Experiment No 11

Aim: Using a pre-trained Image net network to predict images into one of the 1000 Imagenet classes

What is ImageNet?

ImageNet is a large dataset that contains millions of labeled images across 1,000 different categories (e.g., animals, objects, scenes). It has been fundamental in advancing deep learning and computer vision research.

2. Pretrained Networks

A pretrained network refers to a neural network model that has already been trained on a specific dataset, such as ImageNet. Popular architectures include ResNet, VGG, and Inception. These models have learned to extract features from images effectively.

3. How Prediction Works

When using a pretrained network for image classification, the process generally involves the following steps:

a. Input Image Preparation

- **Resizing**: The input image is resized to the expected dimensions of the network (often 224x224 pixels for many architectures).
- **Normalization**: Pixel values are typically normalized to a specific range (e.g., 0 to 1 or standardized to have a mean of 0 and a standard deviation of 1).

b. Feature Extraction

- The image is fed into the pretrained network, which processes it through multiple layers (convolutions, pooling, etc.). Each layer extracts increasingly complex features from the image.

c. Final Classification Layer

- The output of the final layer is a vector of probabilities corresponding to the 1,000 ImageNet classes. This is usually done using a softmax function, which converts the raw scores into probabilities.

d. Making Predictions

- The class with the highest probability is selected as the predicted class for the input image.

4. Why Use Pretrained Networks?

- **Transfer Learning**: Pretrained models leverage the knowledge gained from training on a large dataset, which helps when you have limited data for your specific task.
- **Reduced Training Time**: You can fine-tune a pretrained model on a smaller dataset, which saves time and computational resources.
- **Improved Performance**: These models often achieve better accuracy than training a new model from scratch.

5. Implementation

In practice, using a pretrained network can be done easily with frameworks like TensorFlow or PyTorch. You typically load the model, preprocess your images, and use the model's `predict` function to get predictions.

6. Fine-tuning (Optional)

For specific applications, you might want to fine-tune the pretrained model by retraining it on a smaller, domain-specific dataset. This involves:

- Modifying the last layer(s) to match the number of classes in your new dataset.
- Training the model for a few epochs while possibly freezing earlier layers to retain the learned features.

In [1]:
```
# !pip install tensorflow==2.1.0
```

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from urllib.request import urlopen
from PIL import Image

%matplotlib inline
plt.style.use('default')

print("TF  Version",tf.__version__)
```

```
TF  Version 2.17.0
```

In [3]:
```
model_vgg=tf.keras.applications.vgg16.VGG16(include_top=True, weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/v
gg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

In [4]:
```
model_vgg.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	
input_layer (InputLayer)	(None, 224, 224, 3)	
block1_conv1 (Conv2D)	(None, 224, 224, 64)	
block1_conv2 (Conv2D)	(None, 224, 224, 64)	
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	
block2_conv1 (Conv2D)	(None, 112, 112, 128)	
block2_conv2 (Conv2D)	(None, 112, 112, 128)	
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	
block3_conv1 (Conv2D)	(None, 56, 56, 256)	
block3_conv2 (Conv2D)	(None, 56, 56, 256)	
block3_conv3 (Conv2D)	(None, 56, 56, 256)	
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	
block4_conv1 (Conv2D)	(None, 28, 28, 512)	
block4_conv2 (Conv2D)	(None, 28, 28, 512)	
block4_conv3 (Conv2D)	(None, 28, 28, 512)	
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	
block5_conv1 (Conv2D)	(None, 14, 14, 512)	
block5_conv2 (Conv2D)	(None, 14, 14, 512)	
block5_conv3 (Conv2D)	(None, 14, 14, 512)	
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	
flatten (Flatten)	(None, 25088)	
fc1 (Dense)	(None, 4096)	1
fc2 (Dense)	(None, 4096)	1
predictions (Dense)	(None, 1000)	

Total params: 138,357,544 (527.79 MB)

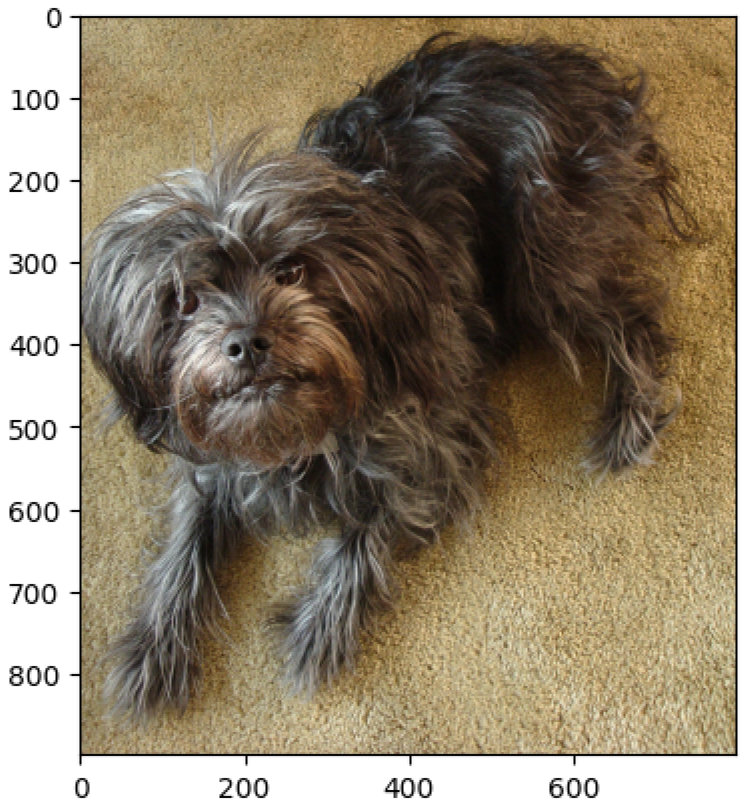
Trainable params: 138,357,544 (527.79 MB)

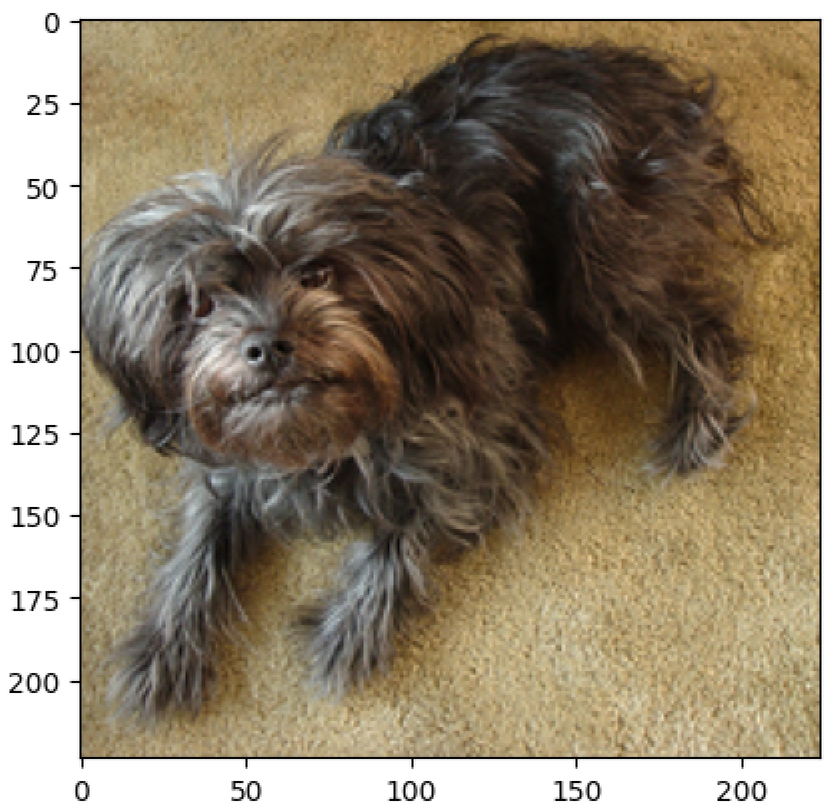
Non-trainable params: 0 (0.00 B)

In [5]:
```
def preprocess_input(img):
    x=np.zeros((224,224,3),dtype="float32")
    x[:,:,0]=img[:,:,2]
    x[:,:,1]=img[:,:,1]
    x[:,:,2]=img[:,:,0]
    mean = [103.939, 116.779, 123.68]
    x[:,:, 0] = x[:,:, 0]-mean[0]
    x[:,:, 1] = x[:,:, 1]-mean[1]
    x[:,:, 2] = x[:,:, 2]-mean[2]
    return x

def undo_preprocess_input(img):
    mean = [103.939, 116.779, 123.68]
    img[:,:, 0] = img[:,:, 0]+mean[0]
    img[:,:, 1] = img[:,:, 1]+mean[1]
    img[:,:, 2] = img[:,:, 2]+mean[2]
    x=np.zeros((224,224,3),dtype="float32")
    x[:,:,0]=img[:,:,2]
    x[:,:,1]=img[:,:,1]
    x[:,:,2]=img[:,:,0]
    return x
```

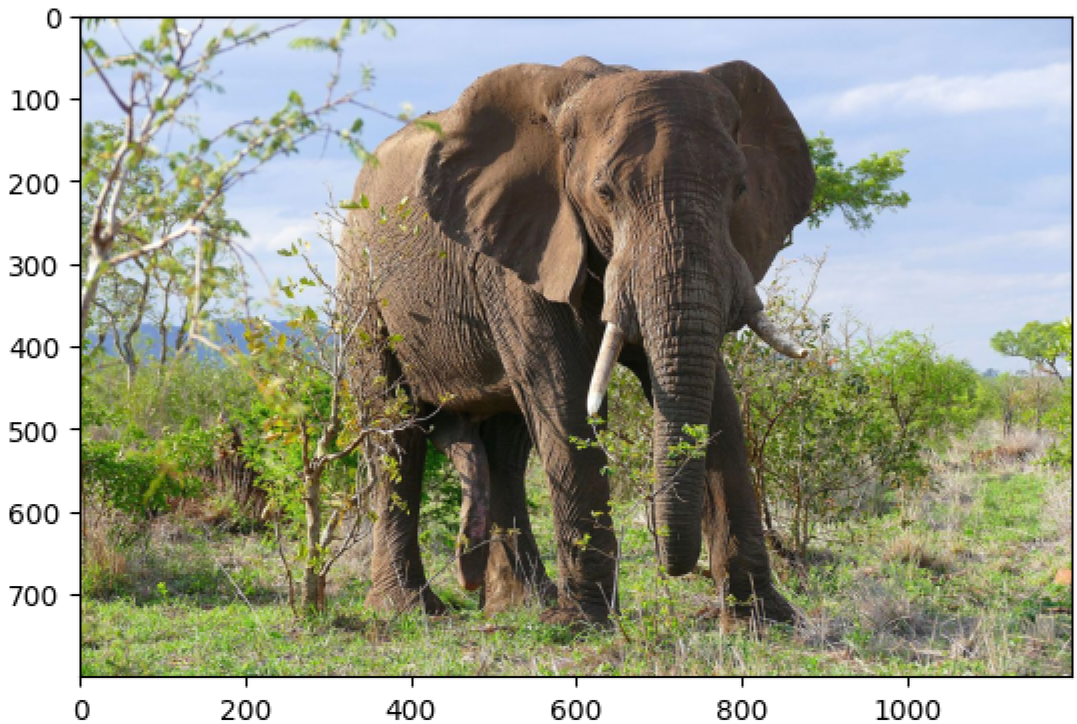
In [8]:
```
img1 = (Image.open(urlopen("https://raw.githubusercontent.com/tensorchiefs/dl_book/
plt.imshow(img1)
plt.show()
new_width  = 224
new_height = 224
# Replacing Image.ANTIALIAS with Image.Resampling.LANCZOS
img1 = img1.resize((new_width, new_height), Image.Resampling.LANCZOS)
plt.imshow(img1)
plt.show()
img1=np.array(img1)
```

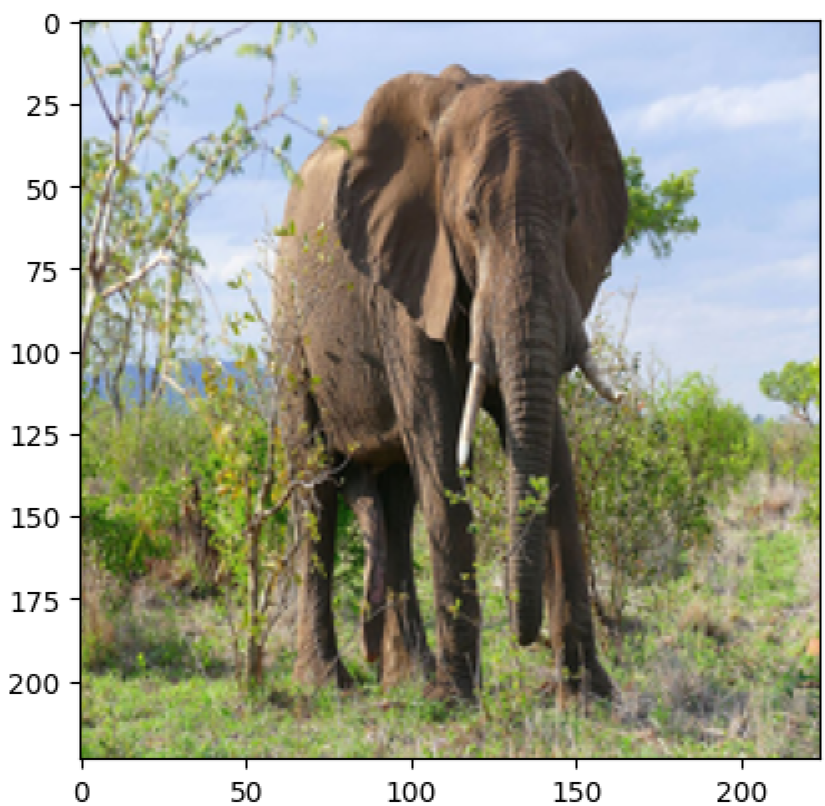




In [10]:
```
img2 = (Image.open(urlopen("https://raw.githubusercontent.com/tensorchiefs/dl_book/
plt.imshow(img2)
plt.show()
new_width  = 224
new_height = 224
```

```
# Replacing Image.ANTIALIAS with Image.Resampling.LANCZOS
img2 = img2.resize((new_width, new_height), Image.Resampling.LANCZOS)
plt.imshow(img2)
plt.show()
img2=np.array(img2)
```

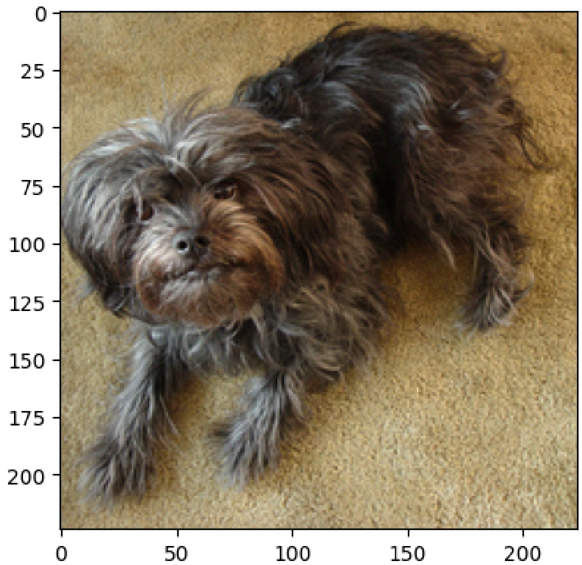


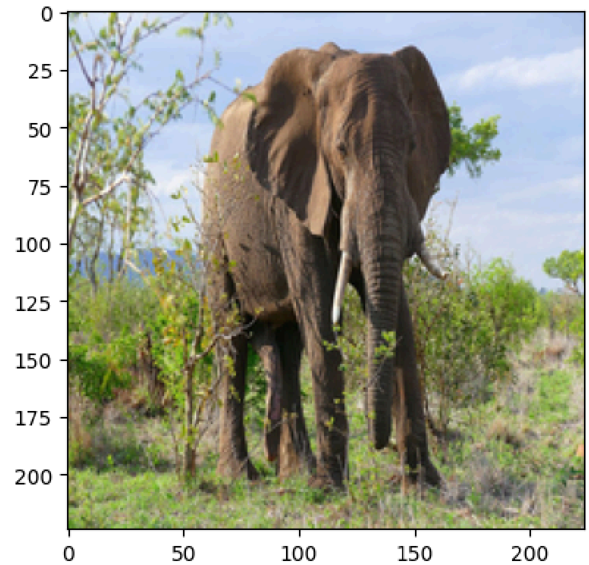


In [11]:
```
plt.figure(figsize=(10,10))
plt.subplot(1,2,1)
```

```
plt.imshow(img1)
plt.subplot(1,2,2)
plt.imshow(img2)
```

Out[11]: <matplotlib.image.AxesImage at 0x7e0130197a00>





In [12]:
```
img1=preprocess_input(img1)
print(img1.shape)
img2=preprocess_input(img2)
print(img2.shape)
```

```
(224, 224, 3)
(224, 224, 3)
```

In [13]:
```
pred1=model_vgg.predict(np.expand_dims(img1,axis=0))
tf.keras.applications.vgg16.decode_predictions(pred1)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 947ms/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/im
agenet_class_index.json
35363/35363 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Out[13]:
```
[[('n02110627', 'affenpinscher', 0.60996807),
  ('n02094433', 'Yorkshire_terrier', 0.115058616),
  ('n02097474', 'Tibetan_terrier', 0.06349397),
  ('n02086240', 'Shih-Tzu', 0.053632747),
  ('n02098413', 'Lhasa', 0.03696351)]]
```

In [14]:
```
pred2=model_vgg.predict(np.expand_dims(img2,axis=0))
tf.keras.applications.vgg16.decode_predictions(pred2)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 539ms/step
```

Out[14]:
```
[[('n01871265', 'tusker', 0.7851958),
  ('n02504458', 'African_elephant', 0.15363708),
  ('n02504013', 'Indian_elephant', 0.061132655),
  ('n02437312', 'Arabian_camel', 6.303232e-06),
  ('n02109047', 'Great_Dane', 5.9501795e-06)]]
```