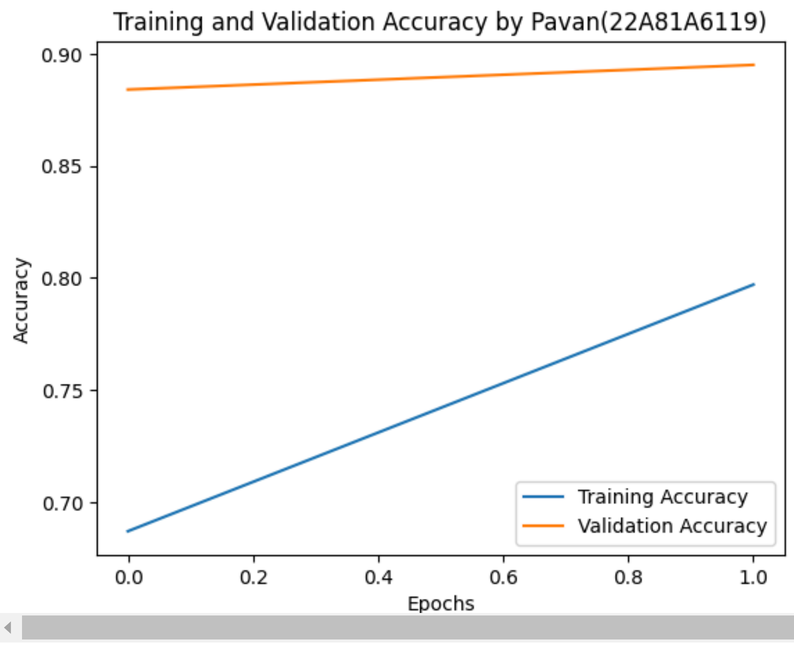
```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)
  self._warn_if_super_not_called()
Epoch 1/2
63/63 ━━━━━━━━━━━━━━━━━━━━ 62s 721ms/step - accuracy: 0.6264 - loss: 2.1554 - val_accuracy: 0.8840 - val_loss: 0.3046
Epoch 2/2
63/63 ━━━━━━━━━━━━━━━━━━━━ 38s 606ms/step - accuracy: 0.7894 - loss: 0.4500 - val_accuracy: 0.8950 - val_loss: 0.2539
```

```
test_loss, test_acc = model.evaluate(validation_generator)
print(f"\n✅ Model Test Accuracy: {test_acc:.2f}")
```

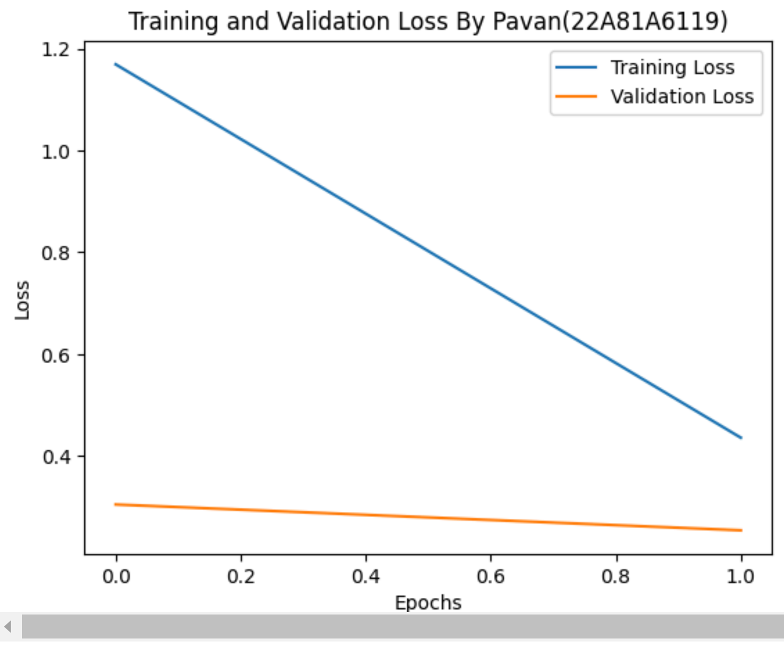
```
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 169ms/step - accuracy: 0.8952 - loss: 0.2546

✅ Model Test Accuracy: 0.89
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy by Pavan(22A81A6119)')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss By Pavan(22A81A6119)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```


Training and Validation Loss By Pavan(22A81A6119)
Loss
Epochs
Training Loss
Validation Loss