```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in c:\users\monik\anaconda3\lib\site-packages (2.19.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (2.2.2)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (4.25.3)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (4.11.0)
Requirement already satisfied: wrapt>=1.11.0 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard~=2.19.0 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (3.10.0)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (1.26.4)
Requirement already satisfied: h5py>=3.11.0 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (3.11.0)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in c:\users\monik\anaconda3\lib\site-packages (from tensorflow) (0.5.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\monik\anaconda3\lib\site-packages (from astunparse>=1.6.0->tensorflow) (0.44.0)
```

```
Requirement already satisfied: rich in c:\users\monik\anaconda3\lib\site-packages (from keras>=3.5.0->tensorflow) (13.7.1)
Requirement already satisfied: namex in c:\users\monik\anaconda3\lib\site-packages (from keras>=3.5.0->tensorflow) (0.0.9)
Requirement already satisfied: optree in c:\users\monik\anaconda3\lib\site-packages (from keras>=3.5.0->tensorflow) (0.15.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\monik\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\monik\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\monik\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorflow) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\monik\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorflow) (2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in c:\users\monik\anaconda3\lib\site-packages (from tensorboard~=2.19.0->tensorflow) (3.4.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in c:\users\monik\anaconda3\lib\site-packages (from tensorboard~=2.19.0->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\monik\anaconda3\lib\site-packages (from tensorboard~=2.19.0->tensorflow) (3.0.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\monik\anaconda3\lib\site-packages (from werkzeug>=1.0.1->tensorboard~=2.19.0->tensorflow) (2.1.3)
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\users\monik\anaconda3\lib\site-packages (from rich->keras>=3.5.0->tensorflow) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\monik\anaconda3\lib\site-packages (from rich->keras>=3.5.0->tensorflow) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in c:\users\monik\anaconda3\lib\site-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow) (0.1.0)
Note: you may need to restart the kernel to use updated packages.
```

```python
import tensorflow as tf

 from tensorflow.keras import layers, models

 import matplotlib.pyplot as plt

import os

data_dir=r"C:\Monika\AnimalProject\animals"
```

```python
# List all files in the directory
files = os.listdir(data_dir)
print(files)
```

```
['antelope', 'badger', 'bat', 'bear', 'bee', 'beetle', 'bison',
'boar', 'butterfly', 'cat', 'caterpillar', 'chimpanzee', 'cockroach',
'cow', 'coyote', 'crab', 'crow', 'deer', 'dog', 'dolphin', 'donkey',
'dragonfly', 'duck', 'eagle', 'elephant', 'flamingo', 'fly', 'fox',
'goat', 'goldfish', 'goose', 'gorilla', 'grasshopper', 'hamster',
'hare', 'hedgehog', 'hippopotamus', 'hornbill', 'horse',
'hummingbird', 'hyena', 'jellyfish', 'kangaroo', 'koala', 'ladybugs',
'leopard', 'lion', 'lizard', 'lobster', 'mosquito', 'moth', 'mouse',
'octopus', 'okapi', 'orangutan', 'otter', 'owl', 'ox', 'oyster',
'panda', 'parrot', 'pelecaniformes', 'penguin', 'pig', 'pigeon',
'porcupine', 'possum', 'raccoon', 'rat', 'reindeer', 'rhinoceros',
'sandpiper', 'seahorse', 'seal', 'shark', 'sheep', 'snake', 'sparrow',
'squid', 'squirrel', 'starfish', 'swan', 'tiger', 'turkey', 'turtle',
'whale', 'wolf', 'wombat', 'woodpecker', 'zebra']
```

```python
# Loop through each subfolder in the main directory to display the
number of files in each subfolder
for folder_name in os.listdir(data_dir):
    folder_path = os.path.join(data_dir, folder_name)

    # Make sure it's a folder (not a file)
    if os.path.isdir(folder_path):
        # List files in the subfolder
        files = [f for f in os.listdir(folder_path) if
os.path.isfile(os.path.join(folder_path, f))]
        print(f"{folder_name}: {len(files)} files")
```

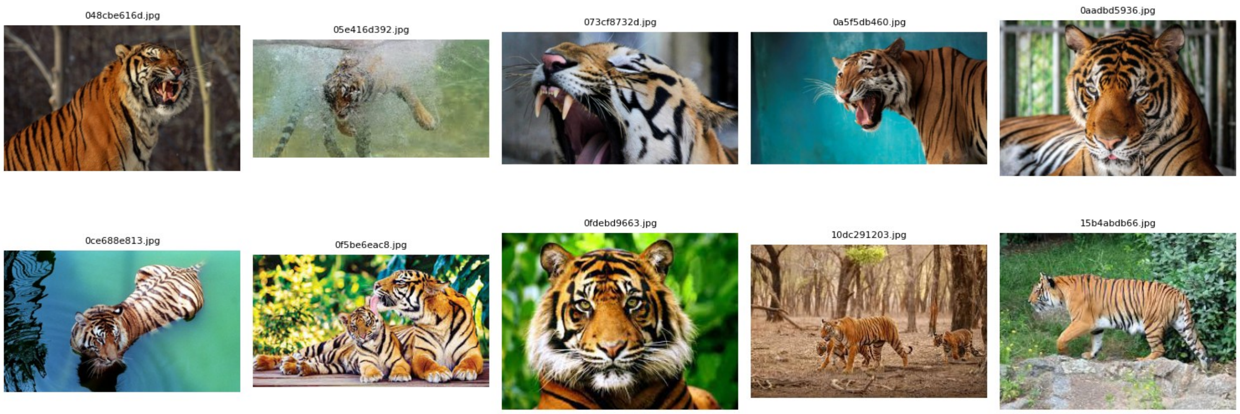
```
antelope: 60 files
badger: 60 files
bat: 60 files
bear: 60 files
bee: 60 files
beetle: 60 files
bison: 60 files
boar: 60 files
butterfly: 60 files
cat: 60 files
caterpillar: 60 files
chimpanzee: 60 files
cockroach: 60 files
cow: 60 files
coyote: 60 files
crab: 60 files
crow: 60 files
deer: 60 files
dog: 60 files
```

```
dolphin: 60 files
donkey: 60 files
dragonfly: 60 files
duck: 60 files
eagle: 60 files
elephant: 60 files
flamingo: 60 files
fly: 60 files
fox: 60 files
goat: 60 files
goldfish: 60 files
goose: 60 files
gorilla: 60 files
grasshopper: 60 files
hamster: 60 files
hare: 60 files
hedgehog: 60 files
hippopotamus: 60 files
hornbill: 60 files
horse: 60 files
hummingbird: 60 files
hyena: 60 files
jellyfish: 60 files
kangaroo: 60 files
koala: 60 files
ladybugs: 60 files
leopard: 60 files
lion: 60 files
lizard: 60 files
lobster: 60 files
mosquito: 60 files
moth: 60 files
mouse: 60 files
octopus: 60 files
okapi: 60 files
orangutan: 60 files
otter: 60 files
owl: 60 files
ox: 60 files
oyster: 60 files
panda: 60 files
parrot: 60 files
pelecaniformes: 60 files
penguin: 60 files
pig: 60 files
pigeon: 60 files
porcupine: 60 files
possum: 60 files
raccoon: 60 files
```

```
rat: 60 files
reindeer: 60 files
rhinoceros: 60 files
sandpiper: 60 files
seahorse: 60 files
seal: 60 files
shark: 60 files
sheep: 60 files
snake: 60 files
sparrow: 60 files
squid: 60 files
squirrel: 60 files
starfish: 60 files
swan: 60 files
tiger: 60 files
turkey: 60 files
turtle: 60 files
whale: 60 files
wolf: 60 files
wombat: 60 files
woodpecker: 60 files
zebra: 60 files
```

