```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
```

```
Start coding or generate with AI.
```

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

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

```
import os

train_path = '/content/drive/My Drive/train'
test_path = '/content/drive/My Drive/test1'

print("Train Folder Contents:", os.listdir(train_path))
print("Test Folder Contents:", os.listdir(test_path))
```

```
Train Folder Contents: ['dog.9189.jpg', 'dog.9199.jpg', 'dog.9186.jpg', 'dog.9185.jpg', 'dog.9201.jpg', 'dog.9203.jpg', 'dog.9194.jp
Test Folder Contents: ['9099.jpg', '9097.jpg', '9106.jpg', '9117.jpg', '9101.jpg', '9105.jpg', '9113.jpg', '9100.jpg', '9119.jpg',
```

```
import os
import shutil
```

```
import os
import shutil

def remove_nested_folders(directory):
    """
    Removes all nested folders in the given directory and moves files to the root directory.

    Args:
        directory (str): Path to the root directory (e.g., train or test folder).
    """
    for root, subdirs, files in os.walk(directory):
        for file in files:
            file_path = os.path.join(root, file)
            # Move the file to the root directory
            if root != directory:  # If not already in the root
                shutil.move(file_path, os.path.join(directory, file))
                print(f"Moved: {file_path} to {directory}")

    # Remove all empty folders after moving the files
    for root, subdirs, _ in os.walk(directory, topdown=False):
        for subdir in subdirs:
            subdir_path = os.path.join(root, subdir)
            if not os.listdir(subdir_path):
                os.rmdir(subdir_path)
                print(f"Removed empty folder: {subdir_path}")

# Example usage for both train and test datasets
train_dir = '/content/drive/My Drive/train'
test_dir = '/content/drive/My Drive/test1'

print("Cleaning train dataset...")
remove_nested_folders(train_dir)

print("Cleaning test dataset...")
remove_nested_folders(test_dir)
```

```
Show hidden output
```

```
import os
import shutil

# Define the path for the train directory
train_path = '/content/drive/My Drive/train'
```

```
# Define paths for the Cat and Dog directories within the train directory
cat_dir = os.path.join(train_path, 'cat')  # Create /train/cat
dog_dir = os.path.join(train_path, 'dog')  # Create /train/dog

# Create class directories (cat and dog)
os.makedirs(cat_dir, exist_ok=True)
os.makedirs(dog_dir, exist_ok=True)

# Move images to respective folders based on their filenames
for filename in os.listdir(train_path):
    file_path = os.path.join(train_path, filename)
    # Check if it's a file, not a directory
    if os.path.isfile(file_path):
        print(f"Processing file: {filename}")
        if 'cat' in filename.lower():
            shutil.move(file_path, os.path.join(cat_dir, filename))
            print(f"Moved {filename} to {cat_dir}")
        elif 'dog' in filename.lower():
            shutil.move(file_path, os.path.join(dog_dir, filename))
            print(f"Moved {filename} to {dog_dir}")

# Print the results
print("\nCat directory contains:", os.listdir(cat_dir))
print("Dog directory contains:", os.listdir(dog_dir))
```

Show hidden output

```
# Load the MobileNetV2 model pre-trained on ImageNet
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(150, 150, 3))

# Freeze the base model's layers
base_model.trainable = False

# Build the model
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification
])
```

```
<ipython-input-13-c5556c8753cc>:2: UserWarning: `input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 22
  base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
```

```
# Use ImageDataGenerator for data loading and augmentation
train_datagen = ImageDataGenerator(
    rescale=1.0/255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2  # 20% of training data used for validation
)

# Train and validation generators
train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)
```

```
Found 0 images belonging to 0 classes.
Found 0 images belonging to 0 classes.
```

```
img_size = (150, 150)
batch_size = 128  # Optimized batch size for speed and memory efficiency

# Data Generators
train_datagen = ImageDataGenerator(
    rescale=1.0/255,  # Only rescale images
    validation_split=0.2  # Split training data into training and validation
)

train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='validation'
)
```

```
Found 20000 images belonging to 2 classes.
Found 5000 images belonging to 2 classes.
```

```
# Print class labels mapping
print(train_generator.class_indices)
```

