```
pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (1.6.17)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2025.1.31)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from kaggle) (4.67.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.3.0)
Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from kaggle) (6.2.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.11/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->kaggle) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->kaggle) (3.10)
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
cp: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
```

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
!ls /content/drive/MyDrive/Cat\ vs\ Dog
```

```
kagglecatsanddogs_5340  kaggle.json
```

```
from scipy.io import loadmat
import numpy as np
```

Double-click (or enter) to edit

```
import os
path, dirs, files = next(os.walk("/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages"))
file_count = len(files)
print("Number of images", file_count)
```

```
Number of images 3891
```

```
!ls "/content/drive/MyDrive/Cat vs Dog/"
```

```
kagglecatsanddogs_5340  kaggle.json
```

```
!cd /content/drive/MyDrive/ && ls -R
```

```
.:
 1285134779158_htwfaip.pdf
 20230110_224128_0000.png
 30-FREE-LUTS-FOR-ADOBE-PREMIERE-PRO
 AadharN.jpg
 AadharN.pdf
'ABC Physics Book.pdf'
'ABC Physics solution.pdf'
'Abhyuday physics.pdf'
'Book 1 - Twilight.pdf'
'Book 2 - New Moon.pdf'
'Book 3 - Eclipse.pdf'
'Book 4 - Breaking Dawn.pdf'
 can-love-happentwice-ebook-full-version-download-pdf-ravinder-singh2.pdf
'Cat vs Dog'
'Codewar 51-Nancy Gupta.png'
'Colab Notebooks'
'DLD Assignment-5.pdf'
'Getting started.pdf'
 IMG-20220928-WA0004.jpg
 inbound5075307809552223482.jpg
 inbound552318024288779044.pptx
```

```
'Lec 1 hydrogen_.pdf'
'Lec 2 hydrogen .pdf'
'Nancy Gupta  CERTIFICATE.pdf'
"Nancy Gupta's cv resume-1.pdf"
'Pdf 3 Hydrogen .pdf'
 power-subconscious-mind.pdf
'Sound Effects'
'Thermodynamics_last_Ankit Gaur Sir.pdf'
'TypeWriter Text.zip'
 Unknown
'Vaccination certificate.pdf'
"Web 3'O clock assignment.mp4"

'./Cat vs Dog':
kagglecatsanddogs_5340  kaggle.json

'./Cat vs Dog/kagglecatsanddogs_5340':
 CDLA-Permissive-2.0.pdf   New   PetImages  'readme[1].txt'

'./Cat vs Dog/kagglecatsanddogs_5340/New':
'cat (1000).jpg'  'cat (583).jpg'   'dog (1148).jpg'  'dog (1706).jpg'  'dog (446).jpg'
'cat (1001).jpg'  'cat (584).jpg'   'dog (1149).jpg'  'dog (1707).jpg'  'dog (447).jpg'
'cat (1002).jpg'  'cat (585).jpg'   'dog (114).jpg'   'dog (1708).jpg'  'dog (448).jpg'
'cat (1003).jpg'  'cat (586).jpg'   'dog (1150).jpg'  'dog (1709).jpg'  'dog (449).jpg'
'cat (1004).jpg'  'cat (587).jpg'   'dog (1151).jpg'  'dog (170).jpg'   'dog (44).jpg'
'cat (1005).jpg'  'cat (588).jpg'   'dog (1152).jpg'  'dog (1710).jpg'  'dog (450).jpg'
'cat (1006).jpg'  'cat (589).jpg'   'dog (1153).jpg'  'dog (1711).jpg'  'dog (451).jpg'
'cat (1007).jpg'  'cat (58).jpg'    'dog (1154).jpg'  'dog (1712).jpg'  'dog (452).jpg'
'cat (1008).jpg'  'cat (590).jpg'   'dog (1155).jpg'  'dog (1713).jpg'  'dog (453).jpg'
'cat (1009).jpg'  'cat (591).jpg'   'dog (1156).jpg'  'dog (1714).jpg'  'dog (454).jpg'
'cat (100).jpg'   'cat (592).jpg'   'dog (1157).jpg'  'dog (1715).jpg'  'dog (455).jpg'
'cat (1010).jpg'  'cat (593).jpg'   'dog (1158).jpg'  'dog (1716).jpg'  'dog (456).jpg'
'cat (1011).jpg'  'cat (594).jpg'   'dog (1159).jpg'  'dog (1717).jpg'  'dog (457).jpg'
'cat (1012).jpg'  'cat (595).jpg'   'dog (115).jpg'   'dog (1718).jpg'  'dog (458).jpg'
'cat (1013).jpg'  'cat (596).jpg'   'dog (1160).jpg'  'dog (1719).jpg'  'dog (459).jpg'
'cat (1014).jpg'  'cat (597).jpg'   'dog (1161).jpg'  'dog (171).jpg'   'dog (45).jpg'
```