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

# Path to the image folder
folder_path = r"C:\Monika\AnimalProject\animals\tiger"

# Get list of image files (filter to image extensions)
image_files = [f for f in os.listdir(folder_path) if
f.lower().endswith(('.png', '.jpg', '.jpeg'))]

# Number of images to display
num_images = min(10, len(image_files))

# Set figure size (wider and taller for 2 rows)
plt.figure(figsize=(15, 6))

# Display images with file names
for i in range(num_images):
    file_name = image_files[i]
    img_path = os.path.join(folder_path, file_name)
    img = mpimg.imread(img_path)

    plt.subplot(2, 5, i + 1)  # 2 rows, 5 columns
    plt.imshow(img)
    plt.axis('off')
    plt.title(file_name, fontsize=8)
```

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




```
img_size=(224,224)
batch_size = 32

 train_ds =
tf.keras.preprocessing.image_dataset_from_directory(data_dir,validatio
n_split=0.2,subset="training",seed=123,image_size=img_size,batch_size=
batch_size)
```

```
Found 5400 files belonging to 90 classes.
Using 4320 files for training.
```

```
# Loop through each class folder
import math
validation_split = 0.2

for class_name in os.listdir(data_dir):
    class_path = os.path.join(data_dir, class_name)
    if os.path.isdir(class_path):
        files = [f for f in os.listdir(class_path) if
f.lower().endswith(('.png', '.jpg', '.jpeg'))]
        total_files = len(files)
        train_count = math.floor(total_files * (1 - validation_split))
        print(f"{class_name}: {train_count} training files out of
{total_files}")
```

```
antelope: 48 training files out of 60
badger: 48 training files out of 60
bat: 48 training files out of 60
bear: 48 training files out of 60
bee: 48 training files out of 60
beetle: 48 training files out of 60
bison: 48 training files out of 60
boar: 48 training files out of 60
```

```
butterfly: 48 training files out of 60
cat: 48 training files out of 60
caterpillar: 48 training files out of 60
chimpanzee: 48 training files out of 60
cockroach: 48 training files out of 60
cow: 48 training files out of 60
coyote: 48 training files out of 60
crab: 48 training files out of 60
crow: 48 training files out of 60
deer: 48 training files out of 60
dog: 48 training files out of 60
dolphin: 48 training files out of 60
donkey: 48 training files out of 60
dragonfly: 48 training files out of 60
duck: 48 training files out of 60
eagle: 48 training files out of 60
elephant: 48 training files out of 60
flamingo: 48 training files out of 60
fly: 48 training files out of 60
fox: 48 training files out of 60
goat: 48 training files out of 60
goldfish: 48 training files out of 60
goose: 48 training files out of 60
gorilla: 48 training files out of 60
grasshopper: 48 training files out of 60
hamster: 48 training files out of 60
hare: 48 training files out of 60
hedgehog: 48 training files out of 60
hippopotamus: 48 training files out of 60
hornbill: 48 training files out of 60
horse: 48 training files out of 60
hummingbird: 48 training files out of 60
hyena: 48 training files out of 60
jellyfish: 48 training files out of 60
kangaroo: 48 training files out of 60
koala: 48 training files out of 60
ladybugs: 48 training files out of 60
leopard: 48 training files out of 60
lion: 48 training files out of 60
lizard: 48 training files out of 60
lobster: 48 training files out of 60
mosquito: 48 training files out of 60
moth: 48 training files out of 60
mouse: 48 training files out of 60
octopus: 48 training files out of 60
okapi: 48 training files out of 60
orangutan: 48 training files out of 60
otter: 48 training files out of 60
owl: 48 training files out of 60
```

```
ox: 48 training files out of 60
oyster: 48 training files out of 60
panda: 48 training files out of 60
parrot: 48 training files out of 60
pelecaniformes: 48 training files out of 60
penguin: 48 training files out of 60
pig: 48 training files out of 60
pigeon: 48 training files out of 60
porcupine: 48 training files out of 60
possum: 48 training files out of 60
raccoon: 48 training files out of 60
rat: 48 training files out of 60
reindeer: 48 training files out of 60
rhinoceros: 48 training files out of 60
sandpiper: 48 training files out of 60
seahorse: 48 training files out of 60
seal: 48 training files out of 60
shark: 48 training files out of 60
sheep: 48 training files out of 60
snake: 48 training files out of 60
sparrow: 48 training files out of 60
squid: 48 training files out of 60
squirrel: 48 training files out of 60
starfish: 48 training files out of 60
swan: 48 training files out of 60
tiger: 48 training files out of 60
turkey: 48 training files out of 60
turtle: 48 training files out of 60
whale: 48 training files out of 60
wolf: 48 training files out of 60
wombat: 48 training files out of 60
woodpecker: 48 training files out of 60
zebra: 48 training files out of 60
```

```python
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
 data_dir,
 validation_split=0.2,
 subset="validation",
 seed=123,
 image_size=img_size,
 batch_size=batch_size
 )
```

```
Found 5400 files belonging to 90 classes.
Using 1080 files for validation.
```

```python
class_names = train_ds.class_names
print("Classes:", class_names)
```

```
Classes: ['antelope', 'badger', 'bat', 'bear', 'bee', 'beetle',
'bison', 'boar', 'butterfly', 'cat', 'caterpillar', 'chimpanzee',
'cockroach', 'cow', 'coyote', 'crab', 'crow', 'deer', 'dog',
'dolphin', 'donkey', 'dragonfly', 'duck', 'eagle', 'elephant',
'flamingo', 'fly', 'fox', 'goat', 'goldfish', 'goose', 'gorilla',
'grasshopper', 'hamster', 'hare', 'hedgehog', 'hippopotamus',
'hornbill', 'horse', 'hummingbird', 'hyena', 'jellyfish', 'kangaroo',
'koala', 'ladybugs', 'leopard', 'lion', 'lizard', 'lobster',
'mosquito', 'moth', 'mouse', 'octopus', 'okapi', 'orangutan', 'otter',
'owl', 'ox', 'oyster', 'panda', 'parrot', 'pelecaniformes', 'penguin',
'pig', 'pigeon', 'porcupine', 'possum', 'raccoon', 'rat', 'reindeer',
'rhinoceros', 'sandpiper', 'seahorse', 'seal', 'shark', 'sheep',
'snake', 'sparrow', 'squid', 'squirrel', 'starfish', 'swan', 'tiger',
'turkey', 'turtle', 'whale', 'wolf', 'wombat', 'woodpecker', 'zebra']
```

```
# highly recommended code for performance optimization when training
neural networks in TensorFlow.
AUTOTUNE = tf.data.AUTOTUNE
train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds
```

```
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None,),
dtype=tf.int32, name=None))>
```

```
# Data Augumentation: It's applied to training images to artificially
create new variations of the images
# by slightly altering them.

data_augmentation = tf.keras.Sequential([
 layers.RandomFlip("horizontal"),
 layers.RandomRotation(0.2),
 layers.RandomZoom(0.2),
 layers.RandomContrast(0.1)
 ])

data_augmentation
```

```
<Sequential name=sequential, built=False>
```

```
 from tensorflow.keras.applications import MobileNetV2
 from tensorflow.keras import layers, models

# Load the base MobileNetV2 model
base_model = MobileNetV2(input_shape=(224, 224,
3),include_top=False,weights='imagenet')

base_model.trainable = False # Freeze it for now (transfer learning)
```

```
# Define your full model

model = models.Sequential([ tf.keras.Input(shape=(224, 224,
3)),layers.Rescaling(1./255), base_model,
layers.GlobalAveragePooling2D(),  layers.Dense(128,
activation='relu'),layers.Dropout(0.5), layers.Dense(len(class_names),
activation='softmax')])


model
```

```
<Sequential name=sequential_1, built=True>
```

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
 initial_learning_rate=0.001,
 decay_steps=1000,
 decay_rate=0.9
 )

lr_schedule
```

```
<keras.src.optimizers.schedules.learning_rate_schedule.ExponentialDeca
y at 0x24900f88f80>
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)

optimizer
```

```
<keras.src.optimizers.adam.Adam at 0x24903736c90>
```

```
model.compile(optimizer='adam',
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])

early_stop = tf.keras.callbacks.EarlyStopping(patience=3,
restore_best_weights=True)

early_stop
```

```
<keras.src.callbacks.early_stopping.EarlyStopping at 0x24903737cb0>
```

```
model.fit(train_ds, validation_data=val_ds, epochs=20,
callbacks=[early_stop])
```

```
Epoch 1/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 65s 440ms/step - accuracy: 0.1223 - loss:
4.0752 - val_accuracy: 0.7185 - val_loss: 1.5139
Epoch 2/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 62s 459ms/step - accuracy: 0.5848 - loss:
1.7042 - val_accuracy: 0.8120 - val_loss: 0.8050
Epoch 3/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 61s 451ms/step - accuracy: 0.7308 - loss:
```

```
1.0399 - val_accuracy: 0.8315 - val_loss: 0.6781
Epoch 4/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 57s 424ms/step - accuracy: 0.7849 - loss:
0.7930 - val_accuracy: 0.8407 - val_loss: 0.6055
Epoch 5/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 58s 429ms/step - accuracy: 0.8294 - loss:
0.6225 - val_accuracy: 0.8444 - val_loss: 0.5527
Epoch 6/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 63s 468ms/step - accuracy: 0.8568 - loss:
0.5332 - val_accuracy: 0.8602 - val_loss: 0.4963
Epoch 7/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 59s 434ms/step - accuracy: 0.8719 - loss:
0.4413 - val_accuracy: 0.8528 - val_loss: 0.5044
Epoch 8/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 58s 433ms/step - accuracy: 0.8800 - loss:
0.4083 - val_accuracy: 0.8593 - val_loss: 0.4904
Epoch 9/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 56s 416ms/step - accuracy: 0.9017 - loss:
0.3378 - val_accuracy: 0.8583 - val_loss: 0.4736
Epoch 10/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 58s 428ms/step - accuracy: 0.9221 - loss:
0.2770 - val_accuracy: 0.8630 - val_loss: 0.4818
Epoch 11/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 58s 431ms/step - accuracy: 0.9298 - loss:
0.2584 - val_accuracy: 0.8611 - val_loss: 0.4669
Epoch 12/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 61s 452ms/step - accuracy: 0.9293 - loss:
0.2485 - val_accuracy: 0.8611 - val_loss: 0.4741
Epoch 13/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 58s 432ms/step - accuracy: 0.9379 - loss:
0.2179 - val_accuracy: 0.8611 - val_loss: 0.4584
Epoch 14/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 59s 441ms/step - accuracy: 0.9479 - loss:
0.1889 - val_accuracy: 0.8593 - val_loss: 0.4702
Epoch 15/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 60s 442ms/step - accuracy: 0.9514 - loss:
0.1741 - val_accuracy: 0.8611 - val_loss: 0.4680
Epoch 16/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 66s 493ms/step - accuracy: 0.9496 - loss:
0.1815 - val_accuracy: 0.8741 - val_loss: 0.4569
Epoch 17/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 59s 440ms/step - accuracy: 0.9543 - loss:
0.1711 - val_accuracy: 0.8648 - val_loss: 0.4502
Epoch 18/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 52s 383ms/step - accuracy: 0.9611 - loss:
0.1486 - val_accuracy: 0.8713 - val_loss: 0.4344
Epoch 19/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 54s 401ms/step - accuracy: 0.9546 - loss:
0.1478 - val_accuracy: 0.8648 - val_loss: 0.4908
```

