In [1]:
```
#Name:Priyanshu
#Roll No:60
```

In [ ]:
```
""""
Object detection using Transfer Learning of CNN architectures
a. Load in a pre-trained CNN model trained on a large dataset
b. Freeze parameters (weights) in model's lower convolutional layers
c. Add custom classifier with several layers of trainable parameters to model
d. Train classifier layers on training data available for task
e. Fine-tune hyper parameters and unfreeze more layers as needed
""""
```

In [1]:
```
# example of using a pre-trained model as a classifier
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from keras.applications.vgg16 import VGG16
```

In [2]:
```
image = load_img('C:\\Users\\hp\\Downloads\\dog.jpg', target_size=(224, 224))
```

In [3]:
```
# convert the image pixels to a numpy array
image = img_to_array(image)
```

In [4]:
```
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

In [5]:
```
# prepare the image for the VGG model
image = preprocess_input(image)
```

In [6]:
```
# load the model
model = VGG16()
```

In [7]:
```
# predict the probability across all output classes
yhat = model.predict(image)
```

```
1/1 [==============================] - 1s 1s/step
```

In [8]:
```
# convert the probabilities to class labels
label = decode_predictions(yhat)
```

In [9]:
```
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
```

In [10]:
```
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
standard_poodle (81.69%)
```

In [12]:
```
# load an image from file
image = load_img('C:\\Users\\hp\\Downloads\\download2.png', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
model = VGG16()
e probability across all output classes
```

Loading [MathJax]/extensions/Safe.js

```
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
1/1 [==============================] - 1s 812ms/step
valley (44.85%)
```

In [13]:

```
# load an image from file
image = load_img('C:\\Users\\hp\\Downloads\\download.jpg', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
model = VGG16()
# predict the probability across all output classes
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
1/1 [==============================] - 1s 864ms/step
castle (34.03%)
```

In [ ]: