IMAGE DATA GENERATOR__Data Augmentaion,Keras

In [60]:
```
pip install --upgrade keras
```

```
Requirement already satisfied: keras in c:\users\roy62\anaconda3\lib\site-package
s (3.7.0)
Requirement already satisfied: absl-py in c:\users\roy62\anaconda3\lib\site-packa
ges (from keras) (2.1.0)
Requirement already satisfied: numpy in c:\users\roy62\anaconda3\lib\site-package
s (from keras) (1.26.4)
Requirement already satisfied: rich in c:\users\roy62\anaconda3\lib\site-packages
(from keras) (13.7.1)
Requirement already satisfied: namex in c:\users\roy62\anaconda3\lib\site-package
s (from keras) (0.0.8)
Requirement already satisfied: h5py in c:\users\roy62\anaconda3\lib\site-packages
(from keras) (3.11.0)
Requirement already satisfied: optree in c:\users\roy62\anaconda3\lib\site-packag
es (from keras) (0.13.1)
Requirement already satisfied: ml-dtypes in c:\users\roy62\anaconda3\lib\site-pac
kages (from keras) (0.4.1)
Requirement already satisfied: packaging in c:\users\roy62\anaconda3\lib\site-pac
kages (from keras) (24.1)
Requirement already satisfied: typing-extensions>=4.5.0 in c:\users\roy62\anacond
a3\lib\site-packages (from optree->keras) (4.11.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\users\roy62\anaconda3
\lib\site-packages (from rich->keras) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\roy62\anaconda
3\lib\site-packages (from rich->keras) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in c:\users\roy62\anaconda3\lib\site-pa
ckages (from markdown-it-py>=2.2.0->rich->keras) (0.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
from tensorflow import keras
```

In [3]:
```
pip show tensorflow
```

```
Name: tensorflow
Version: 2.18.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: C:\Users\roy62\anaconda3\Lib\site-packages
Requires: tensorflow-intel
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
pip show keras
```

```
Name: keras
Version: 3.7.0
Summary: Multi-backend Keras
Home-page:
Author:
Author-email: Keras team <keras-users@googlegroups.com>
License: Apache License 2.0
Location: C:\Users\roy62\anaconda3\Lib\site-packages
Requires: absl-py, h5py, ml-dtypes, namex, numpy, optree, packaging, rich
Required-by: tensorflow_intel
Note: you may need to restart the kernel to use updated packages.
```

In [5]:
```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in c:\users\roy62\anaconda3\lib\site-pa
ckages (2.18.0)
Requirement already satisfied: tensorflow-intel==2.18.0 in c:\users\roy62\anacond
a3\lib\site-packages (from tensorflow) (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\roy62\anaconda3\lib\sit
e-packages (from tensorflow-intel==2.18.0->tensorflow) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\roy62\anaconda3\lib
\site-packages (from tensorflow-intel==2.18.0->tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in c:\users\roy62\anaconda3\l
ib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in c:\users\ro
y62\anaconda3\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (0.6.
0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\roy62\anaconda3\li
b\site-packages (from tensorflow-intel==2.18.0->tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in c:\users\roy62\anaconda3\lib\s
ite-packages (from tensorflow-intel==2.18.0->tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\roy62\anaconda3\lib
\site-packages (from tensorflow-intel==2.18.0->tensorflow) (3.4.0)
Requirement already satisfied: packaging in c:\users\roy62\anaconda3\lib\site-pac
kages (from tensorflow-intel==2.18.0->tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.2
1.4,!=4.21.5,<6.0.0dev,>=3.20.3 in c:\users\roy62\anaconda3\lib\site-packages (fr
om tensorflow-intel==2.18.0->tensorflow) (4.25.3)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\roy62\anaconda3\li
b\site-packages (from tensorflow-intel==2.18.0->tensorflow) (2.32.3)
Requirement already satisfied: setuptools in c:\users\roy62\anaconda3\lib\site-pa
ckages (from tensorflow-intel==2.18.0->tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in c:\users\roy62\anaconda3\lib\site-p
ackages (from tensorflow-intel==2.18.0->tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\roy62\anaconda3\lib\s
ite-packages (from tensorflow-intel==2.18.0->tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\roy62\anacond
a3\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (4.11.0)
Requirement already satisfied: wrapt>=1.11.0 in c:\users\roy62\anaconda3\lib\site
-packages (from tensorflow-intel==2.18.0->tensorflow) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\roy62\anaconda3\li
b\site-packages (from tensorflow-intel==2.18.0->tensorflow) (1.68.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in c:\users\roy62\anaconda
3\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in c:\users\roy62\anaconda3\lib\site-
packages (from tensorflow-intel==2.18.0->tensorflow) (3.7.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in c:\users\roy62\anaconda3\l
ib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (1.26.4)
Requirement already satisfied: h5py>=3.11.0 in c:\users\roy62\anaconda3\lib\site-
packages (from tensorflow-intel==2.18.0->tensorflow) (3.11.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in c:\users\roy62\anaconda
3\lib\site-packages (from tensorflow-intel==2.18.0->tensorflow) (0.4.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\roy62\anaconda3\lib
\site-packages (from astunparse>=1.6.0->tensorflow-intel==2.18.0->tensorflow) (0.
44.0)
Requirement already satisfied: rich in c:\users\roy62\anaconda3\lib\site-packages
(from keras>=3.5.0->tensorflow-intel==2.18.0->tensorflow) (13.7.1)
Requirement already satisfied: namex in c:\users\roy62\anaconda3\lib\site-package
s (from keras>=3.5.0->tensorflow-intel==2.18.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in c:\users\roy62\anaconda3\lib\site-packag
es (from keras>=3.5.0->tensorflow-intel==2.18.0->tensorflow) (0.13.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\roy62\anacond
a3\lib\site-packages (from requests<3,>=2.21.0->tensorflow-intel==2.18.0->tensorf
low) (3.3.2)
```

```
Requirement already satisfied: idna<4,>=2.5 in c:\users\roy62\anaconda3\lib\site-
packages (from requests<3,>=2.21.0->tensorflow-intel==2.18.0->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\roy62\anaconda3\lib
\site-packages (from requests<3,>=2.21.0->tensorflow-intel==2.18.0->tensorflow)
(2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\roy62\anaconda3\lib
\site-packages (from requests<3,>=2.21.0->tensorflow-intel==2.18.0->tensorflow)
(2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in c:\users\roy62\anaconda3\lib\si
te-packages (from tensorboard<2.19,>=2.18->tensorflow-intel==2.18.0->tensorflow)
(3.4.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in c:\users
\roy62\anaconda3\lib\site-packages (from tensorboard<2.19,>=2.18->tensorflow-inte
l==2.18.0->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\roy62\anaconda3\lib\si
te-packages (from tensorboard<2.19,>=2.18->tensorflow-intel==2.18.0->tensorflow)
(3.0.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\roy62\anaconda3\lib
\site-packages (from werkzeug>=1.0.1->tensorboard<2.19,>=2.18->tensorflow-intel==
2.18.0->tensorflow) (2.1.3)
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\users\roy62\anaconda3
\lib\site-packages (from rich->keras>=3.5.0->tensorflow-intel==2.18.0->tensorflo
w) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\roy62\anaconda
3\lib\site-packages (from rich->keras>=3.5.0->tensorflow-intel==2.18.0->tensorflo
w) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in c:\users\roy62\anaconda3\lib\site-pa
ckages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow-intel==2.18.0-
>tensorflow) (0.1.0)
Note: you may need to restart the kernel to use updated packages.
```

Nearest

In [5]:
```python
from tensorflow.keras.utils import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import ImageDataGenerator  # Correct i
from keras.utils import array_to_img, img_to_array, load_img

# Initialize ImageDataGenerator
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

In [6]:
```python
import tensorflow.keras as keras
```

In [14]:
```python
img = load_img(r"E:\Data Science & AI\Dataset files\Bike.jpg")
```

In [15]:
```python
img
```

Out[15]:

In [17]:
```
x = img_to_array(img) # this is a Numpy array with shape (3, 150, 150)
x = x.reshape((1,) + x.shape) # this is a Numpy array with shape (1, 3, 150,150)
# the .flow() command below generates batches of randomly transformed images
# and saves the results to the `preview/` directory
i = 0
for batch in datagen.flow(x, batch_size=1,save_to_dir=r"E:\Data Science & AI\Dat
    i += 1
    if i > 10:
        break # otherwise the generator would loop indefinitely
```

Reflect

In [18]:
```
from tensorflow.keras.utils import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import ImageDataGenerator  # Correct i
from keras.utils import array_to_img, img_to_array, load_img

# Initialize ImageDataGenerator
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='reflect'
)
```

In [20]:
```
img1 = load_img(r"E:\Data Science & AI\Dataset files\Bike1.jpg")
img1
```

Out[20]:

In [21]:
```
x = img_to_array(img1) # this is a Numpy array with shape (3, 150, 150)
x = x.reshape((1,) + x.shape) # this is a Numpy array with shape (1, 3, 150,150)
# the .flow() command below generates batches of randomly transformed images
# and saves the results to the `preview/` directory
i = 0
for batch in datagen.flow(x, batch_size=1,save_to_dir=r"E:\Data Science & AI\Dat
    i += 1
    if i > 10:
        break # otherwise the generator would loop indefinitely
```

ResNet50

In [22]:
```
keras.applications.ResNet50(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102967424/102967424 ━━━━━━━━━━━━━━━━━━━━ 429s 4us/step
```

Out[22]:
```
<Functional name=resnet50, built=True>
```

In [24]:
```
from keras.utils import array_to_img ,img_to_array, load_img
img2 = load_img(r"E:\Data Science & AI\Dataset files\Bike2.jpg")
img2
```

Out[24]:

In [26]:
```
import numpy as np
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input,de
from tensorflow.keras.preprocessing import image

# Load the ResNet-50 model pre-trained on ImageNet data
```

```
model = ResNet50(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Bike2.jpg" # replace with the pa
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Make predictions
predictions = model.predict(img_array)

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=3)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/dat
a/imagenet_class_index.json
35363/35363 ━━━━━━━━━━━━━━━━━━━━ 0s 9us/step
Predictions:
1: mountain_bike (0.26)
2: crash_helmet (0.25)
3: disk_brake (0.11)

Top Prediction Class Index: 671
```

ResNet50V2

In [27]:

```
import numpy as np
from tensorflow.keras.applications.resnet_v2 import ResNet50V2,preprocess_input,
from tensorflow.keras.preprocessing import image

# Load the ResNet50V2 model pre-trained on ImageNet data
model = ResNet50V2(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Bike.jpg" # replace with the pat
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Make predictions
predictions = model.predict(img_array)
# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=3)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
```

```
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels.h5
102869336/102869336 ━━━━━━━━━━━━━━━━━━━━ 168s 2us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
Predictions:
1: moped (0.63)
2: crash_helmet (0.30)
3: mountain_bike (0.04)

Top Prediction Class Index: 665
```

VGG16

In [30]:

```
import numpy as np
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input,decode_p
from tensorflow.keras.preprocessing import image

# Load the VGG16 model pre-trained on ImageNet data
model = VGG16(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Bike1.jpg" # replace with the pa
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Make predictions
predictions = model.predict(img_array)

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=3)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 749ms/step
Predictions:
1: moped (0.44)
2: motor_scooter (0.43)
3: crash_helmet (0.09)

Top Prediction Class Index: 665
```

In [28]:

```
from keras.utils import array_to_img ,img_to_array, load_img
img = load_img(r"E:\Data Science & AI\Dataset files\Picsart.jpg")
img
```

Out[28]:

In [32]:

```
import numpy as np
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input,decode_p
from tensorflow.keras.preprocessing import image

# Load the VGG16 model pre-trained on ImageNet data
model = VGG16(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Picsart.jpg" # replace with the
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Make predictions
predictions = model.predict(img_array)

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=3)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 695ms/step
Predictions:
1: pick (0.19)
2: digital_clock (0.06)
3: street_sign (0.06)

Top Prediction Class Index: 714
```

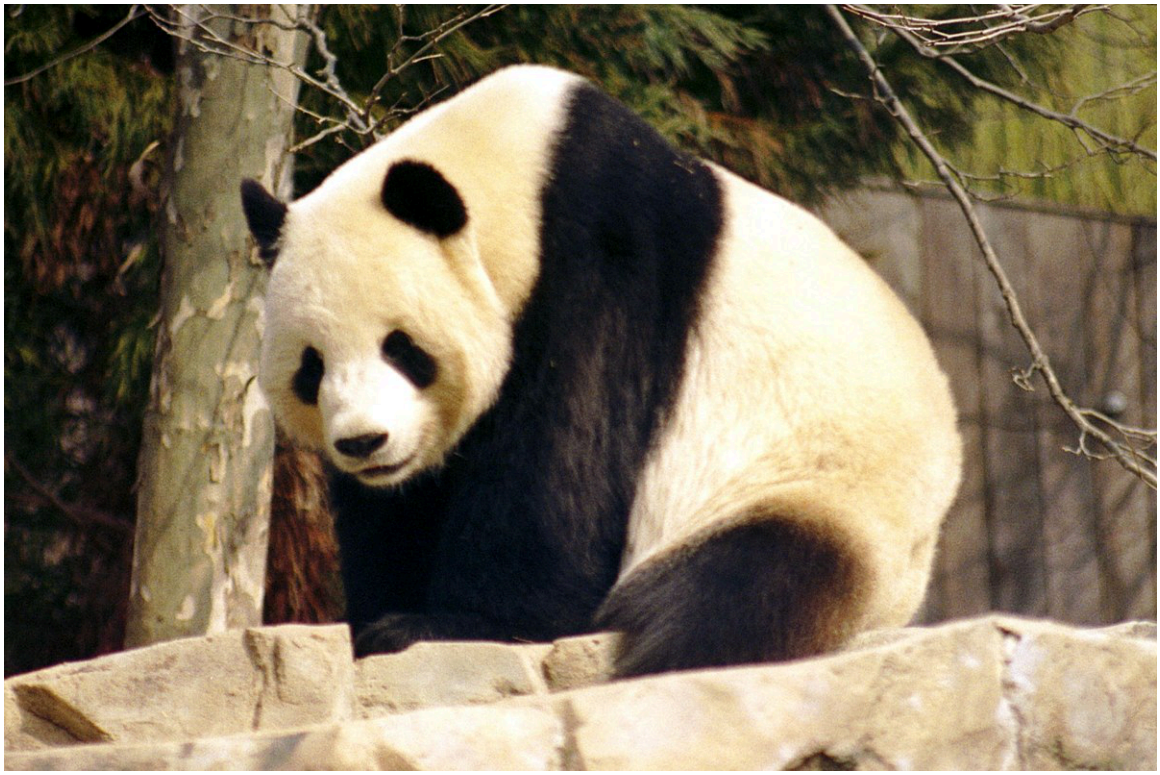
VGG19

In [35]:

```
from keras.utils import array_to_img ,img_to_array, load_img
img = load_img(r"E:\Data Science & AI\Dataset files\Animal.jpg")
```

```
img
```

Out[35]:



In [36]:

```
import numpy as np
from tensorflow.keras.applications.vgg19 import VGG19, preprocess_input,decode_p
from tensorflow.keras.preprocessing import image

# Load the VGG19 model pre-trained on ImageNet data
model = VGG19(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal.jpg" # replace with the p
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Make predictions
predictions = model.predict(img_array)

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=3)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.make_
predict_function.<locals>.one_step_on_data_distributed at 0x000002963C9A2F20> tri
ggered tf.function retracing. Tracing is expensive and the excessive number of tr
acings could be due to (1) creating @tf.function repeatedly in a loop, (2) passin
g tensors with different shapes, (3) passing Python objects instead of tensors. F
or (1), please define your @tf.function outside of the loop. For (2), @tf.functio
n has reduce_retracing=True option that can avoid unnecessary retracing. For (3),
please refer to https://www.tensorflow.org/guide/function#controlling_retracing a
nd https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
Predictions:
1: giant_panda (1.00)
2: lesser_panda (0.00)
3: American_black_bear (0.00)

Top Prediction Class Index: 388
```

In [37]:

```python
import numpy as np
from tensorflow.keras.applications.vgg19 import VGG19, preprocess_input,decode_p
from tensorflow.keras.preprocessing import image

# Load the VGG19 model pre-trained on ImageNet data
model = VGG19(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal.jpg" # replace with the_p
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Make predictions
predictions = model.predict(img_array)

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=8)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 914ms/step
Predictions:
1: giant_panda (1.00)
2: lesser_panda (0.00)
3: American_black_bear (0.00)
4: colobus (0.00)
5: ice_bear (0.00)
6: brown_bear (0.00)
7: Arctic_fox (0.00)
8: badger (0.00)

Top Prediction Class Index: 388
```

In [38]:

```python
import numpy as np
from tensorflow.keras.applications.resnet_v2 import ResNet50V2,preprocess_input,
from tensorflow.keras.preprocessing import image
```