```
# Printing hte names of images
files_names = os.listdir("/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages")
print(files_names)
```

```
['dog (947).jpg', 'dog (944).jpg', 'dog (945).jpg', 'dog (941).jpg', 'dog (943).jpg', 'dog (942).jpg', 'dog (937).jpg', 'dog (938).jpg',
```

Importing the dependencies

Double-click (or enter) to edit

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
from google.colab.patches import cv2_imshow

file_path = '/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages/cat (1).jpg'
img = mpimg.imread(file_path)
plt.imshow(img)
plt.show()
```

Cat's image

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
from google.colab.patches import cv2_imshow

file_path = '/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages/dog (19).jpg'
img = mpimg.imread(file_path)
plt.imshow(img)
plt.show()
```

Resizing all the images

```
import os

# Path to the directory
file_names = '/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages'

# Check if the directory exists
if os.path.exists(file_names):
    # List all files in the directory
    files = os.listdir(file_names)

    for i in range(5):
        if i < len(files):  # Ensure we don't go out of bounds
            print(files[i])
```

```
else:
    print(f"The directory {file_names} does not exist.")
```

```
 dog (947).jpg
 dog (944).jpg
 dog (945).jpg
 dog (941).jpg
 dog (943).jpg
```

```
import os
path, dirs, files=next(os.walk("/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages"))
file_count=len(files)
print('Number of files ',file_count)
```

```
Number of files  3891
```

```
import os
file_names = os.listdir("/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages")
dog_count = 0
cat_count = 0

for img_files in file_names:
      name = img_files[0:3]
      if name == "dog":
            dog_count += 1

      else:
            cat_count += 1
print("Number of dog images =", dog_count)
print("Number of cat images =", cat_count)
```

```
Number of dog images = 1188
Number of cat images = 2704
```

```
import os
from PIL import Image

original_folder = "/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages/"
resized_folder = "/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages/Resized/"

# Create the Resized folder if it doesn't exist
if not os.path.exists(resized_folder):
    os.makedirs(resized_folder)

for i in range(4722):
    filename = os.listdir(original_folder)[i]
    img_path = os.path.join(original_folder, filename)

    try:
        img = Image.open(img_path)
        img = img.resize((224, 224))
        img = img.convert('RGB')

        new_img_path = os.path.join(resized_folder, filename)
        img.save(new_img_path)

    except Exception as e:
        print(f"Error processing {filename}: {e}")
```

```
Error processing Resized: [Errno 21] Is a directory: '/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages/Resized'
```

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
from google.colab.patches import cv2_imshow
img = mpimg.imread('/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages/Resized/cat (1).jpg')
plt.imshow(img)
plt.show()
```

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

```
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
from google.colab.patches import cv2_imshow

file_path = '/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages/Resized/dog (1700).jpg'
img = mpimg.imread(file_path)
plt.imshow(img)
plt.show()
```

Creating labels for resized images of dogs and cats Cat –> 0 Dog –> 1

```
# filenames = os.listdir("/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages/Resized")
# labels = []
# print(len(filenames))
# for i in range (3252):
#   file_name = filenames[i]
#   label = file_name[0:3]
#   if label == "cat":
#     labels.append(0)
#   else:
#     labels.append(1)
# print(filenames[0:3905])



#counting the images of dog and cat out of 3252 images
values, counts = np.unique(labels, return_counts =True)
print(values)
print(counts)
```

converting all resized images into numpy arrays

```
import cv2
import glob


# image_directory = '/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages/Resized/'
# image_extension = ['png', 'jpg']

# files = []

# [files.extend(glob.glob(image_directory + '*.' + e)) for e in image_extension]
# dog_cat_images = np.asarray([cv2.imread(file) for file in files])


print(dog_cat_images)


type(dog_cat_images)


print(dog_cat_images.shape)


X=dog_cat_images
y= np.asarray(labels)
```

Train Test Split

```
import os
import cv2
import glob
import numpy as np
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Step 1: Collect images and labels
image_directory = "/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages/Resized/"
image_extension = ["png", "jpg"]

files = []
[files.extend(glob.glob(image_directory + "*." + e)) for e in image_extension]

print(f"Total images found: {len(files)}")
```

```
dog_cat_images = []
labels = []

# Step 2: Filter valid images and assign labels
for file in files:
    img = cv2.imread(file)
    if img is not None:  # Ensure the image is valid
        resized_img = cv2.resize(img, (224, 224))  # Resize images to (224, 224)
        if "cat" in os.path.basename(file).lower():
            labels.append(0)  # Cat label
            dog_cat_images.append(resized_img)
        elif "dog" in os.path.basename(file).lower():
            labels.append(1)  # Dog label
            dog_cat_images.append(resized_img)

# Convert to NumPy arrays
X = np.array(dog_cat_images)
y = np.array(labels)

print(f"Original dataset size: {len(X)}")
print(f"Number of cat images: {np.sum(y == 0)}, Number of dog images: {np.sum(y == 1)}")

# Step 3: Flatten image arrays for SMOTE
X_flattened = X.reshape(len(X), -1)  # Flatten images to 2D array

# Step 4: Apply SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_flattened, y)

print(f"Resampled dataset size: {len(X_resampled)}")
print(f"Number of cat images after SMOTE: {np.sum(y_resampled == 0)}, Number of dog images after SMOTE: {np.sum(y_resampled == 1)}")

# Step 5: Reshape X back to image form
X_resampled_images = X_resampled.reshape(-1, 224, 224, 3)

# Step 6: Split the dataset
x_train, x_test, y_train, y_test = train_test_split(X_resampled_images, y_resampled, test_size=0.2, random_state=4)

# Normalize images
x_train_scaled = x_train / 255.0
x_test_scaled = x_test / 255.0

print(f"x_train shape: {x_train_scaled.shape}, y_train shape: {y_train.shape}")
print(f"x_test shape: {x_test_scaled.shape}, y_test shape: {y_test.shape}")



x=dog_cat_images
print(x.shape, x_train.shape, x_test.shape)
```

Training images –>2601 testing images –> 651

```
x_train_scaled=x_train/255
x_test_scaled=x_test/255
print(x_train_scaled)
```

```
[[0.00011146 0.         0.         ... 0.         0.         0.        ]
 [0.00996396 0.         0.         ... 0.         0.         0.        ]
 [0.00484116 0.02352941 0.         ... 0.         0.         0.        ]
 ...
 [0.01148178 0.         0.         ... 0.         0.         0.        ]
 [0.01030514 0.         0.         ... 0.         0.         0.        ]
 [0.         0.00739533 0.         ... 0.         0.         0.        ]]
```

```
import tensorflow as tf
import tensorflow_hub as hub

mobilenet_model = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4'

# Create the KerasLayer from the pretrained model
pretrained_model = hub.KerasLayer(mobilenet_model, input_shape=(224, 224, 3), trainable=False)
```

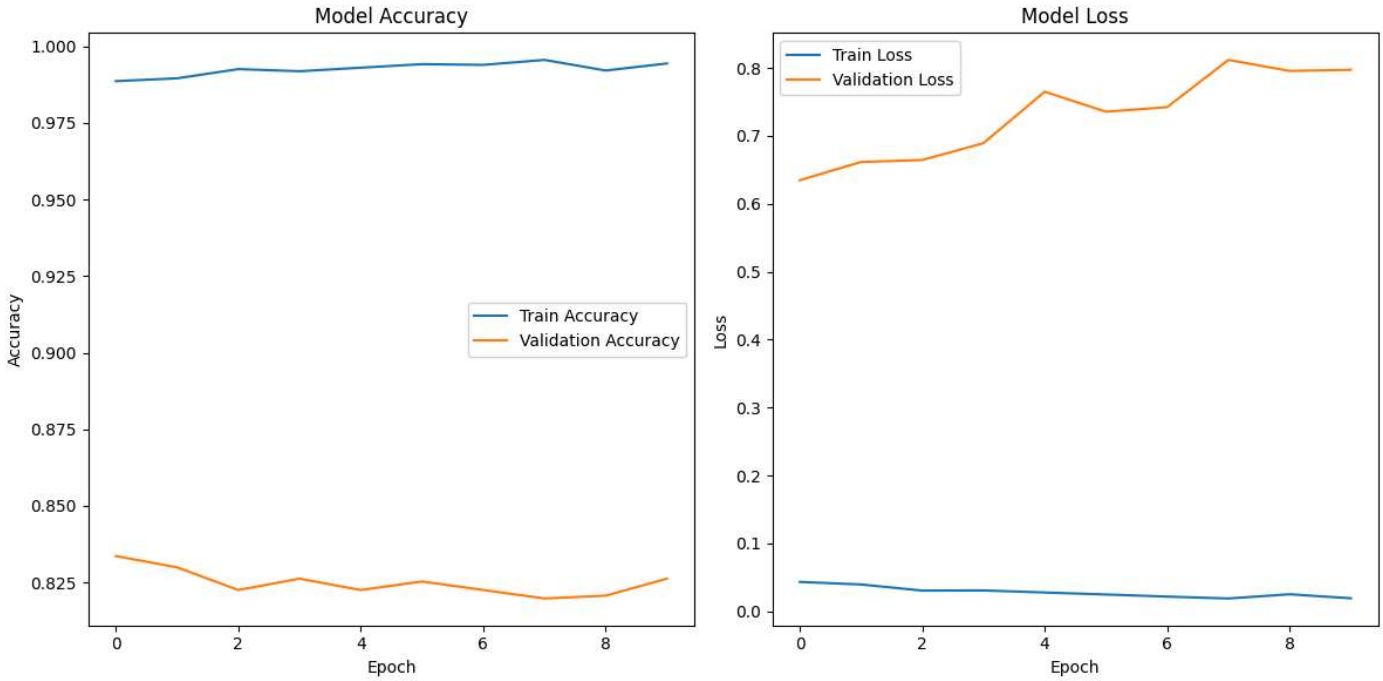
```
import tensorflow as tf
import tensorflow_hub as hub

# URL for the pretrained MobileNetV2 feature vector.
mobilenet_model = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4'

# Define a custom wrapper for the Hub layer.
class WrappedKerasLayer(tf.keras.layers.Layer):
    def __init__(self, hub_url, trainable=False, **kwargs):
        super(WrappedKerasLayer, self).__init__(**kwargs)
        # Create the hub layer inside the custom layer.
        self.hub_layer = hub.KerasLayer(hub_url, trainable=trainable)

    def call(self, inputs):
        # Forward inputs to the hub layer.
        return self.hub_layer(inputs)

# Build your Sequential model.
model = tf.keras.Sequential([
    # Use an Input layer to define the shape.
    tf.keras.layers.Input(shape=(224, 224, 3)),
    # Use your custom wrapped hub layer.
    WrappedKerasLayer(mobilenet_model, trainable=False),
    # Add a Dense layer for your classification task.
    tf.keras.layers.Dense(2)
])

model.summary()



import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
import cv2
import numpy as np
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model

# Reduce image size to 64x64 to save memory
IMAGE_SIZE = (64, 64)

# Load and process images
image_directory = "/content/drive/MyDrive/Cat vs Dog/kagglecatsanddogs_5340/PetImages/Resized/"
filenames = os.listdir(image_directory)
labels = []

dog_cat_images = []
valid_files = []

for file_name in filenames:
    label = file_name[:3]
    img_path = os.path.join(image_directory, file_name)
    img = cv2.imread(img_path)
    if img is not None:
        resized_img = cv2.resize(img, IMAGE_SIZE)
        dog_cat_images.append(resized_img)
        valid_files.append(file_name)
        if label == "cat":
            labels.append(0)
        elif label == "dog":
            labels.append(1)

# Convert to NumPy arrays
X = np.asarray(dog_cat_images) / 255.0
y = np.asarray(labels)

print(f"Image array shape: {X.shape}, Label array length: {len(y)}")

# Feature extraction using MobileNetV2 (pretrained)
base_model = MobileNetV2(weights="imagenet", include_top=False, input_shape=(64, 64, 3))
feature_extractor = Model(inputs=base_model.input, outputs=base_model.output)
X_features = feature_extractor.predict(X)

# Flatten features for SMOTE
X_flat = X_features.reshape((X_features.shape[0], -1))
```

```
# Apply SMOTE
smote = SMOTE(random_state=42)
X_flat_resampled, y_resampled = smote.fit_resample(X_flat, y)

# Split the balanced dataset
x_train, x_test, y_train, y_test = train_test_split(X_flat_resampled, y_resampled, test_size=0.2, random_state=4)

print(f"x_train shape: {x_train.shape}, y_train shape: {y_train.shape}")
print(f"x_test shape: {x_test.shape}, y_test shape: {y_test.shape}")

from tensorflow.keras import layers, models

# Define a simple neural network model
model = models.Sequential([
    layers.InputLayer(input_shape=(5120,)),  # Input shape matches the flattened feature size
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')  # Binary classification (cat vs dog)
])

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

# Train the model
history = model.fit(x_train, y_train, epochs=10, validation_split=0.2, batch_size=32)

# Evaluate the model
score, acc = model.evaluate(x_test, y_test)
print('Test score:', score)
print('Test accuracy:', acc)
```

```
Image array shape: (4722, 64, 64, 3), Label array length: 4722
<ipython-input-2-9d2cfb50068b>:41: UserWarning: `input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224].
  base_model = MobileNetV2(weights="imagenet", include_top=False, input_shape=(64, 64, 3))
136/148 ━━━━━━━━━━━━━━━━━━ 1s 123ms/step
```

```
import matplotlib.pyplot as plt

# Assuming you have the training history from model.fit()
history = model.fit(
    x_train_scaled, y_train,
    validation_data=(x_test_scaled, y_test),
    epochs=10
)

# Plot training & validation accuracy values
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Epoch 1/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 5s 38ms/step - accuracy: 0.9879 - loss: 0.0431 - val_accuracy: 0.8336 - val_loss: 0.6347
