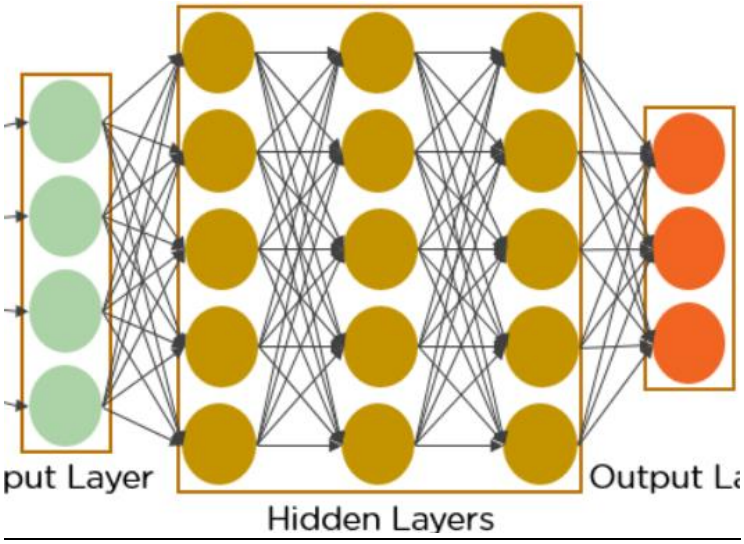
BDA Lab 08

- Jainil Trivedi (CE166)

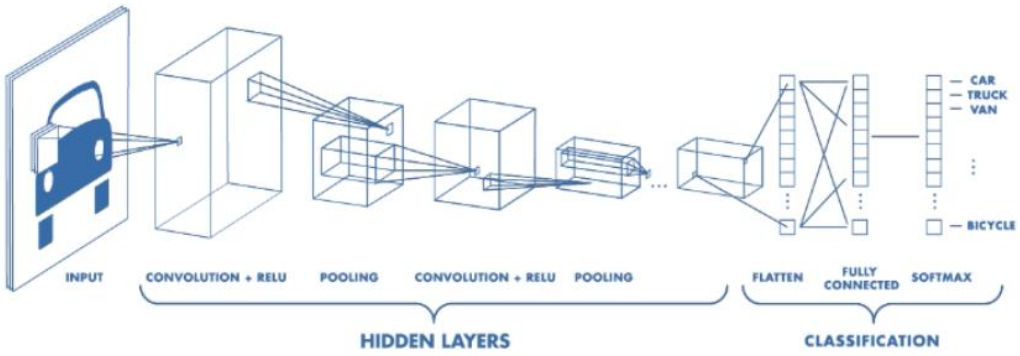
➢ **Aim:** "Leveraging machine learning to classify image."

In this lab, I have tried to classify Images of husky and golden retriever using CNN. It is a deep learning technique, specially used in detecting images.

Structure and Working of CNN:



The input layer(green) is where we give dataset to the model. In the hidden layer, we perform convolution, max pooling and flattening. Finally, this output is sent to the output layer(orange).



In [2]:
```
#using CNN classifing DOG image into husky or retriever
import numpy as np
import cv2
import os
import random
import matplotlib.pyplot as plt
import time
```

In [3]:
```
DIRECTORY = r'C:\Users\Jainil\Desktop\Jainil\College\Sem-7\BDA\L8'
CATEGORIES = ['GoldenRetriever','Husky']
IMG_SIZE = 150
```

In [4]:
```
#convert image into array and save it in a list

data = []

for category in CATEGORIES:
    folder = os.path.join(DIRECTORY,category)
    label = CATEGORIES.index(category)
    #print(folder)
    for img in os.listdir(folder):       # listdir will list all files present in folde
        img_path = os.path.join(folder,img)
        img_arr = cv2.imread(img_path)   #read image and convert into array
        img_arr = cv2.resize(img_arr,(IMG_SIZE,IMG_SIZE))

        data.append([img_arr,label])
```