```
# Load the ResNet50V2 model pre-trained on ImageNet data
model = ResNet50V2(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal.jpg" # replace with the p
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Make predictions
predictions = model.predict(img_array)

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=10)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
Predictions:
1: giant_panda (1.00)
2: soccer_ball (0.00)
3: dalmatian (0.00)
4: capuchin (0.00)
5: lesser_panda (0.00)
6: custard_apple (0.00)
7: hog (0.00)
8: paper_towel (0.00)
9: Siamese_cat (0.00)
10: toilet_tissue (0.00)

Top Prediction Class Index: 388
```

Predictions for Combainaton Animal image using Keras Api Applications.....

In [40]:
```
from keras.utils import array_to_img ,img_to_array, load_img
img = load_img(r"E:\Data Science & AI\Dataset files\Animal1.jpg")
img
```

Out[40]:

In [43]:
```
import time
```

In [44]:
```
import numpy as np
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input,de
from tensorflow.keras.preprocessing import image

# Load the ResNet-50 model pre-trained on ImageNet data
model = ResNet50(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal1.jpg" # replace with the
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Measure inference time
start_time = time.time()
predictions = model.predict(img_array)
end_time = time.time()

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=5)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")

# Calculate and print the inference time per step
inference_time_ms = (end_time - start_time) * 1000.0
print(f"Inference Time: {inference_time_ms:.2f} ms")

# Model summary provides information about parameters and layers
```

```
#model.summary()

# Get the size of the model in megabytes
model_size_MB = model.count_params() * 4 / (1024 ** 2) # 4 bytes for float32
print(f"Size (MB): {model_size_MB:.2f} MB")

# Get the number of parameters and depth from the model's layers
num_parameters = model.count_params()
model_depth = len(model.layers)
print(f"Parameters: {num_parameters}")
print(f"Depth: {model_depth}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
Predictions:
1: coyote (0.85)
2: red_wolf (0.07)
3: timber_wolf (0.04)
4: dingo (0.01)
5: wallaby (0.01)

Top Prediction Class Index: 272
Inference Time: 3444.70 ms
Size (MB): 97.80 MB
Parameters: 25636712
Depth: 177
```

In [45]:

```
import numpy as np
from tensorflow.keras.applications.resnet_v2 import ResNet50V2,preprocess_input,
from tensorflow.keras.preprocessing import image

# Load the ResNet50V2 model pre-trained on ImageNet data
model = ResNet50V2(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal1.jpg" # replace with the
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Measure inference time
start_time = time.time()
predictions = model.predict(img_array)
end_time = time.time()

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=5)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")

# Calculate and print the inference time per step
inference_time_ms = (end_time - start_time) * 1000.0
print(f"Inference Time: {inference_time_ms:.2f} ms")

# Model summary provides information about parameters and layers
```

```
#model.summary()

# Get the size of the model in megabytes
model_size_MB = model.count_params() * 4 / (1024 ** 2) # 4 bytes for float32
print(f"Size (MB): {model_size_MB:.2f} MB")

# Get the number of parameters and depth from the model's layers
num_parameters = model.count_params()
model_depth = len(model.layers)
print(f"Parameters: {num_parameters}")
print(f"Depth: {model_depth}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
Predictions:
1: coyote (0.93)
2: timber_wolf (0.05)
3: red_wolf (0.01)
4: grey_fox (0.00)
5: white_wolf (0.00)

Top Prediction Class Index: 272
Inference Time: 3723.50 ms
Size (MB): 97.71 MB
Parameters: 25613800
Depth: 192
```

In [46]:

```
import numpy as np
from tensorflow.keras.applications.vgg19 import VGG19, preprocess_input,decode_p
from tensorflow.keras.preprocessing import image
import time

# Load the VGG19 model pre-trained on ImageNet data
model = VGG19(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal1.jpg" # replace with the
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Measure inference time
start_time = time.time()
predictions = model.predict(img_array)
end_time = time.time()

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=5)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")

# Calculate and print the inference time per step
inference_time_ms = (end_time - start_time) * 1000.0
print(f"Inference Time: {inference_time_ms:.2f} ms")
```

```
# Model summary provides information about parameters and layers
#model.summary()

# Get the size of the model in megabytes
model_size_MB = model.count_params() * 4 / (1024 ** 2) # 4 bytes for float32
print(f"Size (MB): {model_size_MB:.2f} MB")

# Get the number of parameters and depth from the model's layers
num_parameters = model.count_params()
model_depth = len(model.layers)
print(f"Parameters: {num_parameters}")
print(f"Depth: {model_depth}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 778ms/step
Predictions:
1: timber_wolf (0.47)
2: coyote (0.31)
3: white_wolf (0.11)
4: red_wolf (0.09)
5: dingo (0.01)

Top Prediction Class Index: 269
Inference Time: 857.58 ms
Size (MB): 548.05 MB
Parameters: 143667240
Depth: 26
```

In [48]:

```
import numpy as np
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input,decode_p
from tensorflow.keras.preprocessing import image

# Load the VGG16 model pre-trained on ImageNet data
model = VGG16(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal.jpg" # replace with the_p
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Measure inference time
start_time = time.time()
predictions = model.predict(img_array)
end_time = time.time()

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=5)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")

# Calculate and print the inference time per step
inference_time_ms = (end_time - start_time) * 1000.0
print(f"Inference Time: {inference_time_ms:.2f} ms")
```

```
# Model summary provides information about parameters and layers
#model.summary()

# Get the size of the model in megabytes
model_size_MB = model.count_params() * 4 / (1024 ** 2) # 4 bytes for float32
print(f"Size (MB): {model_size_MB:.2f} MB")

# Get the number of parameters and depth from the model's layers
num_parameters = model.count_params()
model_depth = len(model.layers)
print(f"Parameters: {num_parameters}")
print(f"Depth: {model_depth}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 736ms/step
Predictions:
1: giant_panda (1.00)
2: ice_bear (0.00)
3: badger (0.00)
4: American_black_bear (0.00)
5: lesser_panda (0.00)

Top Prediction Class Index: 388
Inference Time: 821.63 ms
Size (MB): 527.79 MB
Parameters: 138357544
Depth: 23
```

In [49]:

```
import numpy as np
from tensorflow.keras.applications import Xception
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.xception import preprocess_input,decode_predi

# Load the Xception model pre-trained on ImageNet data
model = Xception(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal.jpg" # replace with the_p
img = image.load_img(img_path, target_size=(299, 299)) # Xception requires input
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Measure inference time
start_time = time.time()
predictions = model.predict(img_array)
end_time = time.time()

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=5)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")

# Calculate and print the inference time per step
inference_time_ms = (end_time - start_time) * 1000.0
print(f"Inference Time: {inference_time_ms:.2f} ms")
```

```
# Model summary provides information about parameters and layers
#model.summary()

# Get the size of the model in megabytes
model_size_MB = model.count_params() * 4 / (1024 ** 2) # 4 bytes for float32
print(f"Size (MB): {model_size_MB:.2f} MB")

# Get the number of parameters and depth from the model's layers
num_parameters = model.count_params()
model_depth = len(model.layers)
print(f"Parameters: {num_parameters}")
print(f"Depth: {model_depth}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/xception/xception_weights_tf_dim_ordering_tf_kernels.h5
91884032/91884032 ━━━━━━━━━━━━━━━━━━━━ 13s 0us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
Predictions:
1: giant_panda (0.95)
2: lesser_panda (0.01)
3: brown_bear (0.00)
4: American_black_bear (0.00)
5: ice_bear (0.00)

Top Prediction Class Index: 388
Inference Time: 2073.24 ms
Size (MB): 87.40 MB
Parameters: 22910480
Depth: 134
```

In [50]:

```
import numpy as np
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.inception_v3 import preprocess_input,decode_p

# Load the InceptionV3 model pre-trained on ImageNet data
model = InceptionV3(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal.jpg" # replace with the_p
img = image.load_img(img_path, target_size=(299, 299)) # Xception requires_input
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Measure inference time
start_time = time.time()
predictions = model.predict(img_array)
end_time = time.time()

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=5)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")
```

```
# Calculate and print the inference time per step
inference_time_ms = (end_time - start_time) * 1000.0
print(f"Inference Time: {inference_time_ms:.2f} ms")

# Model summary provides information about parameters and layers
#model.summary()

# Get the size of the model in megabytes
model_size_MB = model.count_params() * 4 / (1024 ** 2) # 4 bytes for float32
print(f"Size (MB): {model_size_MB:.2f} MB")

# Get the number of parameters and depth from the model's layers
num_parameters = model.count_params()
model_depth = len(model.layers)
print(f"Parameters: {num_parameters}")
print(f"Depth: {model_depth}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels.h5
96112376/96112376 ━━━━━━━━━━━━━━━━━━━━ 14s 0us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
Predictions:
1: giant_panda (0.96)
2: lesser_panda (0.00)
3: space_shuttle (0.00)
4: soccer_ball (0.00)
5: tray (0.00)

Top Prediction Class Index: 388
Inference Time: 4137.63 ms
Size (MB): 90.99 MB
Parameters: 23851784
Depth: 313
```

In [51]:

```
import numpy as np
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input,decode_p

# Load the MobileNetV2 model pre-trained on ImageNet data
model = MobileNetV2(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal.jpg" # replace with the_p
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Measure inference time
start_time = time.time()
predictions = model.predict(img_array)
end_time = time.time()

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=5)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")
```

```
# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")

# Calculate and print the inference time per step
inference_time_ms = (end_time - start_time) * 1000.0
print(f"Inference Time: {inference_time_ms:.2f} ms")

# Model summary provides information about parameters and layers
#model.summary()

# Get the size of the model in megabytes
model_size_MB = model.count_params() * 4 / (1024 ** 2) # 4 bytes for float32
print(f"Size (MB): {model_size_MB:.2f} MB")

# Get the number of parameters and depth from the model's layers
num_parameters = model.count_params()
model_depth = len(model.layers)
print(f"Parameters: {num_parameters}")
print(f"Depth: {model_depth}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224.h5
14536120/14536120 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
Predictions:
1: giant_panda (0.96)
2: lesser_panda (0.00)
3: ice_bear (0.00)
4: badger (0.00)
5: soccer_ball (0.00)

Top Prediction Class Index: 388
Inference Time: 2173.69 ms
Size (MB): 13.50 MB
Parameters: 3538984
Depth: 156
```

In [52]:

```
import numpy as np
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.densenet import preprocess_input,decode_predi
from tensorflow.keras.preprocessing import image

# Load the DenseNet121 model pre-trained on ImageNet data
model = DenseNet121(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal.jpg" # replace with the p
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Measure inference time
start_time = time.time()
predictions = model.predict(img_array)
end_time = time.time()

# Decode and print the top-3 predicted classes
```

```
decoded_predictions = decode_predictions(predictions, top=5)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")

# Calculate and print the inference time per step
inference_time_ms = (end_time - start_time) * 1000.0
print(f"Inference Time: {inference_time_ms:.2f} ms")

# Model summary provides information about parameters and layers
#model.summary()

# Get the size of the model in megabytes
model_size_MB = model.count_params() * 4 / (1024 ** 2) # 4 bytes for float32
print(f"Size (MB): {model_size_MB:.2f} MB")

# Get the number of parameters and depth from the model's layers
num_parameters = model.count_params()
model_depth = len(model.layers)
print(f"Parameters: {num_parameters}")
print(f"Depth: {model_depth}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/densenet/densenet121_weights_tf_dim_ordering_tf_kernels.h5
33188688/33188688 ━━━━━━━━━━━━━━━━━━━━ 6s 0us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 6s 6s/step
Predictions:
1: giant_panda (1.00)
2: lesser_panda (0.00)
3: badger (0.00)
4: brown_bear (0.00)
5: American_black_bear (0.00)

Top Prediction Class Index: 388
Inference Time: 5953.76 ms
Size (MB): 30.76 MB
Parameters: 8062504
Depth: 429
```

In [53]:

```
import numpy as np
from tensorflow.keras.applications import NASNetMobile
from tensorflow.keras.applications.nasnet import preprocess_input,decode_predict
from tensorflow.keras.preprocessing import image

# Load the NASNetMobile model pre-trained on ImageNet data
model = NASNetMobile(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal.jpg" # replace with the_p
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Measure inference time
start_time = time.time()
```

```
predictions = model.predict(img_array)
end_time = time.time()

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=5)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")

# Calculate and print the inference time per step
inference_time_ms = (end_time - start_time) * 1000.0
print(f"Inference Time: {inference_time_ms:.2f} ms")

