Digital Empowerment Network

Artificial Intelligence

Week 03

Image Classification using pre-trained CNNs and Deployment with Streamlet

Name : Abdul Salam

ID: STU-ML-251-140

Mentor: Hussain Shoaib

Image Classification using pre-trained CNNs and Deployment with Streamlit

Objective:

To implement an image classification model using pre-trained CNNs (transfer learning) and deploy it using Streamlit as a simple web application that accepts image uploads and shows predictions.

Step 1: Dataset Selection

- ➢ Choose one of the following image datasets:
- ➢ Cats' vs Dogs
- ➢ Flowers (5 Classes)
- ➢ Intel Image Classification
- ➢ Any dataset of your choice (2–10 classes)

Step 2: Model Implementation (Transfer Learning)

A. Data Preparation

- ➢ Organize dataset into folders (train/, val/, test/)
- ➢ Resize all images (e.g., 224×224)
- ➢ Normalize and apply data augmentation
- ➢ Split data appropriately

B. Pre-trained CNN Model

Use at least two of the following: VGG16, MobileNetV2, ResNet50, or EfficientNetB0

Steps:
- Load pre-trained model (without top layers)
- Add your own classification head
- Freeze base layers for initial training
- (Optional) fine-tune later layers

C. Evaluation

- Evaluate with Accuracy, Loss, Confusion Matrix
- Plot training/validation accuracy/loss
- Visualize a few predictions with probabilities

Step 3: Web App Deployment Using Streamlit

A. Streamlit App Features

- Upload image (.jpg/.png)
- Model processes the image and shows:
- Predicted class with confidence
- Optionally: top 3 probabilities
- Show visual feedback (image + prediction)

B. App Structure

- app.py (Streamlit script)
- model.h5 or saved .pt model
- requirements.txt for dependencies

Deliverables:

- Trained model file (e.g., .h5 or .pt)
- app.py Streamlit file
- Jupyter Notebook (for training phase)
- Sample predictions (screenshots or display)

Deadline:

Submit within 7 days of receiving the task.

Tools to Use:

- Python, TensorFlow/Keras or PyTorch
- Streamlit
- Google Colab or Jupyter
- Libraries: opencv-python, PIL, matplotlib, seaborn

1. Step 01: Dataset Selection

======================= Cats' vs Dogs ==============================

Code:

```
# ===== Data Download and Extraction =====
import os
import zipfile

# Create .kaggle directory and set up credentials
os.makedirs('/root/.kaggle', exist_ok=True)
os.system('cp kaggle.json /root/.kaggle/')
os.system('chmod 600 /root/.kaggle/kaggle.json')

# Download dataset
os.system('kaggle datasets download -d salader/dogs-vs-cats')
```

```python
# Extract dataset
with zipfile.ZipFile('dogs-vs-cats.zip', 'r') as zip_ref:
    zip_ref.extractall('/content')

# Set data paths
train_dir = '/content/train'
validation_dir = '/content/test'

# Verify directories
assert os.path.exists(train_dir), f"Training directory not found at 
{train_dir}"
assert os.path.exists(validation_dir), f"Validation directory not found 
at {validation_dir}"
```

Step 2: Model Implementation (Transfer Learning)

```
 ==================== Model Training Setup ====================
```

Code:

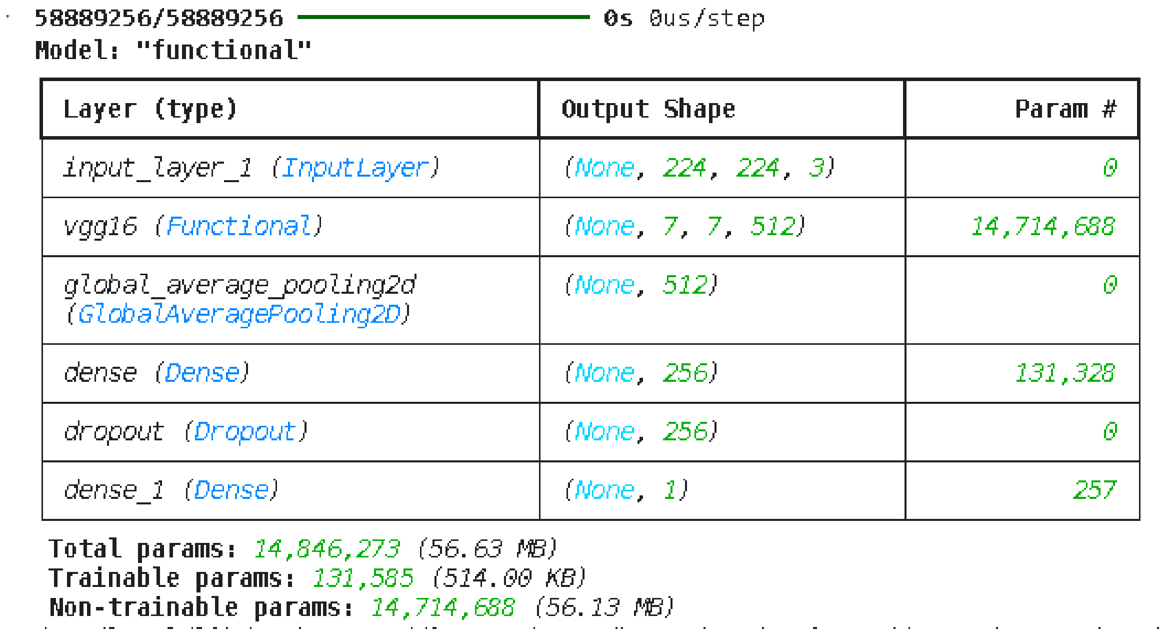
```python
# ===== Model Training Setup =====
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.applications import VGG16, MobileNetV2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import numpy as np
import cv2

# Data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./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,
    fill_mode='nearest'
)

validation_datagen = ImageDataGenerator(rescale=1./255)
```

```
# Create data generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)

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

=====================: VGG16 ====== = = = = = = = = = = = = = = = = = = =

Code:

```
# ===== VGG16 Model =====
def create_vgg16_model():
    base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
    base_model.trainable = False

    inputs = keras.Input(shape=(224, 224, 3))
    x = base_model(inputs, training=False)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(256, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(1, activation='sigmoid')(x)

    model = keras.Model(inputs, outputs)
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

vgg_model = create_vgg16_model()
vgg_model.summary()

history_vgg = vgg_model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)
```



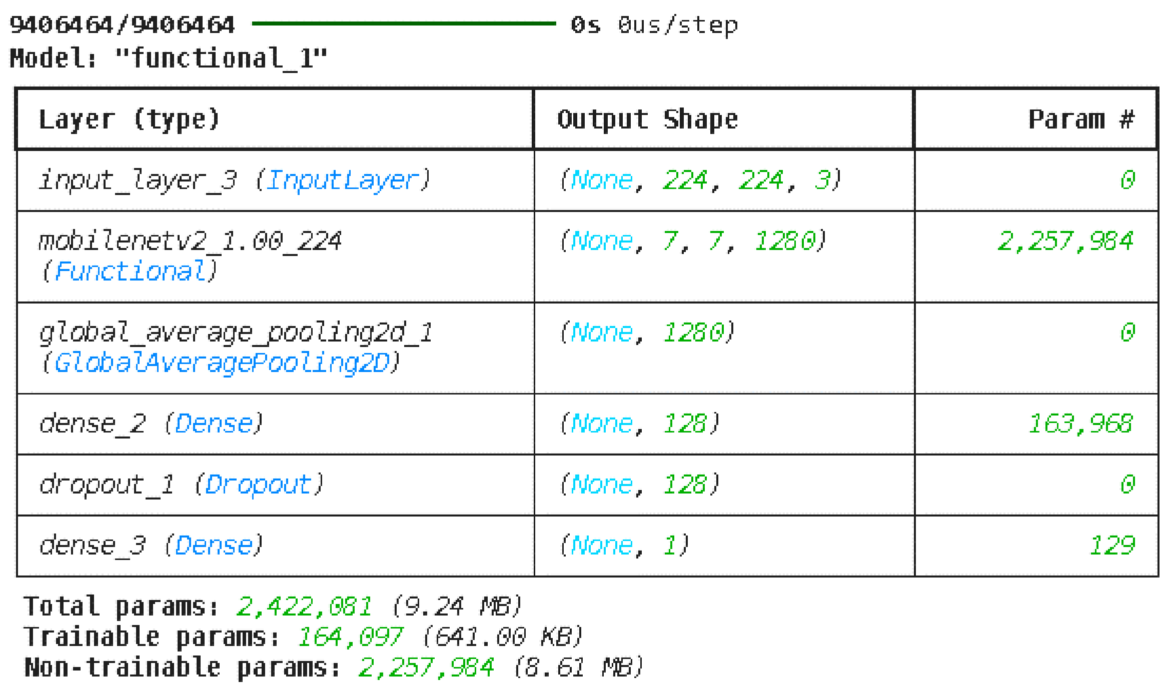
```
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Model: "functional"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer_1 (InputLayer)       │ (None, 224, 224, 3)    │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ vgg16 (Functional)               │ (None, 7, 7, 512)      │    14,714,688 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ global_average_pooling2d         │ (None, 512)            │             0 │
│ (GlobalAveragePooling2D)         │                        │               │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                    │ (None, 256)            │       131,328 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)                │ (None, 256)            │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                  │ (None, 1)              │           257 │
└──────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 14,846,273 (56.63 MB)
 Trainable params: 131,585 (514.00 KB)
 Non-trainable params: 14,714,688 (56.13 MB)
```

==================: **MobileNetV2 Model** ====== = = = = = = = = = =

Code:

```python
# ===== MobileNetV2 Model =====
def create_mobilenet_model():
    base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
    base_model.trainable = False

    inputs = keras.Input(shape=(224, 224, 3))
    x = base_model(inputs, training=False)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.3)(x)
    outputs = layers.Dense(1, activation='sigmoid')(x)

    model = keras.Model(inputs, outputs)
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

mobilenet_model = create_mobilenet_model()
mobilenet_model.summary()
```

```
history_mobilenet = mobilenet_model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)
```



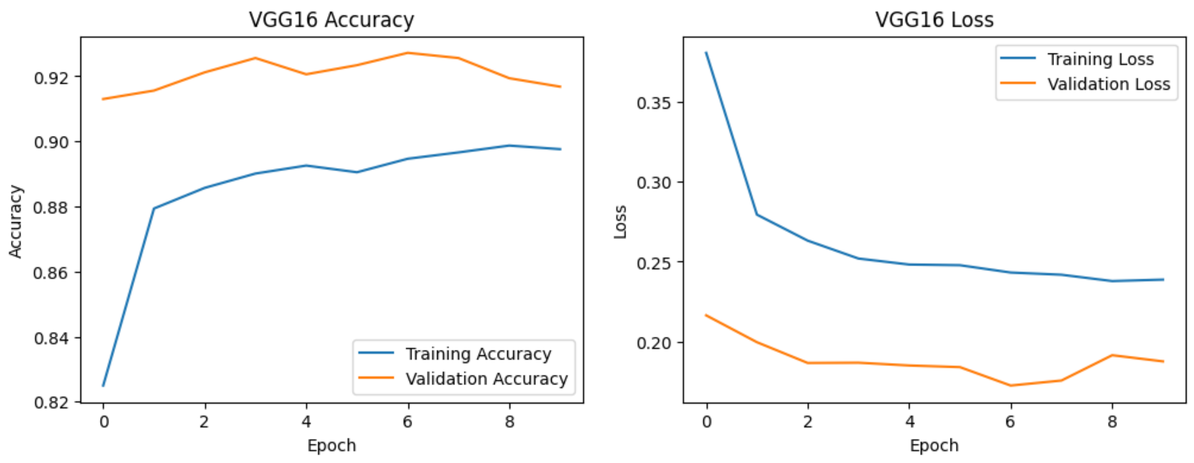
```
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Model: "functional_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer_3 (InputLayer)      │ (None, 224, 224, 3)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ mobilenetv2_1.00_224            │ (None, 7, 7, 1280)     │     2,257,984 │
│ (Functional)                    │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ global_average_pooling2d_1      │ (None, 1280)           │             0 │
│ (GlobalAveragePooling2D)        │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 128)            │       163,968 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_1 (Dropout)             │ (None, 128)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 1)              │           129 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 2,422,081 (9.24 MB)
 Trainable params: 164,097 (641.00 KB)
 Non-trainable params: 2,257,984 (8.61 MB)
```

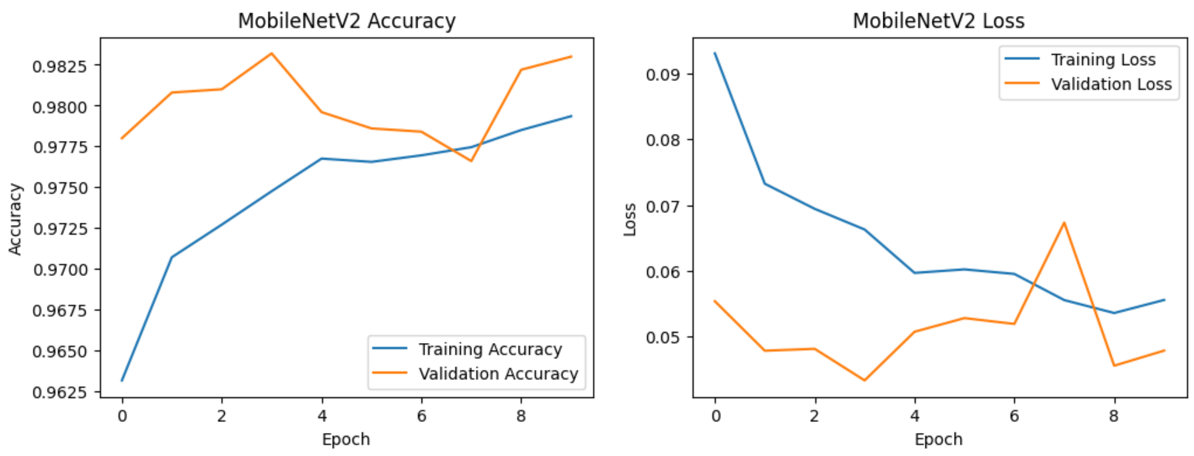
Step 03: Visualization:

Code:

```
# ===== Visualization =====
def plot_history(history, model_name):
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
    plt.title(f'{model_name} Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'{model_name} Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

plot_history(history_vgg, 'VGG16')
plot_history(history_mobilenet, 'MobileNetV2')

# Save the best model
mobilenet_model.save('cats_dogs_mobilenet.h5')
```

Output:





Step 04: Prediction Examples:

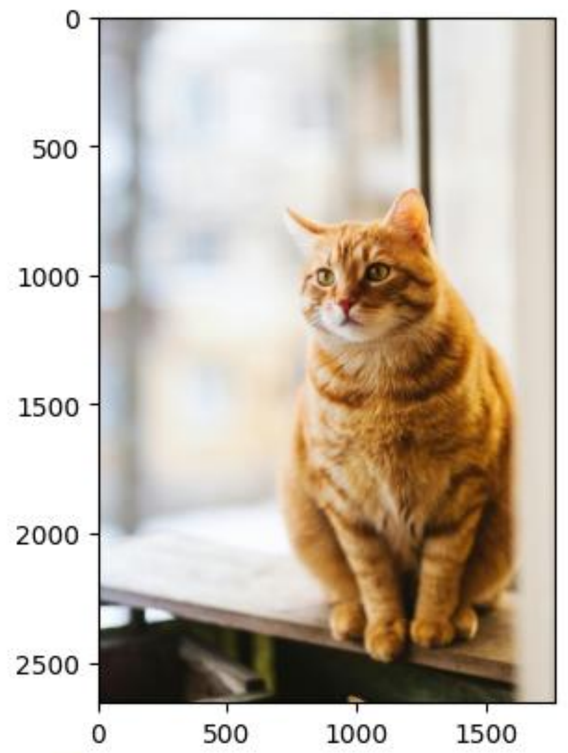
```
========== Prediction Example of Mobilenet Model =============
```

Code:

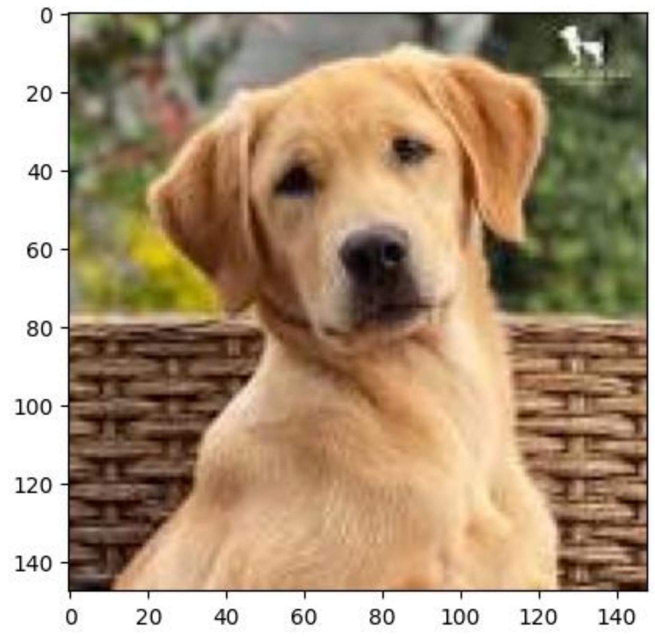
```
# ===== Prediction Example =====
def predict_image(model, image_path):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(img)
    plt.show()

    img = cv2.resize(img, (224, 224))
    img = img.reshape((1, 224, 224, 3)) / 255.0

    prediction = model.predict(img)
    class_name = 'Dog' if prediction > 0.5 else 'Cat'
    confidence = float(prediction[0][0]) if prediction > 0.5 else 1 - 
float(prediction[0][0])

    print(f"Prediction: {class_name} with confidence: 
{confidence:.2%}")
    return prediction

# Test with sample images
predict_image(mobilenet_model, '/content/Cat_01.jpg')
predict_image(mobilenet_model, '/content/Dog_01.jpg')
```

Output:

The Image is Input and the Class is Output.



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
Prediction: Cat with confidence: 100.00%
```



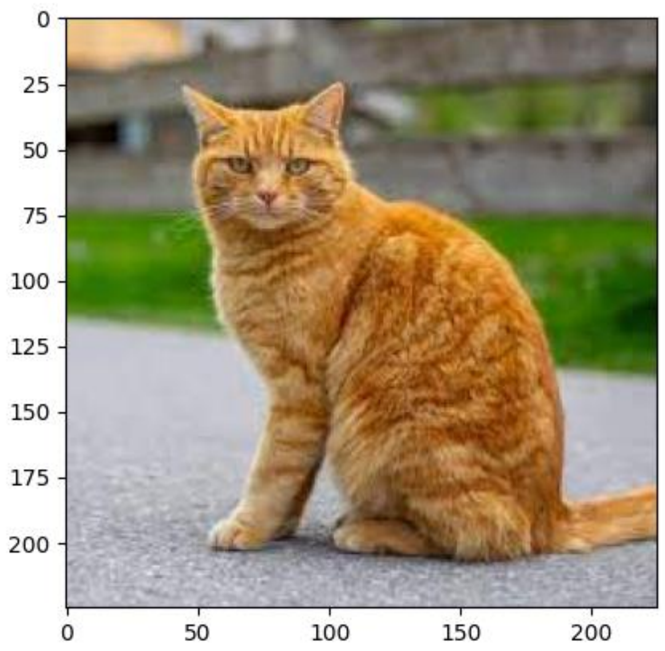
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
Prediction: Dog with confidence: 100.00%
array([[1.]], dtype=float32)
```

========= Prediction Example of vgg16 Model ==============

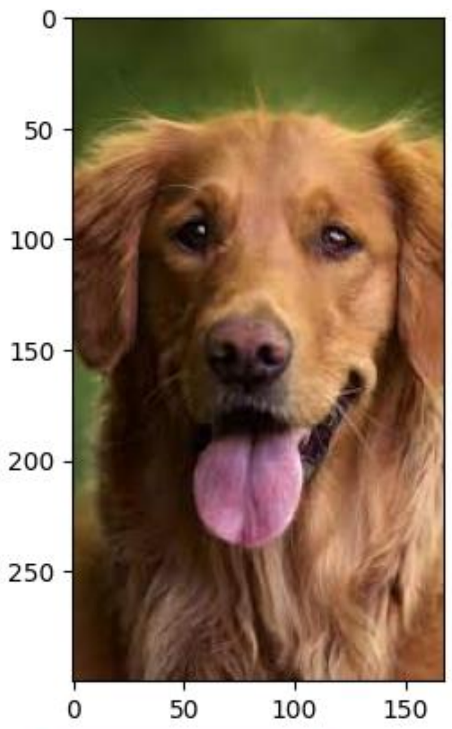
Code:

```
# ===== Prediction Example =====
def predict_image01(model, image_path):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(img)
    plt.show()

    img = cv2.resize(img, (224, 224))
    img = img.reshape((1, 224, 224, 3)) / 255.0

    prediction = model.predict(img)
    class_name = 'Dog' if prediction > 0.5 else 'Cat'
    confidence = float(prediction[0][0]) if prediction > 0.5 else 1 -
float(prediction[0][0])

    print(f"Prediction: {class_name} with confidence:
{confidence:.2%}")
    return prediction

# Test with sample images
predict_image01(vgg_model, '/content/Cat_03.jpg')
predict_image01(vgg_model, '/content/Dog_03.jpg')
```

Output:



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
Prediction: Cat with confidence: 99.91%
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
Prediction: Dog with confidence: 100.00%
array([[0.9999995]], dtype=float32)
```

Explanation Step By Step:

1. Data Download and Extraction

This section handles dataset acquisition and preparation:

- Creates a Kaggle credentials directory and sets permissions
- Downloads the Dogs vs Cats dataset from Kaggle using the API
- Extracts the ZIP file to /content directory
- Sets up paths for training and validation data
- Verifies the directories exist using assertions

Key points:

- Uses Kaggle API for dataset download (requires kaggle.json credentials)
- Organizes data into train/test directories
- Includes error checking with assertions

2. Model Training Setup

This section prepares the data pipeline:

- Imports necessary TensorFlow/Keras modules
- Sets up data augmentation for training:
  - Rotation, shifting, zooming, flipping
  - Normalization (rescaling to 0-1 range)
- Creates image generators that:
  - Flow images from directories
  - Resize to 224x224 (standard for VGG/MobileNet)
  - Use binary classification mode (cats vs dogs)
  - Batch size of 32 images

Key points:

- Data augmentation helps prevent overfitting
- Separate generators for training (with augmentation) and validation (just normalization)
- Automatic label inference from directory structure

3. VGG16 Model

This implements transfer learning with VGG16:

- Loads pre-trained VGG16 (ImageNet weights) without top layers
- Freezes base model weights to prevent updating during training
- Adds new classification head:
  - Global average pooling
  - Dense layer (256 units, ReLU)
  - Dropout for regularization (0.5)
  - Final sigmoid output (binary classification)
- Compiles with Adam optimizer and binary crossentropy loss

Key points:

- Transfer learning leverages pre-trained features
- Freezing base preserves learned patterns
- Custom head adapts to new task (cats vs dogs)

4. MobileNetV2 Model

Similar structure to VGG16 but:

- Uses MobileNetV2 as base (more efficient architecture)
- Smaller dense layer (128 units)
- Less dropout (0.3)
- Otherwise same training setup

Key points:

- MobileNet is lighter/faster than VGG
- Similar transfer learning approach

5. Training and Visualization

- Both models trained for 10 epochs
- Training history tracked (accuracy/loss)
- Visualization function plots:
    - Training vs validation accuracy
    - Training vs validation loss
- MobileNet model saved to HDF5 file

Key points:

- Standard training procedure
- Visualizations help evaluate training progress
- Model saving preserves trained weights

6. Prediction Functions

Two similar prediction functions:

1. `predict_image()` for MobileNet
2. `predict_image01()` for VGG16

Both:

- Load and display input image
- Preprocess (resize, normalize) to match training
- Make prediction
- Output class (Cat/Dog) and confidence
- Return raw prediction value

Key differences:

- `predict_image()` uses MobileNet model
- `predict_image01()` uses VGG16 model
- Otherwise identical functionality

Key Observations:

1. Complete pipeline from data to prediction
2. Two model architectures compared
3. Good practices:
    - Data augmentation
    - Transfer learning
    - Model evaluation
    - Visualization
4. Prediction functions could be consolidated into one flexible function
5. Both models use same input size (224x224) despite different original designs