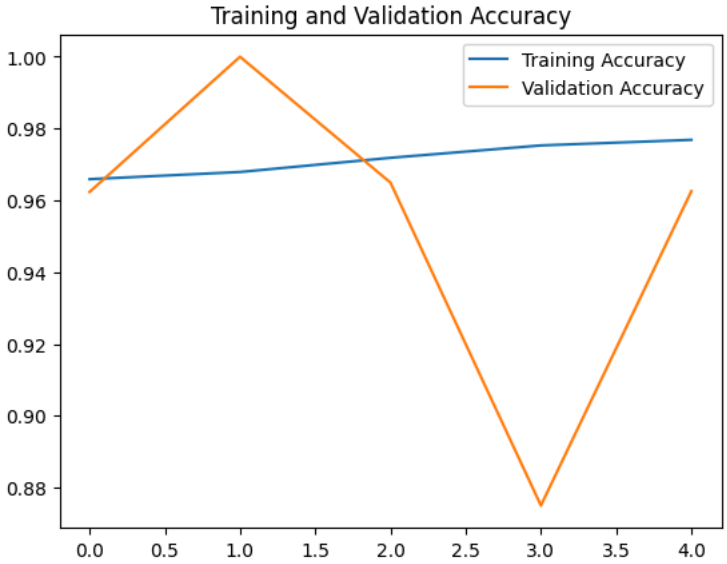
```
{'cat': 0, 'dog': 1}
```

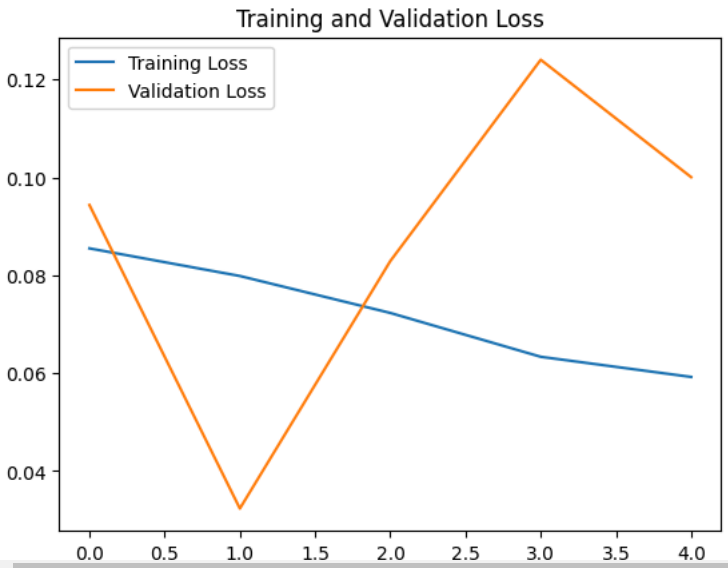
```
num_train_samples = sum(len(files) for _, _, files in os.walk(os.path.join(train_path)))
num_validation_samples = num_train_samples * 0.2


steps_per_epoch = num_train_samples // batch_size
validation_steps = int(num_validation_samples) // batch_size


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



history = model.fit(
    train_generator,
    validation_data=validation_generator,
    steps_per_epoch=steps_per_epoch,  # Use all training data per epoch
    validation_steps=validation_steps,
    epochs=5  # Adjust based on runtime needs
)
```

```
Epoch 1/5
157/195 ━━━━━━━━━━━━━━━━━━━━ 22s 585ms/step - accuracy: 0.9673 - loss: 0.0846/usr/lib/python3.10/contextlib.py:153: UserWarning: You
  self.gen.throw(typ, value, traceback)
195/195 ━━━━━━━━━━━━━━━━━━━━ 130s 614ms/step - accuracy: 0.9670 - loss: 0.0848 - val_accuracy: 0.9623 - val_loss: 0.0944
Epoch 2/5
195/195 ━━━━━━━━━━━━━━━━━━━━ 117s 578ms/step - accuracy: 0.9695 - loss: 0.0770 - val_accuracy: 1.0000 - val_loss: 0.0324
Epoch 3/5
195/195 ━━━━━━━━━━━━━━━━━━━━ 118s 585ms/step - accuracy: 0.9719 - loss: 0.0716 - val_accuracy: 0.9649 - val_loss: 0.0829
Epoch 4/5
195/195 ━━━━━━━━━━━━━━━━━━━━ 114s 435ms/step - accuracy: 0.9774 - loss: 0.0605 - val_accuracy: 0.8750 - val_loss: 0.1240
Epoch 5/5
195/195 ━━━━━━━━━━━━━━━━━━━━ 162s 544ms/step - accuracy: 0.9785 - loss: 0.0573 - val_accuracy: 0.9625 - val_loss: 0.1000
```

```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
```

```
plt.title('Training and Validation Loss')
plt.show()
```






```
import os
from IPython.display import display
import ipywidgets as widgets
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

# Define the test directory and load the trained model
test_dir = '/content/drive/My Drive/test1'
model = model  # Use the model you trained earlier

# Get all image filenames in the test directory
image_files = [f for f in os.listdir(test_dir) if f.lower().endswith(('.png', '.jpg', '.jpeg'))]

# Dropdown widget for selecting an image
dropdown = widgets.Dropdown(
    options=image_files,
    description='Select Image:',
    style={'description_width': 'initial'}
)

# Function to make predictions and display the selected image
def display_prediction(change):
    selected_image = change.new
    image_path = os.path.join(test_dir, selected_image)

    # Load and preprocess the image
    img = Image.open(image_path).resize((150, 150))  # Resize to match model input size
    img_array = np.array(img) / 255.0  # Normalize pixel values
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
```

```
        # Make a prediction
        prediction = model.predict(img_array)
        predicted_class = 'Dog' if prediction > 0.5 else 'Cat'

        # Display the image and prediction
        plt.imshow(img)
        plt.title(f"Prediction: {predicted_class}")
        plt.axis('off')
        plt.show()

    # Attach the function to the dropdown
    dropdown.observe(display_prediction, names='value')

    # Display the dropdown
    print("Select an image to verify the prediction:")
    display(dropdown)
```

```
Select an image to verify the prediction:
```

Select Image: 10258.jpg

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

Prediction: Cat


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

Prediction: Cat

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

Prediction: Cat

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
```

Prediction: Dog