Differentiate between cat and dog Dataset from : 'https://www.microsoft.com/en-us/download/details.aspx?id=54765'

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
import os
import zipfile
import random
from shutil import copyfile


# Extract the zip file
zip_file_path = '/content/PetImages.zip'
extracted_folder = '/content/extracted_data'
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extracted_folder)

# Data Preprocessing and Augmentation
data_dir = '/content/extracted_data/PetImages'
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    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')

# Correct class_mode to 'binary'
train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary')
```

```
 Found 102 images belonging to 2 classes.
```

```
# Create separate training and validation directories
train_dir = '/content/extracted_data/train'
val_dir = '/content/extracted_data/validation'

os.makedirs(train_dir, exist_ok=True)
os.makedirs(val_dir, exist_ok=True)

#classes
class_names = ['Cat', 'Dog']

# Split data into training and validation sets manually
for class_name in class_names:
    class_dir = os.path.join(data_dir, class_name)
    train_class_dir = os.path.join(train_dir, class_name)
    val_class_dir = os.path.join(val_dir, class_name)

    os.makedirs(train_class_dir, exist_ok=True)
    os.makedirs(val_class_dir, exist_ok=True)

    image_files = os.listdir(class_dir)
    random.shuffle(image_files)
    num_train = int(0.8 * len(image_files))  # 80% for training
    train_images = image_files[:num_train]
    val_images = image_files[num_train:]

    for image in train_images:
        src = os.path.join(class_dir, image)
        dst = os.path.join(train_class_dir, image)
        copyfile(src, dst)

    for image in val_images:
```

```
            src = os.path.join(class_dir, image)
            dst = os.path.join(val_class_dir, image)
            copyfile(src, dst)

    # Create data generators for training and validation
    train_datagen = ImageDataGenerator(
        rescale=1.0/255.0,
        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')

    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(224, 224),
        batch_size=32,
        class_mode='binary')

    val_datagen = ImageDataGenerator(rescale=1.0/255.0)
    val_generator = val_datagen.flow_from_directory(
        val_dir,
        target_size=(224, 224),
        batch_size=32,
        class_mode='binary')
```

```
     Found 80 images belonging to 2 classes.
     Found 22 images belonging to 2 classes.
```

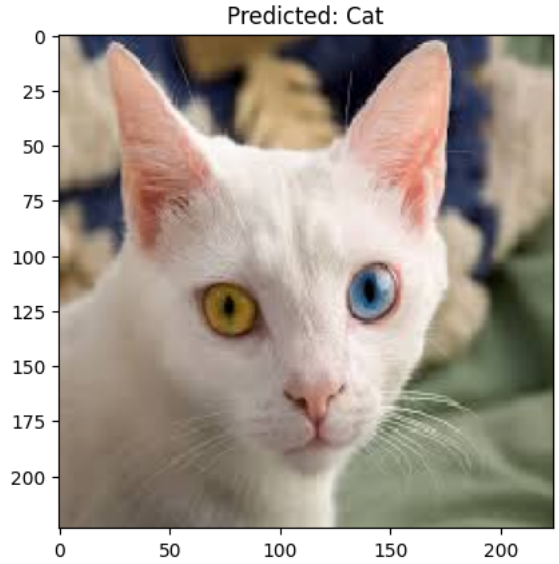
```
    # Model Creation and Fine-Tuning

    base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    for layer in base_model.layers:
        layer.trainable = False

    x = Flatten()(base_model.output)
    x = Dense(256, activation='relu')(x)
    output = Dense(1, activation='sigmoid')(x)  # Use 'sigmoid' activation for binary classification

    model = Model(inputs=base_model.input, outputs=output)

    # Compile the Model
    model.compile(optimizer=Adam(lr=0.001), loss='binary_crossentropy', metrics=['accuracy'])

    # Model Training
    history = model.fit(train_generator, epochs=10, validation_data=val_generator)

    # Model Evaluation
    val_generator.reset()  # Reset the generator before evaluation
    val_loss, val_acc = model.evaluate(val_generator)

    # Generate Classification Report
    val_generator.reset()
    predictions = model.predict(val_generator)
    predicted_classes = np.round(predictions)
    true_classes = val_generator.classes
    class_labels = ['Cat', 'Dog']  # Adjusting class labels here

    report = classification_report(true_classes, predicted_classes, target_names=class_labels)
    print(report)
```

```
     Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop
     58889256/58889256 [==============================] - 0s 0us/step
     WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.le
     Epoch 1/10
     3/3 [==============================] - 42s 14s/step - loss: 2.2886 - accuracy: 0.5250 - val_loss: 2.0491 - val_accuracy: 0.5000
     Epoch 2/10
     3/3 [==============================] - 40s 17s/step - loss: 2.2335 - accuracy: 0.5125 - val_loss: 0.8106 - val_accuracy: 0.6818
     Epoch 3/10
     3/3 [==============================] - 39s 13s/step - loss: 1.1475 - accuracy: 0.6750 - val_loss: 1.2158 - val_accuracy: 0.6364
     Epoch 4/10
     3/3 [==============================] - 39s 16s/step - loss: 0.9702 - accuracy: 0.6750 - val_loss: 1.0641 - val_accuracy: 0.5455
     Epoch 5/10
     3/3 [==============================] - 40s 13s/step - loss: 0.4963 - accuracy: 0.8250 - val_loss: 1.3634 - val_accuracy: 0.6364
     Epoch 6/10
     3/3 [==============================] - 39s 16s/step - loss: 0.5178 - accuracy: 0.7875 - val_loss: 0.5557 - val_accuracy: 0.7273
```

```
Epoch 7/10
3/3 [==============================] - 39s 16s/step - loss: 0.3003 - accuracy: 0.8375 - val_loss: 0.5633 - val_accuracy: 0.6818
Epoch 8/10
3/3 [==============================] - 40s 14s/step - loss: 0.3036 - accuracy: 0.8875 - val_loss: 0.8608 - val_accuracy: 0.7727
Epoch 9/10
3/3 [==============================] - 39s 13s/step - loss: 0.2872 - accuracy: 0.8875 - val_loss: 0.4161 - val_accuracy: 0.7727
Epoch 10/10
3/3 [==============================] - 39s 13s/step - loss: 0.2771 - accuracy: 0.9125 - val_loss: 0.4019 - val_accuracy: 0.8182
1/1 [==============================] - 9s 9s/step - loss: 0.4019 - accuracy: 0.8182
1/1 [==============================] - 9s 9s/step
              precision    recall  f1-score   support

         Cat       0.55      0.55      0.55        11
         Dog       0.55      0.55      0.55        11

    accuracy                           0.55        22
   macro avg       0.55      0.55      0.55        22
weighted avg       0.55      0.55      0.55        22
```

```
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
# Step 6: Deployment and Testing (simple user interface)
def preprocess_image(image_path):
    img = image.load_img(image_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0
    return img_array

def predict_image(image_array):
    prediction = model.predict(image_array)
    class_index = np.argmax(prediction)
    class_labels = list(train_generator.class_indices.keys())
    predicted_label = class_labels[class_index]
    confidence = prediction[0][class_index]  # Confidence score for the predicted class
    return predicted_label, confidence

def display_prediction(image_path):
    preprocessed_image = preprocess_image(image_path)
    predicted_label, confidence = predict_image(preprocessed_image)

    plt.imshow(preprocessed_image[0])
    plt.title(f'Predicted: {predicted_label}')
    plt.show()

# Provide the path to your test image
test_image_path = '/content/cat2.jfif'
display_prediction(test_image_path)
```

```
1/1 [==============================] - 1s 528ms/step
```



✓ 1s completed at 7:05 PM