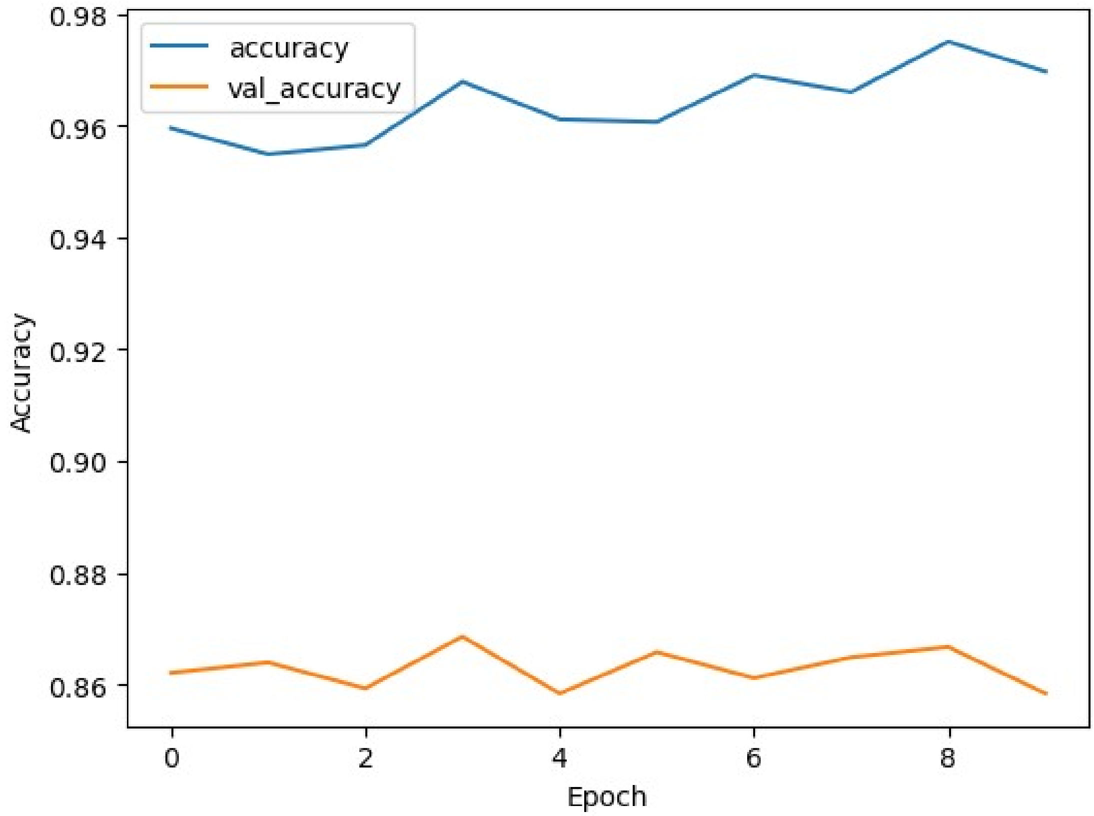
```
Epoch 20/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 53s 395ms/step - accuracy: 0.9534 - loss:
0.1463 - val_accuracy: 0.8704 - val_loss: 0.4713
```

```
<keras.src.callbacks.history.History at 0x249016112e0>
```

```
history = model.fit(
 train_ds,
 validation_data=val_ds,
 epochs=10
 )
```

```
Epoch 1/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 57s 420ms/step - accuracy: 0.9664 - loss:
0.1380 - val_accuracy: 0.8620 - val_loss: 0.4694
Epoch 2/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 58s 429ms/step - accuracy: 0.9554 - loss:
0.1480 - val_accuracy: 0.8639 - val_loss: 0.4809
Epoch 3/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 58s 428ms/step - accuracy: 0.9588 - loss:
0.1306 - val_accuracy: 0.8593 - val_loss: 0.4618
Epoch 4/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 57s 425ms/step - accuracy: 0.9669 - loss:
0.1104 - val_accuracy: 0.8685 - val_loss: 0.4824
Epoch 5/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 57s 425ms/step - accuracy: 0.9607 - loss:
0.1262 - val_accuracy: 0.8583 - val_loss: 0.5148
Epoch 6/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 57s 426ms/step - accuracy: 0.9590 - loss:
0.1285 - val_accuracy: 0.8657 - val_loss: 0.4887
Epoch 7/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 59s 437ms/step - accuracy: 0.9690 - loss:
0.1022 - val_accuracy: 0.8611 - val_loss: 0.4945
Epoch 8/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 60s 447ms/step - accuracy: 0.9645 - loss:
0.1080 - val_accuracy: 0.8648 - val_loss: 0.4902
Epoch 9/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 60s 443ms/step - accuracy: 0.9743 - loss:
0.0923 - val_accuracy: 0.8667 - val_loss: 0.4975
Epoch 10/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 59s 437ms/step - accuracy: 0.9692 - loss:
0.1055 - val_accuracy: 0.8583 - val_loss: 0.5257
```

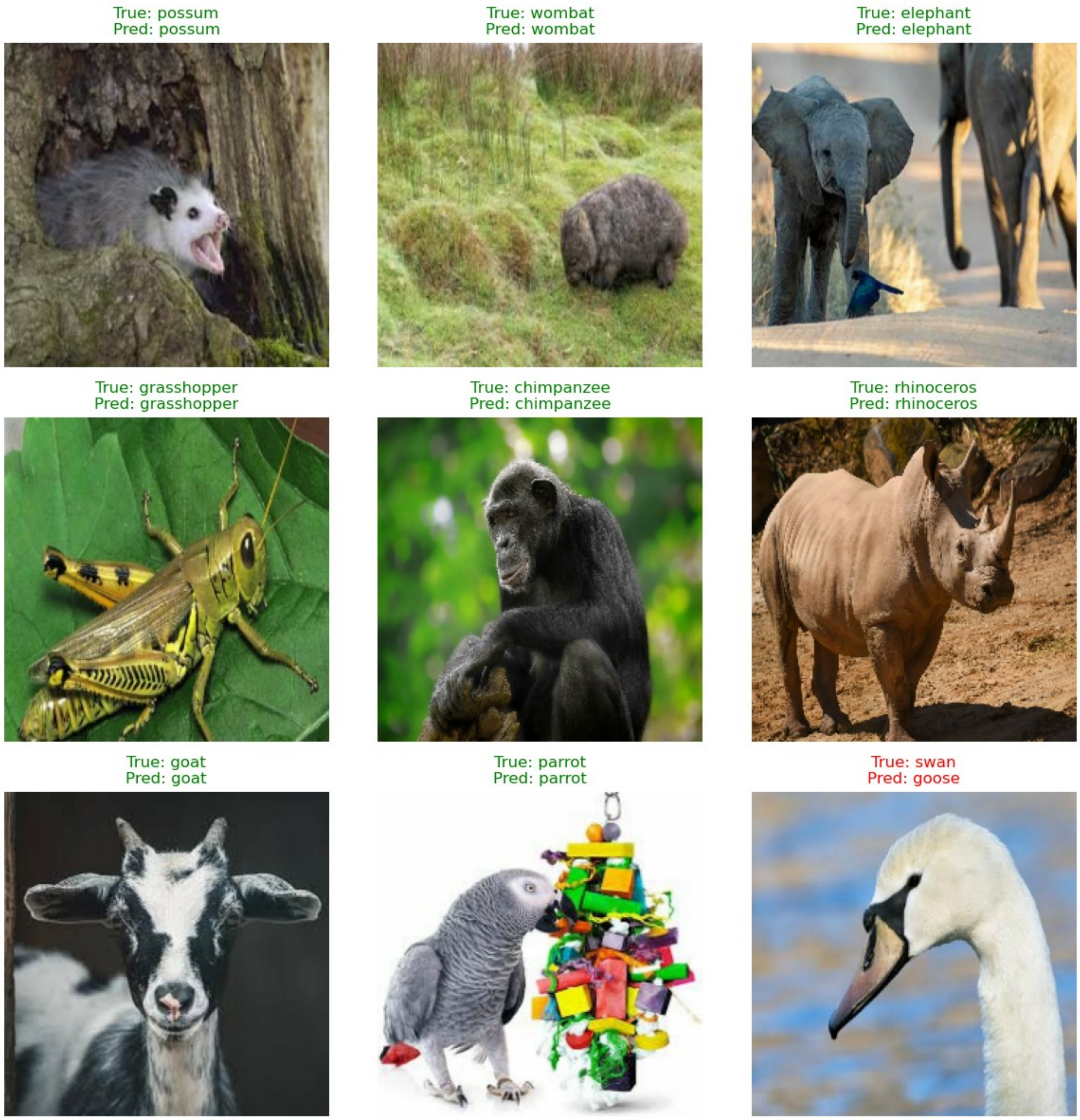
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
import numpy as np

for images, labels in val_ds.take(1):  # take 1 batch
    preds = model.predict(images)
    predicted_classes = np.argmax(preds, axis=1)
```

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

```
plt.figure(figsize=(12, 12))
for i in range(9):  # Show first 9 images
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        true_label = class_names[labels[i]]
        predicted_label = class_names[predicted_classes[i]]
        color = "green" if predicted_label == true_label else "red"
        plt.title(f"True: {true_label}\nPred: {predicted_label}",
color=color)
        plt.axis("off")
plt.tight_layout()
plt.show()
```




```python
from tensorflow.keras.applications import EfficientNetB0

# Load EfficientNetB0 as base model
efficientnet_base = EfficientNetB0(input_shape=(224, 224, 3),
include_top=False, weights='imagenet')
efficientnet_base.trainable = False

# Define the EfficientNet model
efficientnet_model = models.Sequential([
    tf.keras.Input(shape=(224, 224, 3)),
    layers.Rescaling(1./255),
```

```python
    data_augmentation,  # reuse augmentation
    efficientnet_base,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(len(class_names), activation='softmax')
])
```

```
Downloading data from https://storage.googleapis.com/keras-
applications/efficientnetb0_notop.h5
16705208/16705208 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

```python
efficientnet_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

efficientnet_history = efficientnet_model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=10,
    callbacks=[early_stop]
)
```

```
Epoch 1/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 97s 676ms/step - accuracy: 0.0091 - loss:
4.5258 - val_accuracy: 0.0046 - val_loss: 4.5009
Epoch 2/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 97s 719ms/step - accuracy: 0.0092 - loss:
4.5004 - val_accuracy: 0.0037 - val_loss: 4.5021
Epoch 3/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 104s 773ms/step - accuracy: 0.0137 -
loss: 4.4995 - val_accuracy: 0.0037 - val_loss: 4.5032
Epoch 4/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 104s 769ms/step - accuracy: 0.0163 -
loss: 4.4991 - val_accuracy: 0.0037 - val_loss: 4.5042
```

```python
# MobileNetV2 results
print("\nMobileNetV2 Final Accuracy:")
print(f"Train Accuracy: {history.history['accuracy'][-1]:.4f}")
print(f"Val Accuracy:   {history.history['val_accuracy'][-1]:.4f}")

# EfficientNetB0 results
print("\nEfficientNetB0 Final Accuracy:")
print(f"Train Accuracy: {efficientnet_history.history['accuracy'][-
1]:.4f}")
print(f"Val Accuracy:   {efficientnet_history.history['val_accuracy']
[-1]:.4f}")
```

```
MobileNetV2 Final Accuracy:
Train Accuracy: 0.9697
Val Accuracy:   0.8583

EfficientNetB0 Final Accuracy:
Train Accuracy: 0.0137
Val Accuracy:   0.0037
```

```python
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.applications.efficientnet import
preprocess_input
from tensorflow.keras import layers, models

# Load EfficientNetB0 with pretrained weights
efficientnet_base = EfficientNetB0(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet'
)

# Optionally fine-tune: freeze most, unfreeze last 20 layers
for layer in efficientnet_base.layers[:-20]:
    layer.trainable = False
for layer in efficientnet_base.layers[-20:]:
    layer.trainable = True

# Build the model
efficientnet_model = models.Sequential([
    tf.keras.Input(shape=(224, 224, 3)),
    data_augmentation,  # Apply augmentation first
    layers.Lambda(preprocess_input),  # EfficientNet-specific
preprocessing
    efficientnet_base,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(len(class_names), activation='softmax')
])

# Compile with a smaller learning rate (important for fine-tuning)
efficientnet_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Train the model
efficientnet_history = efficientnet_model.fit(
    train_ds,
```

```
    validation_data=val_ds,
    epochs=20,
    callbacks=[early_stop]  # Early stopping still applies
)
```