In [5]:
```
# len(data)
```

In [6]:
```
# data[12]
```

In [7]:
```
random.shuffle(data)
```

In [8]:
```
X = []
y = []

for features,labels in data:
    X.append(features)
    y.append(labels)
```

In [9]:
```
X = np.array(X)
y = np.array(y)
```

In [10]:
```
print(str(len(X))+'\n'+str(len(y)))
```

```
30
```

```
30
```

In [11]:
```
y
```

Out[11]:
```
array([1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 0, 1, 0])
```

In [12]:
```
X = X/255
# X
"""
    Lesser the values contained in the array representation
    of the image easier the calculation
"""
X.shape
```

Out[12]:
```
(30, 150, 150, 3)
```

In [13]:
```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Flatten, Dense
from tensorflow.keras.callbacks import TensorBoard
# Dense layers are just regular layers
```

In [14]:
```
NAME = f'reeriever-husky-{int(time.time())}'
tensorboard = TensorBoard(log_dir = f'logs\\{NAME}\\')
```

In [32]:
```
model = Sequential()

#HIDDEN LAYER
model.add(Conv2D(64,(3,3),activation='relu'))
"""
number of features to be detected or convolution layers,
feature detector size( matrix )
activation function (softmax,sigmoid,relu) -> generally relu works best in this case
"""
model.add(MaxPool2D((2,2)))

model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPool2D((2,2)))

model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPool2D((2,2)))

#model.add(Conv2D(64,(3,3),activation='relu'))
#model.add(MaxPool2D((2,2)))

model.add(Flatten())
model.add(Dense(128,input_shape=X.shape[1:]))
model.add(Dense(128))
#128 neurons in hidden layer,shape of input image


#OUTPUT LAYER
model.add(Dense(2,activation='softmax'))
```

In [33]:
```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accurac

#adam optimizer is the goto optimizer for most cases
model.fit(X,y,epochs=10,validation_split=0.1,callbacks=tensorboard)
```

```
Epoch 1/10
1/1 [==============================] - 1s 793ms/step - loss: 0.6886 - accuracy: 0.5185 -
val_loss: 7.2512 - val_accuracy: 0.3333
Epoch 2/10
1/1 [==============================] - 0s 488ms/step - loss: 4.5682 - accuracy: 0.5185 -
val_loss: 0.6526 - val_accuracy: 0.3333
Epoch 3/10
1/1 [==============================] - 0s 449ms/step - loss: 0.5592 - accuracy: 0.6667 -
val_loss: 0.8276 - val_accuracy: 0.6667
Epoch 4/10
1/1 [==============================] - 0s 480ms/step - loss: 1.2247 - accuracy: 0.4815 -
val_loss: 0.5777 - val_accuracy: 0.6667
Epoch 5/10
1/1 [==============================] - 0s 473ms/step - loss: 0.6349 - accuracy: 0.5185 -
val_loss: 1.7714 - val_accuracy: 0.3333
Epoch 6/10
1/1 [==============================] - 0s 450ms/step - loss: 1.0493 - accuracy: 0.5185 -
val_loss: 1.0155 - val_accuracy: 0.3333
Epoch 7/10
1/1 [==============================] - 0s 483ms/step - loss: 0.6797 - accuracy: 0.5556 -
val_loss: 0.5527 - val_accuracy: 1.0000
Epoch 8/10
1/1 [==============================] - 0s 448ms/step - loss: 0.5584 - accuracy: 0.8519 -
val_loss: 0.4980 - val_accuracy: 0.6667
Epoch 9/10
1/1 [==============================] - 0s 441ms/step - loss: 0.6226 - accuracy: 0.4815 -
val_loss: 0.4957 - val_accuracy: 0.6667
Epoch 10/10
1/1 [==============================] - 0s 433ms/step - loss: 0.5602 - accuracy: 0.6667 -
val_loss: 0.6005 - val_accuracy: 1.0000
```

Out[33]:
```
<keras.callbacks.History at 0x140043b6070>
```

In [ ]:
```

```