Epoch 2/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 6s 44ms/step - accuracy: 0.9900 - loss: 0.0395 - val_accuracy: 0.8299 - val_loss: 0.6616
Epoch 3/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 6s 41ms/step - accuracy: 0.9942 - loss: 0.0282 - val_accuracy: 0.8226 - val_loss: 0.6645
Epoch 4/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 11s 48ms/step - accuracy: 0.9926 - loss: 0.0293 - val_accuracy: 0.8262 - val_loss: 0.6892
Epoch 5/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 6s 40ms/step - accuracy: 0.9933 - loss: 0.0284 - val_accuracy: 0.8226 - val_loss: 0.7650
Epoch 6/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 6s 42ms/step - accuracy: 0.9950 - loss: 0.0245 - val_accuracy: 0.8253 - val_loss: 0.7355
Epoch 7/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 9s 37ms/step - accuracy: 0.9944 - loss: 0.0219 - val_accuracy: 0.8226 - val_loss: 0.7422
Epoch 8/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 7s 50ms/step - accuracy: 0.9958 - loss: 0.0187 - val_accuracy: 0.8198 - val_loss: 0.8118
Epoch 9/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 8s 37ms/step - accuracy: 0.9918 - loss: 0.0250 - val_accuracy: 0.8207 - val_loss: 0.7956
Epoch 10/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 6s 47ms/step - accuracy: 0.9946 - loss: 0.0171 - val_accuracy: 0.8262 - val_loss: 0.7971
```




```
import os
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.models import Model

# Assume that your trained model is already defined and loaded as "model"
# For this example, we also need a feature extractor that was used during training.
# We'll use MobileNetV2 (with include_top=False) to extract features.
# The model was trained on flattened features, so our feature extractor input size must match what was used.
# Here, we use an input size of (64, 64, 3) so that the flattened feature size becomes 5120.
feature_extractor = MobileNetV2(weights="imagenet", include_top=False, input_shape=(64, 64, 3))
feature_extractor.trainable = False  # Freeze the extractor

input_image_path = input("Path of the image to be predicted: ")

if not os.path.exists(input_image_path):
    print("Error: The image path is invalid or the file does not exist.")
else:
    # Read the image
    input_image = cv2.imread(input_image_path)

    if input_image is None:
        print("Error: Failed to read the image. It may be an unsupported format.")
```

```
else:
    # Optionally display the original image
    cv2_imshow(input_image)

    # Resize the image to match the feature extractor's expected input (64x64)
    input_image_resized = cv2.resize(input_image, (64, 64))

    # Expand dimensions to create a batch of 1 (shape becomes (1, 64, 64, 3))
    input_image_resized = np.expand_dims(input_image_resized, axis=0)

    # Preprocess the image for MobileNetV2 (scaling, etc.)
    input_image_preprocessed = preprocess_input(input_image_resized)

    # Extract features using MobileNetV2
    features = feature_extractor.predict(input_image_preprocessed)

    # Flatten the extracted features to a 1D vector (the model expects shape (None, 5120))
    features_flat = features.reshape((features.shape[0], -1))

    # Now, use your trained model to make a prediction on the flattened features
    input_prediction = model.predict(features_flat)
    print("Raw prediction output:", input_prediction)

    # For binary classification, you might do:
    input_pred_label = int(input_prediction[0] > 0.5)
    print("Predicted label:", input_pred_label)

    if input_pred_label == 0:
        print("The image represents a Cat")
    else:
        print("The image represents a Dog")
```

```
<ipython-input-3-52ab45274d06>:14: UserWarning: `input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224].
  feature_extractor = MobileNetV2(weights="imagenet", include_top=False, input_shape=(64, 64, 3))
Path of the image to be predicted: /content/dog.jpeg
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
Raw prediction output: [[0.9746949]]
Predicted label: 1
The image represents a Dog
<ipython-input-3-52ab45274d06>:51: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error ir
  input_pred_label = int(input_prediction[0] > 0.5)
```