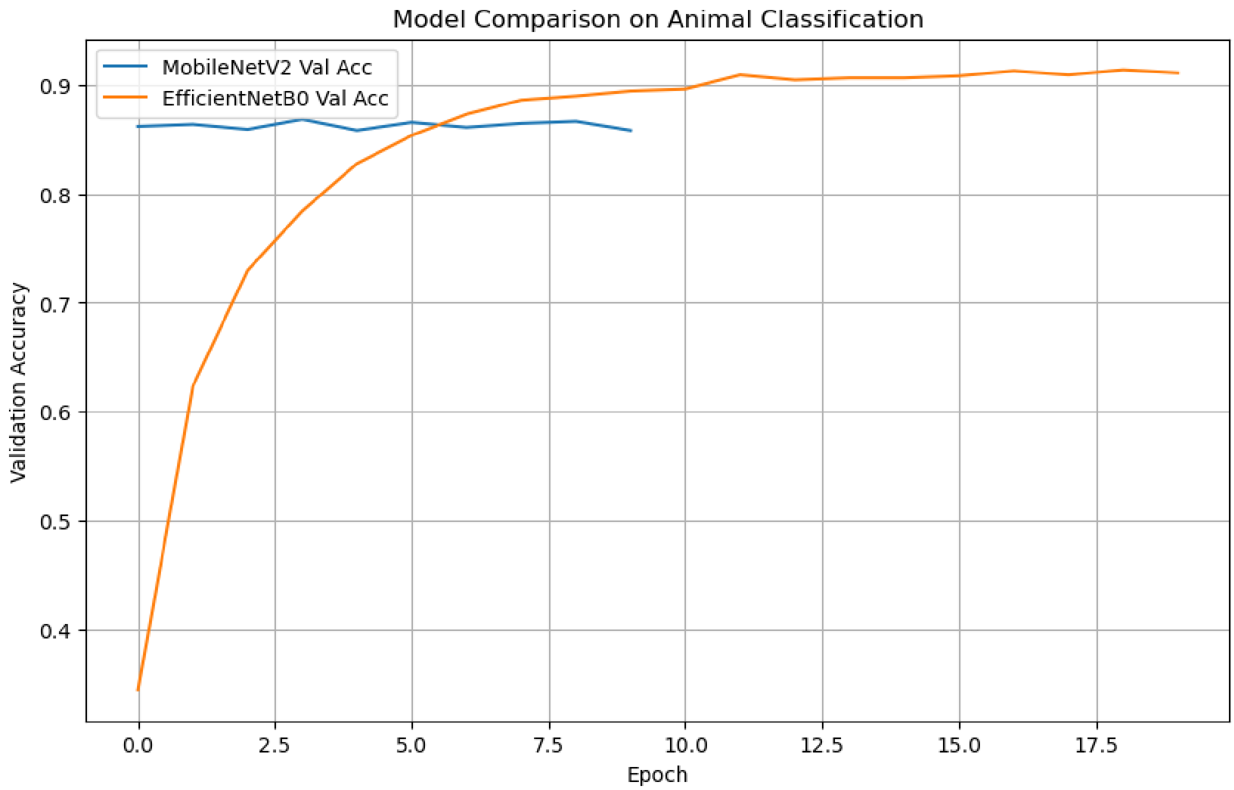
```
Epoch 1/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 118s 826ms/step - accuracy: 0.0259 -
loss: 4.4689 - val_accuracy: 0.3444 - val_loss: 3.8994
Epoch 2/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 116s 862ms/step - accuracy: 0.2072 -
loss: 3.8406 - val_accuracy: 0.6231 - val_loss: 2.6333
Epoch 3/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 115s 854ms/step - accuracy: 0.3856 -
loss: 2.9774 - val_accuracy: 0.7296 - val_loss: 1.6070
Epoch 4/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 117s 865ms/step - accuracy: 0.4791 -
loss: 2.3292 - val_accuracy: 0.7843 - val_loss: 1.1185
Epoch 5/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 118s 875ms/step - accuracy: 0.5626 -
loss: 1.8771 - val_accuracy: 0.8278 - val_loss: 0.8632
Epoch 6/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 119s 879ms/step - accuracy: 0.6171 -
loss: 1.5917 - val_accuracy: 0.8537 - val_loss: 0.7036
Epoch 7/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 114s 844ms/step - accuracy: 0.6663 -
loss: 1.3900 - val_accuracy: 0.8731 - val_loss: 0.5928
Epoch 8/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 118s 872ms/step - accuracy: 0.7076 -
loss: 1.2069 - val_accuracy: 0.8861 - val_loss: 0.5317
Epoch 9/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 116s 864ms/step - accuracy: 0.7351 -
loss: 1.0827 - val_accuracy: 0.8898 - val_loss: 0.4807
Epoch 10/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 117s 866ms/step - accuracy: 0.7515 -
loss: 1.0045 - val_accuracy: 0.8944 - val_loss: 0.4502
Epoch 11/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 122s 902ms/step - accuracy: 0.7761 -
loss: 0.9101 - val_accuracy: 0.8963 - val_loss: 0.4141
Epoch 12/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 122s 904ms/step - accuracy: 0.7822 -
loss: 0.8531 - val_accuracy: 0.9093 - val_loss: 0.3977
Epoch 13/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 116s 862ms/step - accuracy: 0.7963 -
loss: 0.7774 - val_accuracy: 0.9046 - val_loss: 0.3909
Epoch 14/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 121s 896ms/step - accuracy: 0.8183 -
loss: 0.7171 - val_accuracy: 0.9065 - val_loss: 0.3702
Epoch 15/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 117s 864ms/step - accuracy: 0.8304 -
loss: 0.6879 - val_accuracy: 0.9065 - val_loss: 0.3583
```

```
Epoch 16/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 120s 888ms/step - accuracy: 0.8441 -
loss: 0.6095 - val_accuracy: 0.9083 - val_loss: 0.3543
Epoch 17/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 115s 856ms/step - accuracy: 0.8494 -
loss: 0.6426 - val_accuracy: 0.9130 - val_loss: 0.3432
Epoch 18/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 115s 855ms/step - accuracy: 0.8554 -
loss: 0.5653 - val_accuracy: 0.9093 - val_loss: 0.3301
Epoch 19/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 114s 846ms/step - accuracy: 0.8693 -
loss: 0.5138 - val_accuracy: 0.9139 - val_loss: 0.3253
Epoch 20/20
135/135 ━━━━━━━━━━━━━━━━━━━━ 109s 805ms/step - accuracy: 0.8760 -
loss: 0.5051 - val_accuracy: 0.9111 - val_loss: 0.3167
```

```python
print("\nMobileNetV2 Final Accuracy:")
print(f"Train Accuracy: {history.history['accuracy'][-1]:.4f}")
print(f"Val Accuracy:   {history.history['val_accuracy'][-1]:.4f}")

print("\nEfficientNetB0 Final Accuracy:")
print(f"Train Accuracy: {efficientnet_history.history['accuracy'][-
1]:.4f}")
print(f"Val Accuracy:   {efficientnet_history.history['val_accuracy']
[-1]:.4f}")
```

```
MobileNetV2 Final Accuracy:
Train Accuracy: 0.9697
Val Accuracy:   0.8583

EfficientNetB0 Final Accuracy:
Train Accuracy: 0.8706
Val Accuracy:   0.9111
```

```python
plt.figure(figsize=(10, 6))
plt.plot(history.history['val_accuracy'], label='MobileNetV2 Val Acc')
plt.plot(efficientnet_history.history['val_accuracy'],
label='EfficientNetB0 Val Acc')
plt.xlabel('Epoch')
plt.ylabel('Validation Accuracy')
plt.title('Model Comparison on Animal Classification')
plt.legend()
plt.grid(True)
plt.show()
```




```python
from sklearn.metrics import classification_report
import numpy as np

# Collect true and predicted labels
true_labels = []
mobilenet_preds = []
efficientnet_preds = []

for images, labels in val_ds:
    true_labels.extend(labels.numpy())

    # MobileNetV2 predictions
    mobile_preds = model.predict(images)
    mobilenet_preds.extend(np.argmax(mobile_preds, axis=1))

    # EfficientNetB0 predictions
    eff_preds = efficientnet_model.predict(images)
    efficientnet_preds.extend(np.argmax(eff_preds, axis=1))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 249ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 233ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 316ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 223ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 284ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 224ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 301ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 216ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 288ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 223ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 297ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 226ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 300ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 248ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 295ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 248ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 313ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 219ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 299ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 230ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 308ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 220ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 306ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 230ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 285ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 218ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 288ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 214ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 301ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 219ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 319ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 214ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 286ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 225ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 302ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 223ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 301ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 220ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 300ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 220ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 299ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 233ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 303ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 229ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 294ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 220ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 317ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 291ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 519ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 389ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 583ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 424ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 557ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 435ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 601ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 420ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 565ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 423ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 527ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 406ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 583ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 378ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 572ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 378ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 541ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```

```python
print("=== MobileNetV2 Classification Report ===")
print(classification_report(true_labels, mobilenet_preds,
target_names=class_names))

print("\n=== EfficientNetB0 Classification Report ===")
print(classification_report(true_labels, efficientnet_preds,
target_names=class_names))
```

```
=== MobileNetV2 Classification Report ===
                precision    recall  f1-score   support

      antelope       0.81      0.93      0.87        14
        badger       0.88      0.88      0.88        16
           bat       0.67      0.40      0.50         5
          bear       0.88      0.78      0.82         9
           bee       0.92      1.00      0.96        12
        beetle       1.00      1.00      1.00         8
         bison       0.80      1.00      0.89        12
          boar       0.92      0.86      0.89        14
     butterfly       0.90      0.69      0.78        13
           cat       0.75      0.90      0.82        10
   caterpillar       0.83      0.62      0.71         8
    chimpanzee       0.93      1.00      0.96        13
     cockroach       0.90      1.00      0.95         9
           cow       0.71      0.83      0.77        12
        coyote       0.90      0.69      0.78        13
          crab       1.00      0.85      0.92        13
          crow       0.44      1.00      0.62         4
          deer       0.30      0.60      0.40         5
           dog       0.73      0.80      0.76        10
       dolphin       0.67      0.67      0.67        12
        donkey       0.90      0.82      0.86        11
     dragonfly       1.00      1.00      1.00         8
          duck       0.61      0.79      0.69        14
         eagle       1.00      1.00      1.00        13
      elephant       0.89      0.89      0.89         9
      flamingo       1.00      1.00      1.00        18
```

fly	0.82	1.00	0.90	9
fox	1.00	1.00	1.00	12
goat	0.80	0.57	0.67	14
goldfish	1.00	0.86	0.92	14
goose	0.71	1.00	0.83	10
gorilla	0.94	0.94	0.94	17
grasshopper	1.00	0.90	0.95	21
hamster	0.90	0.69	0.78	13
hare	1.00	0.80	0.89	10
hedgehog	1.00	0.88	0.93	8
hippopotamus	0.87	0.87	0.87	15
hornbill	1.00	1.00	1.00	16
horse	0.57	0.57	0.57	7
hummingbird	0.93	1.00	0.97	14
hyena	0.93	1.00	0.97	14
jellyfish	1.00	0.92	0.96	13
kangaroo	0.93	0.81	0.87	16
koala	1.00	0.93	0.97	15
ladybugs	0.89	1.00	0.94	8
leopard	0.90	1.00	0.95	9
lion	0.83	0.91	0.87	11
lizard	0.80	1.00	0.89	12
lobster	0.92	1.00	0.96	11
mosquito	0.89	0.80	0.84	10
moth	0.83	0.83	0.83	12
mouse	0.73	0.69	0.71	16
octopus	0.67	0.55	0.60	11
okapi	0.82	1.00	0.90	9
orangutan	1.00	0.92	0.96	12
otter	1.00	0.89	0.94	18
owl	0.71	0.56	0.62	9
ox	0.64	0.56	0.60	16
oyster	1.00	0.83	0.91	12
panda	1.00	0.92	0.96	12
parrot	0.83	0.91	0.87	11
pelecaniformes	0.93	1.00	0.97	14
penguin	0.82	1.00	0.90	9
pig	0.75	0.69	0.72	13
pigeon	0.89	0.53	0.67	15
porcupine	0.81	1.00	0.90	13
possum	0.69	0.75	0.72	12
raccoon	0.89	0.62	0.73	13
rat	0.73	0.69	0.71	16
reindeer	0.91	0.83	0.87	12
rhinoceros	0.69	0.92	0.79	12
sandpiper	1.00	0.91	0.95	11
seahorse	0.73	1.00	0.84	8
seal	0.80	0.92	0.86	13
shark	1.00	0.78	0.88	9

```
       sheep       0.75      0.55      0.63        11
       snake       0.88      1.00      0.93        14
     sparrow       1.00      1.00      1.00        11
       squid       0.75      0.82      0.78        11
    squirrel       0.85      1.00      0.92        11
    starfish       0.91      1.00      0.95        10
        swan       0.86      0.92      0.89        13
       tiger       1.00      0.88      0.93        16
      turkey       0.89      0.80      0.84        10
      turtle       1.00      0.77      0.87        13
       whale       0.75      0.82      0.78        11
        wolf       0.82      0.93      0.88        15
      wombat       0.71      0.91      0.80        11
  woodpecker       1.00      0.89      0.94        18
       zebra       1.00      1.00      1.00        13

    accuracy                           0.86      1080
   macro avg       0.86      0.86      0.85      1080
weighted avg       0.87      0.86      0.86      1080


=== EfficientNetB0 Classification Report ===
              precision    recall  f1-score   support

    antelope       1.00      1.00      1.00        14
      badger       1.00      0.94      0.97        16
         bat       0.50      0.20      0.29         5
        bear       0.90      1.00      0.95         9
         bee       1.00      0.92      0.96        12
      beetle       0.89      1.00      0.94         8
       bison       0.92      1.00      0.96        12
        boar       0.93      0.93      0.93        14
   butterfly       0.90      0.69      0.78        13
         cat       1.00      0.90      0.95        10
 caterpillar       0.71      0.62      0.67         8
  chimpanzee       1.00      1.00      1.00        13
   cockroach       1.00      1.00      1.00         9
         cow       0.73      0.92      0.81        12
      coyote       0.69      0.69      0.69        13
        crab       1.00      1.00      1.00        13
        crow       0.80      1.00      0.89         4
        deer       0.71      1.00      0.83         5
         dog       0.82      0.90      0.86        10
     dolphin       0.67      1.00      0.80        12
      donkey       0.83      0.91      0.87        11
   dragonfly       0.89      1.00      0.94         8
        duck       1.00      0.71      0.83        14
       eagle       0.87      1.00      0.93        13
    elephant       0.89      0.89      0.89         9
    flamingo       1.00      1.00      1.00        18
```

fly	1.00	1.00	1.00	9
fox	0.91	0.83	0.87	12
goat	0.91	0.71	0.80	14
goldfish	1.00	0.93	0.96	14
goose	0.69	0.90	0.78	10
gorilla	1.00	1.00	1.00	17
grasshopper	1.00	0.95	0.98	21
hamster	1.00	1.00	1.00	13
hare	1.00	0.60	0.75	10
hedgehog	1.00	1.00	1.00	8
hippopotamus	0.93	0.93	0.93	15
hornbill	1.00	0.94	0.97	16
horse	0.55	0.86	0.67	7
hummingbird	1.00	1.00	1.00	14
hyena	0.88	1.00	0.93	14
jellyfish	1.00	1.00	1.00	13
kangaroo	0.89	1.00	0.94	16
koala	1.00	0.93	0.97	15
ladybugs	1.00	1.00	1.00	8
leopard	1.00	1.00	1.00	9
lion	1.00	1.00	1.00	11
lizard	0.79	0.92	0.85	12
lobster	0.92	1.00	0.96	11
mosquito	1.00	1.00	1.00	10
moth	0.71	0.83	0.77	12
mouse	0.69	0.56	0.62	16
octopus	1.00	0.64	0.78	11
okapi	1.00	1.00	1.00	9
orangutan	1.00	1.00	1.00	12
otter	1.00	0.94	0.97	18
owl	0.90	1.00	0.95	9
ox	0.91	0.62	0.74	16
oyster	0.92	1.00	0.96	12
panda	0.92	1.00	0.96	12
parrot	1.00	1.00	1.00	11
pelecaniformes	0.78	1.00	0.88	14
penguin	0.90	1.00	0.95	9
pig	0.83	0.77	0.80	13
pigeon	1.00	0.87	0.93	15
porcupine	1.00	1.00	1.00	13
possum	0.92	0.92	0.92	12
raccoon	0.80	0.92	0.86	13
rat	0.67	0.75	0.71	16
reindeer	0.86	1.00	0.92	12
rhinoceros	0.79	0.92	0.85	12
sandpiper	1.00	0.91	0.95	11
seahorse	1.00	0.88	0.93	8
seal	1.00	1.00	1.00	13
shark	1.00	0.89	0.94	9

```
         sheep       1.00      0.73      0.84        11
         snake       1.00      1.00      1.00        14
       sparrow       1.00      1.00      1.00        11
         squid       0.83      0.91      0.87        11
      squirrel       0.91      0.91      0.91        11
      starfish       0.91      1.00      0.95        10
          swan       0.92      0.92      0.92        13
         tiger       1.00      0.94      0.97        16
        turkey       1.00      1.00      1.00        10
        turtle       1.00      0.92      0.96        13
         whale       1.00      0.73      0.84        11
          wolf       1.00      0.80      0.89        15
        wombat       0.77      0.91      0.83        11
    woodpecker       0.94      0.94      0.94        18
         zebra       1.00      1.00      1.00        13

      accuracy                           0.91      1080
     macro avg       0.91      0.91      0.90      1080
  weighted avg       0.92      0.91      0.91      1080
```

```
with open("classification_reports.txt", "w") as f:
    f.write("=== MobileNetV2 Classification Report ===\n")
    f.write(classification_report(true_labels, mobilenet_preds,
target_names=class_names))
    f.write("\n\n=== EfficientNetB0 Classification Report ===\n")
    f.write(classification_report(true_labels, efficientnet_preds,
target_names=class_names))
```