# Model summary provides information about parameters and layers
#model.summary()

# Get the size of the model in megabytes
model_size_MB = model.count_params() * 4 / (1024 ** 2) # 4 bytes for float32
print(f"Size (MB): {model_size_MB:.2f} MB")
# Get the number of parameters and depth from the model's layers
num_parameters = model.count_params()
model_depth = len(model.layers)
print(f"Parameters: {num_parameters}")
print(f"Depth: {model_depth}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/nasnet/NASNet-mobile.h5
24227760/24227760 ━━━━━━━━━━━━━━━━━━━━ 19s 1us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 10s 10s/step
Predictions:
1: giant_panda (0.91)
2: lesser_panda (0.00)
3: colobus (0.00)
4: soccer_ball (0.00)
5: capuchin (0.00)

Top Prediction Class Index: 388
Inference Time: 9860.78 ms
Size (MB): 20.32 MB
Parameters: 5326716
Depth: 771
```

In [56]:

```
import numpy as np
from tensorflow.keras.applications import NASNetLarge
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.nasnet import preprocess_input,decode_predict

# Load the NASNetLarge model pre-trained on ImageNet data
model = NASNetLarge(weights='imagenet')

# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal1.jpg" # replace with the
img = image.load_img(img_path, target_size=(331, 331)) # NASNetLarge requires in
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)
```

```
# Measure inference time
start_time = time.time()
predictions = model.predict(img_array)
end_time = time.time()
# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=5)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")

# Calculate and print the inference time per step
inference_time_ms = (end_time - start_time) * 1000.0
print(f"Inference Time: {inference_time_ms:.2f} ms")

# Model summary provides information about parameters and layers
#model.summary()

# Get the size of the model in megabytes
model_size_MB = model.count_params() * 4 / (1024 ** 2) # 4 bytes for float32
print(f"Size (MB): {model_size_MB:.2f} MB")

# Get the number of parameters and depth from the model's layers
num_parameters = model.count_params()
model_depth = len(model.layers)
print(f"Parameters: {num_parameters}")
print(f"Depth: {model_depth}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/nasnet/NASNet-large.h5
359748576/359748576 ━━━━━━━━━━━━━━━━━━━━ 48s 0us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 13s 13s/step
Predictions:
1: coyote (0.64)
2: timber_wolf (0.20)
3: red_wolf (0.04)
4: white_wolf (0.00)
5: grey_fox (0.00)

Top Prediction Class Index: 272
Inference Time: 13029.63 ms
Size (MB): 339.32 MB
Parameters: 88949818
Depth: 1041
```

In [59]:

```
import numpy as np
from tensorflow.keras.applications import EfficientNetV2B0
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.efficientnet_v2 import preprocess_input,decod
# Load the EfficientNetV2B0 model pre-trained on ImageNet data
model = EfficientNetV2B0(weights='imagenet')
# Load and preprocess the input image
img_path = r"E:\Data Science & AI\Dataset files\Animal1.jpg" # replace with the_
img = image.load_img(img_path, target_size=(224, 224)) # NASNetLarge requires_in
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
```

```
img_array = preprocess_input(img_array)

# Measure inference time
start_time = time.time()
predictions = model.predict(img_array)
end_time = time.time()

# Decode and print the top-3 predicted classes
decoded_predictions = decode_predictions(predictions, top=5)[0]
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Optionally, you can obtain the class index for the top prediction
top_class_index = np.argmax(predictions[0])
print(f"\nTop Prediction Class Index: {top_class_index}")

# Calculate and print the inference time per step
inference_time_ms = (end_time - start_time) * 1000.0
print(f"Inference Time: {inference_time_ms:.2f} ms")

# Model summary provides information about parameters and layers
#model.summary()

# Get the size of the model in megabytes
model_size_MB = model.count_params() * 4 / (1024 ** 2) # 4 bytes for float32
print(f"Size (MB): {model_size_MB:.2f} MB")

# Get the number of parameters and depth from the model's layers
num_parameters = model.count_params()
model_depth = len(model.layers)
print(f"Parameters: {num_parameters}")
print(f"Depth: {model_depth}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/efficientnet_v2/efficientnetv2-b0.h5
29403144/29403144 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
Predictions:
1: coyote (0.81)
2: red_wolf (0.06)
3: timber_wolf (0.05)
4: white_wolf (0.01)
5: dingo (0.01)

Top Prediction Class Index: 272
Inference Time: 3879.04 ms
Size (MB): 27.47 MB
Parameters: 7200312
Depth: 273
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: