Data Augmentation

In [1]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import array_to_img, img_to_array, load_img

# Create an ImageDataGenerator instance
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 [5]:
```
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg",target_
img
```

Out[5]:

In [6]:
```
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, 15

# 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"C:\Users\chitt\OneDrive\Desktop\Data Aug
    i += 1
    if i > 10:
        break  # otherwise the generator would loop indefinitely
```

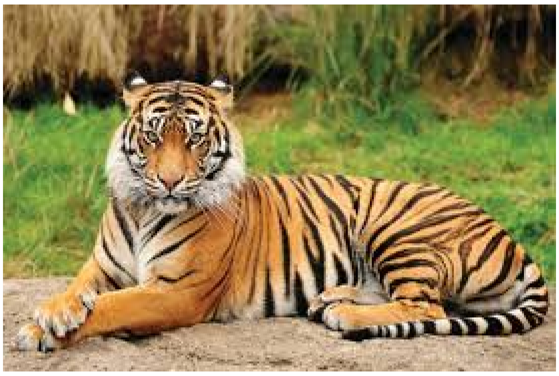
Prediction Of Images Using Keras Models

RESNET50

In [1]:
```
from keras.utils import array_to_img ,img_to_array, load_img
```

In [2]:
```
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg")
img
```

Out[2]:



In [8]:
```
import numpy as np
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input, d
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"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg"  # Replace
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})")  # Indented correctly

# 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}")
```

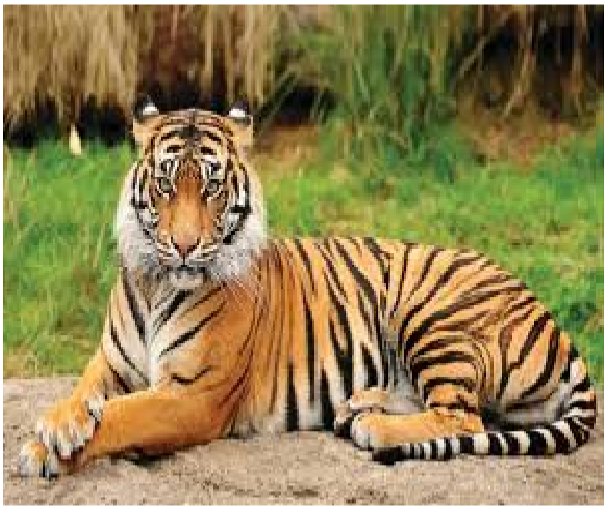
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102967424/102967424 ━━━━━━━━━━━━━━━━━━━━ 21s 0us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/dat
a/imagenet_class_index.json
35363/35363 ━━━━━━━━━━━━━━━━━━━━ 0s 1us/step
Predictions:
1: tiger (0.90)
2: tiger_cat (0.10)
3: zebra (0.00)

Top Prediction Class Index: 292
```

ResNet50V2

In [7]:

```
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg",target_
img
```

Out[7]:



In [10]:

```
import numpy as np
from tensorflow.keras.applications.resnet_v2 import ResNet50V2,preprocess_input,
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"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg"  # Replace
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})")  # Indented correctly

# 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}")
```

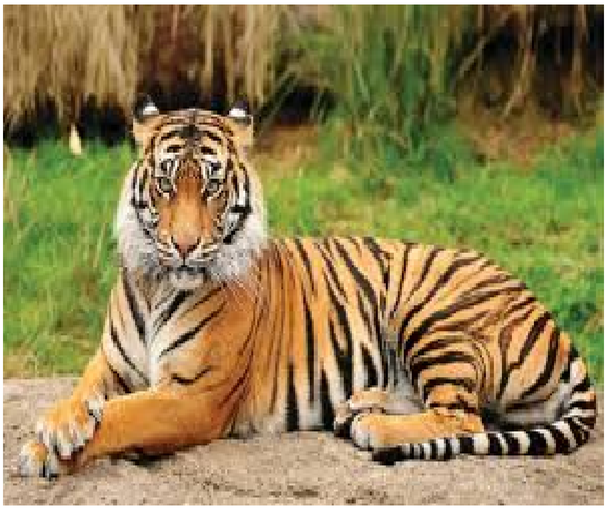
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
Predictions:
1: nematode (0.10)
2: matchstick (0.04)
3: nail (0.02)

Top Prediction Class Index: 111
```

VGG16

In [8]:

```
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg",target_
img
```

Out[8]:



In [13]:

```
import numpy as np
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input, decode_
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"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg"  # Replace
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}")
```

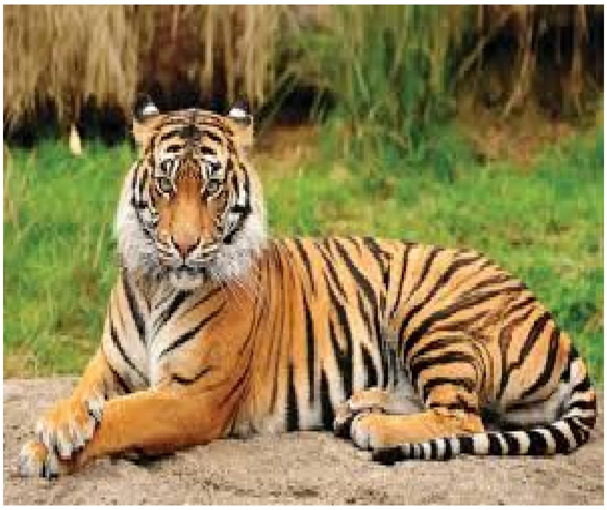
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 ━━━━━━━━━━━━━━━━━━━━ 165s 0us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 988ms/step
Predictions:
1: tiger (0.78)
2: tiger_cat (0.22)
3: lynx (0.00)

Top Prediction Class Index: 292
```

VGG19

In [9]:

```
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg",target_
img
```

Out[9]:



In [13]:

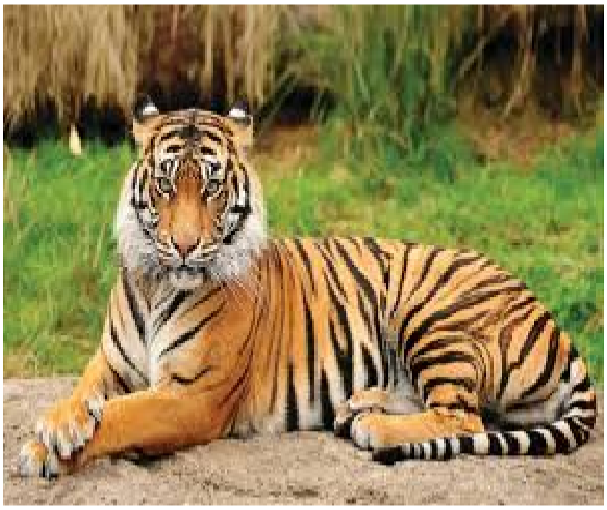
```
import numpy as np
from tensorflow.keras.applications.vgg19 import VGG19, preprocess_input, decode_
from tensorflow.keras.preprocessing import image

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

# Load and preprocess the input image
img_path = r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg"  # Replace
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 1s/step
Predictions:
1: tiger (0.78)
2: tiger_cat (0.22)
3: zebra (0.00)

Top Prediction Class Index: 292
```

Xception

In [14]:
```
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg",target_
img
```

Out[14]:



In [11]:
```
import numpy as np
from tensorflow.keras.applications import Xception
from tensorflow.keras.applications.xception import preprocess_input, decode_pred
from tensorflow.keras.preprocessing import image

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

# Load and preprocess the input image
img_path = r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg"  # Replace
img = image.load_img(img_path, target_size=(299, 299))  # Xception expects 299x2
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/xception/xception_weights_tf_dim_ordering_tf_kernels.h5
91884032/91884032 ━━━━━━━━━━━━━━━━━━━━ 9s 0us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
Predictions:
1: tiger (0.86)
2: tiger_cat (0.06)
3: jaguar (0.00)

Top Prediction Class Index: 292
```

InceptionV3

In [21]:

```
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg",target_
img
```

Out[21]:

In [16]:

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

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

# Load and preprocess the input image
img_path = r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg"  # Replace 
img = image.load_img(img_path, target_size=(299, 299))  # InceptionV3 expects 29
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-5 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})")
```

```python
# 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}")
```

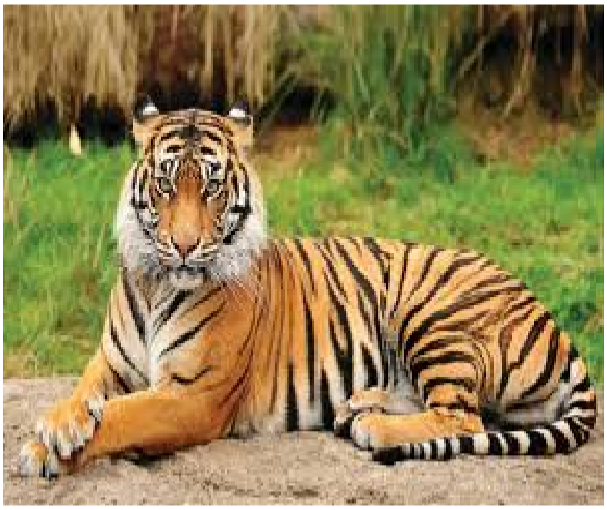
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels.h5
96112376/96112376 ━━━━━━━━━━━━━━━━━━━━ 10s 0us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 5s 5s/step
Predictions:
1: tiger (0.89)
2: tiger_cat (0.04)
3: lynx (0.00)
4: zebra (0.00)
5: jaguar (0.00)

Top Prediction Class Index: 292
Inference Time: 5045.19 ms
Size (MB): 90.99 MB
Parameters: 23851784
Depth: 313
```

MobileNetV2

In [22]:

```python
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg",target_
img
```

Out[22]:



In [20]:

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

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

# Load and preprocess the input image
img_path = r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg"  # Replace
img = image.load_img(img_path, target_size=(224, 224))  # MobileNetV2 expects 22
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-5 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 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
Predictions:
1: tiger (0.81)
2: tiger_cat (0.06)
3: jaguar (0.01)
4: zebra (0.00)
5: leopard (0.00)

Top Prediction Class Index: 292
Inference Time: 2125.15 ms
Size (MB): 13.50 MB
Parameters: 3538984
Depth: 156
```

MobileNetV2

In [24]:

```
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg",target_
img
```

Out[24]:

In [23]:

```
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"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg"  # Replace
img = image.load_img(img_path, target_size=(224, 224))  # MobileNetV2 expects 22
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-5 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}")
```

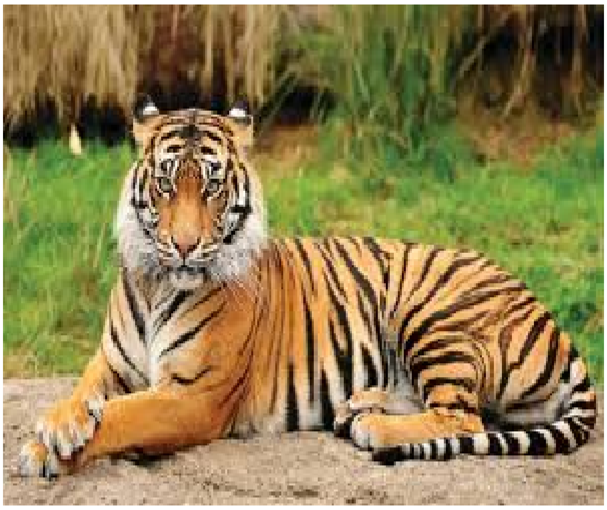
```
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_
predict_function.<locals>.one_step_on_data_distributed at 0x0000028134164C10> tri
ggered tf.function retracing. Tracing is expensive and the excessive number of tr
acings could be due to (1) creating @tf.function repeatedly 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: tiger (0.81)
2: tiger_cat (0.06)
3: jaguar (0.01)
4: zebra (0.00)
5: leopard (0.00)

Top Prediction Class Index: 292
Inference Time: 1996.47 ms
Size (MB): 13.50 MB
Parameters: 3538984
Depth: 156
```

DenseNet121

In [26]:

```
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg",target_
img
```

Out[26]:



In [25]:

```
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"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg"  # Replace
img = image.load_img(img_path, target_size=(224, 224))  # MobileNetV2 expects 22
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-5 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 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.make_
predict_function.<locals>.one_step_on_data_distributed at 0x000002813648D000> 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 ━━━━━━━━━━━━━━━━━━━━ 8s 8s/step
Predictions:
1: tiger (0.97)
2: tiger_cat (0.02)
3: zebra (0.00)
4: jaguar (0.00)
5: leopard (0.00)

Top Prediction Class Index: 292
Inference Time: 8462.83 ms
Size (MB): 30.76 MB
Parameters: 8062504
Depth: 429
```

NASNetMobile

In [28]:

```
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg",target_
img
```

Out[28]:

In [27]:

```
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 DenseNet121 model pre-trained on ImageNet data
model = DenseNet121(weights='imagenet')
# Load and preprocess the input image
img_path = r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg"  # Replace 
img = image.load_img(img_path, target_size=(224, 224))  # MobileNetV2 expects 22
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-5 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 ━━━━━━━━━━━━━━━━━━━━ 9s 9s/step
Predictions:
1: tiger (0.95)
2: tiger_cat (0.04)
3: zebra (0.00)
4: jaguar (0.00)
5: lynx (0.00)

Top Prediction Class Index: 292
Inference Time: 8761.78 ms
Size (MB): 30.76 MB
Parameters: 8062504
Depth: 429
```

NASNetLarge

In [30]:

```
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg",target_
img
```

Out[30]:

In [29]:

```
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 DenseNet121 model pre-trained on ImageNet data
model = DenseNet121(weights='imagenet')
# Load and preprocess the input image
img_path = r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg"  # Replace
img = image.load_img(img_path, target_size=(224, 224))  # MobileNetV2 expects 22
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-5 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 ━━━━━━━━━━━━━━━━━━━━ 10s 10s/step
Predictions:
1: tiger (0.95)
2: tiger_cat (0.04)
3: zebra (0.00)
4: jaguar (0.00)
5: lynx (0.00)

Top Prediction Class Index: 292
Inference Time: 10102.48 ms
Size (MB): 30.76 MB
Parameters: 8062504
Depth: 429
```

EfficientNetV2B0

In [31]:

```
img = load_img(r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg",target_
img
```

Out[31]:

In [32]:

```
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 DenseNet121 model pre-trained on ImageNet data
model = DenseNet121(weights='imagenet')
# Load and preprocess the input image
img_path = r"C:\Users\chitt\OneDrive\Pictures\image\Tiger Image.jpg"  # Replace
img = image.load_img(img_path, target_size=(224, 224))  # MobileNetV2 expects 22
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-5 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})")
```

```python
# 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 ━━━━━━━━━━━━━━━━━━━━ 9s 9s/step
Predictions:
1: matchstick (0.93)
2: honeycomb (0.06)
3: panpipe (0.01)
4: maraca (0.00)
5: web_site (0.00)

Top Prediction Class Index: 644
Inference Time: 9346.20 ms
Size (MB): 30.76 MB
Parameters: 8062504
Depth: 429
```

Completed

In [ ]: