```
!ls /content
```

```
archive1.zip  archive.zip  sample_data
```

```
import zipfile

for file in ["archive.zip","archive1.zip"]:
    try:
        with zipfile.ZipFile("/content/"+file, 'r') as zip_ref:
            print(file, "✅ is a valid zip file")
    except zipfile.BadZipFile:
        print(file, "❌ is NOT a valid zip file")
```

```
archive.zip ✅ is a valid zip file
archive1.zip ✅ is a valid zip file
```

```
# Create folder to store raw images
!mkdir -p /content/raw

# Extract Lion & Tiger dataset
!unzip -q /content/archive.zip -d /content/raw/

# Extract Cat & Dog dataset
!unzip -q /content/archive1.zip -d /content/raw/
```

```
!ls /content/raw
!ls /content/raw/*
```

```
PetImages  test  train
/content/raw/PetImages:
Cat  Dog

/content/raw/test:
lion  tiger

/content/raw/train:
lion  tiger
```

```
from pathlib import Path
from glob import glob

RAW_ROOT = Path("/content/raw")

# Correct paths based on your folder structure
lion_src = RAW_ROOT / "train" / "lion"
tiger_src = RAW_ROOT / "train" / "tiger"
cat_src = RAW_ROOT / "PetImages" / "Cat"
dog_src = RAW_ROOT / "PetImages" / "Dog"

print("Lion images:", len(glob(str(lion_src/"*"))))
print("Tiger images:", len(glob(str(tiger_src/"*"))))
print("Cat images:", len(glob(str(cat_src/"*"))))
print("Dog images:", len(glob(str(dog_src/"*"))))
```

```
Lion images: 180
Tiger images: 180
Cat images: 12499
Dog images: 12499
```

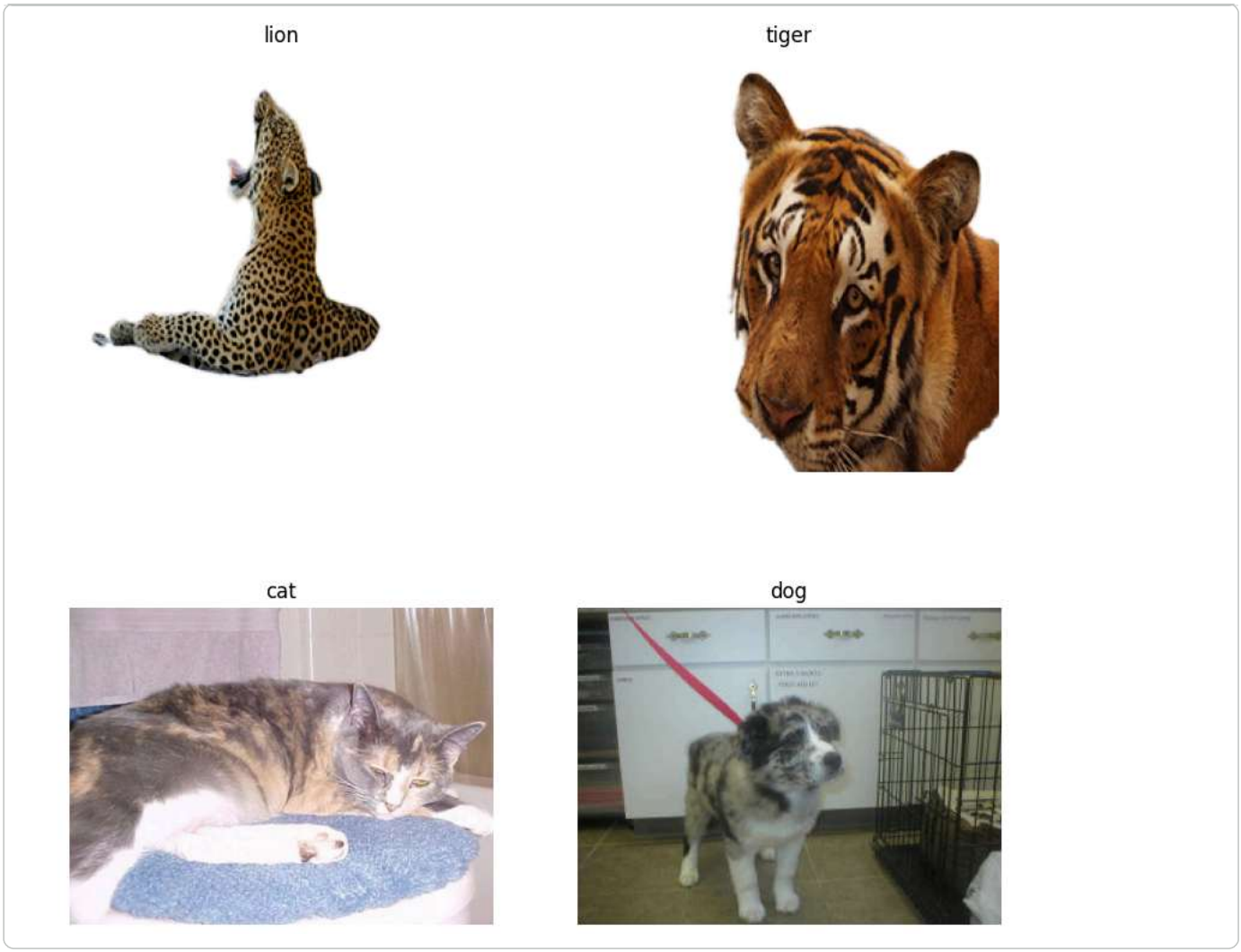
```
import random

target_per_class = 180  # match Lion/Tiger
print("Target images per class:", target_per_class)

def downsample_folder(src_path, target_count):
    files = glob(str(src_path/"*"))
    random.shuffle(files)
    return files[:target_count]
```

```
cat_files = downsample_folder(cat_src, target_per_class)
dog_files = downsample_folder(dog_src, target_per_class)

print("Downsampled Cat:", len(cat_files))
print("Downsampled Dog:", len(dog_files))
```

```
Target images per class: 180
Downsampled Cat: 180
Downsampled Dog: 180
```

```
import shutil
from sklearn.model_selection import train_test_split

DEST_ROOT = Path("/content/dataset_split")

# Create folders
for split in ["train","val","test"]:
    for cls in ["lion","tiger","cat","dog"]:
        (DEST_ROOT / split / cls).mkdir(parents=True, exist_ok=True)

def split_and_copy(files, class_name):
    train_and_val, test = train_test_split(files, test_size=0.15, random_state=42)
    train, val = train_test_split(train_and_val, test_size=0.1765, random_state=42)
    for f in train:
        shutil.copy(f, DEST_ROOT/"train"/class_name)
    for f in val:
        shutil.copy(f, DEST_ROOT/"val"/class_name)
    for f in test:
        shutil.copy(f, DEST_ROOT/"test"/class_name)

# Apply split to all classes
split_and_copy(glob(str(lion_src/"*")), "lion")
split_and_copy(glob(str(tiger_src/"*")), "tiger")
split_and_copy(cat_files, "cat")
split_and_copy(dog_files, "dog")
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

fig, axes = plt.subplots(2,2, figsize=(10,10))
classes = ["lion","tiger","cat","dog"]

for ax, cls in zip(axes.flatten(), classes):
    img_path = glob(str(DEST_ROOT/"train"/cls/"*"))[0]
    img = mpimg.imread(img_path)
    ax.imshow(img)
    ax.set_title(cls)
    ax.axis("off")
plt.show()
```




```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMG_SIZE = (224,224)
BATCH_SIZE = 16

train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    rotation_range=20,
    zoom_range=0.15
)

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_gen = train_datagen.flow_from_directory(
    DEST_ROOT/"train",
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

val_gen = val_datagen.flow_from_directory(
    DEST_ROOT/"val",
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

test_gen = test_datagen.flow_from_directory(
    DEST_ROOT/"test",
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
```

```
    shuffle=False
)
```

```
Found 500 images belonging to 4 classes.
Found 112 images belonging to 4 classes.
Found 108 images belonging to 4 classes.
```

```
from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam

# Load VGG19 without top layers
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(224,224,3))

# Freeze all layers first
for layer in base_model.layers:
    layer.trainable = False

# Unfreeze last few layers
for layer in base_model.layers[-5:]:
    layer.trainable = True

# Add custom classification head
x = base_model.output
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(4, activation='softmax')(x)  # 4 classes

model = Model(inputs=base_model.input, outputs=output)
model.compile(optimizer=Adam(learning_rate=1e-4), loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kern
80134624/80134624 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
Model: "functional"
```

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
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_conv4 (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_conv4 (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_conv4 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 512)	12,845,568
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 4)	2,052

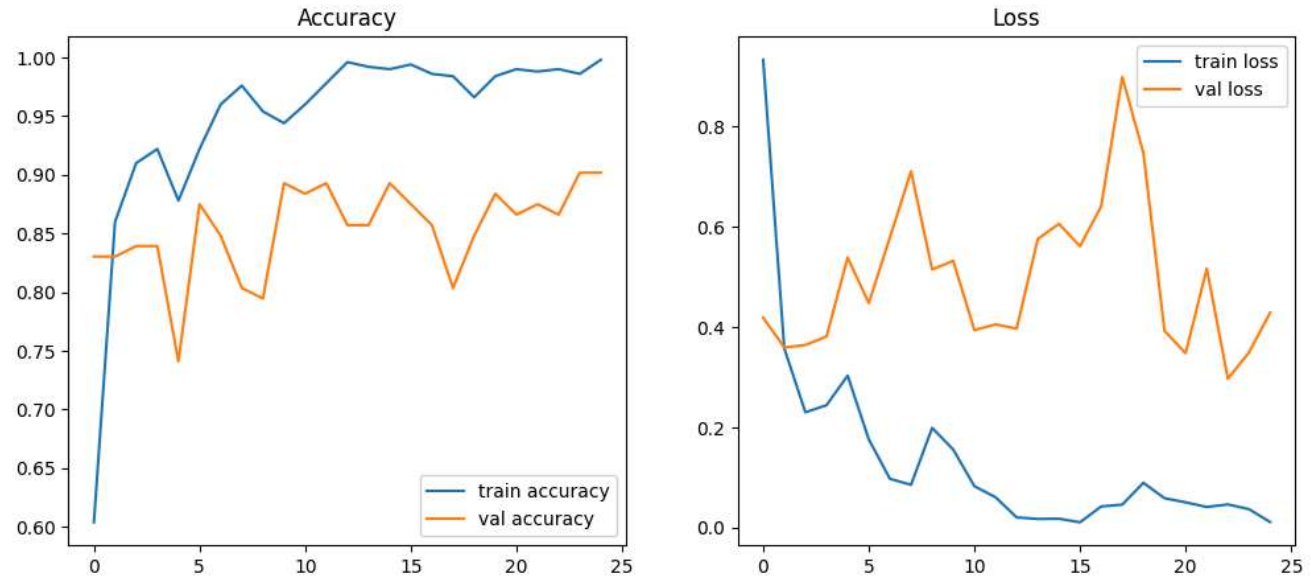
```
Total params: 32,872,004 (125.40 MB)
Trainable params: 22,286,852 (85.02 MB)
Non-trainable params: 10,585,152 (40.38 MB)
```

```
history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=25
)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset
  self._warn_if_super_not_called()
Epoch 1/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 28s 467ms/step - accuracy: 0.5181 - loss: 1.2037 - val_accuracy: 0.8304 - val_loss: 0.4189
Epoch 2/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 272ms/step - accuracy: 0.8286 - loss: 0.4060 - val_accuracy: 0.8304 - val_loss: 0.3593
Epoch 3/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 292ms/step - accuracy: 0.8944 - loss: 0.2271 - val_accuracy: 0.8393 - val_loss: 0.3643
Epoch 4/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 288ms/step - accuracy: 0.9495 - loss: 0.1491 - val_accuracy: 0.8393 - val_loss: 0.3810
Epoch 5/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 268ms/step - accuracy: 0.8924 - loss: 0.2742 - val_accuracy: 0.7411 - val_loss: 0.5384
Epoch 6/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 282ms/step - accuracy: 0.8893 - loss: 0.2317 - val_accuracy: 0.8750 - val_loss: 0.4479
Epoch 7/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 286ms/step - accuracy: 0.9455 - loss: 0.1158 - val_accuracy: 0.8482 - val_loss: 0.5785
Epoch 8/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 278ms/step - accuracy: 0.9683 - loss: 0.0673 - val_accuracy: 0.8036 - val_loss: 0.7105
Epoch 9/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 8s 262ms/step - accuracy: 0.9628 - loss: 0.1193 - val_accuracy: 0.7946 - val_loss: 0.5149
Epoch 10/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 287ms/step - accuracy: 0.9199 - loss: 0.2032 - val_accuracy: 0.8929 - val_loss: 0.5320
Epoch 11/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 287ms/step - accuracy: 0.9666 - loss: 0.0825 - val_accuracy: 0.8839 - val_loss: 0.3939
Epoch 12/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 284ms/step - accuracy: 0.9850 - loss: 0.0457 - val_accuracy: 0.8929 - val_loss: 0.4053
Epoch 13/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 268ms/step - accuracy: 0.9990 - loss: 0.0218 - val_accuracy: 0.8571 - val_loss: 0.3969
Epoch 14/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 301ms/step - accuracy: 0.9949 - loss: 0.0125 - val_accuracy: 0.8571 - val_loss: 0.5753
Epoch 15/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 291ms/step - accuracy: 0.9888 - loss: 0.0216 - val_accuracy: 0.8929 - val_loss: 0.6058
Epoch 16/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 280ms/step - accuracy: 0.9970 - loss: 0.0071 - val_accuracy: 0.8750 - val_loss: 0.5613
Epoch 17/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 269ms/step - accuracy: 0.9887 - loss: 0.0170 - val_accuracy: 0.8571 - val_loss: 0.6406
Epoch 18/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 287ms/step - accuracy: 0.9860 - loss: 0.0420 - val_accuracy: 0.8036 - val_loss: 0.8985
Epoch 19/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 288ms/step - accuracy: 0.9735 - loss: 0.0846 - val_accuracy: 0.8482 - val_loss: 0.7470
Epoch 20/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 272ms/step - accuracy: 0.9688 - loss: 0.1297 - val_accuracy: 0.8839 - val_loss: 0.3924
Epoch 21/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 279ms/step - accuracy: 0.9935 - loss: 0.0336 - val_accuracy: 0.8661 - val_loss: 0.3479
Epoch 22/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 290ms/step - accuracy: 0.9810 - loss: 0.0538 - val_accuracy: 0.8750 - val_loss: 0.5166
Epoch 23/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 287ms/step - accuracy: 0.9863 - loss: 0.0277 - val_accuracy: 0.8661 - val_loss: 0.2969
Epoch 24/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 268ms/step - accuracy: 0.9885 - loss: 0.0247 - val_accuracy: 0.9018 - val_loss: 0.3495
Epoch 25/25
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 283ms/step - accuracy: 0.9990 - loss: 0.0116 - val_accuracy: 0.9018 - val_loss: 0.4287
```

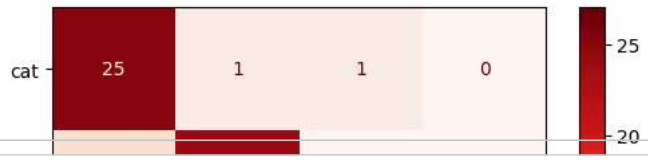
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,5))
```

```
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='train accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title("Accuracy")
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title("Loss")
plt.legend()

plt.show()
```




```
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Predict on test set
y_pred = model.predict(test_gen)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = test_gen.classes

# Confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)
disp = ConfusionMatrixDisplay(cm, display_labels=list(train_gen.class_indices.keys()))
disp.plot(cmap=plt.cm.Reds)
plt.show()

# Test accuracy
test_acc = np.sum(y_pred_classes==y_true)/len(y_true)
print("Test Accuracy:", test_acc)
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 120ms/step
```



Start coding or generate with AI.