TASK FOR PROJECT

NOTE:- SELECT A DATASET FOR YOUR PROJECT FROM KAGGLE OR CHOOSE ONE TAT SULITS YOUR NEEDS.

CAT AND DOGS IMAGES CLASSIFIER:-

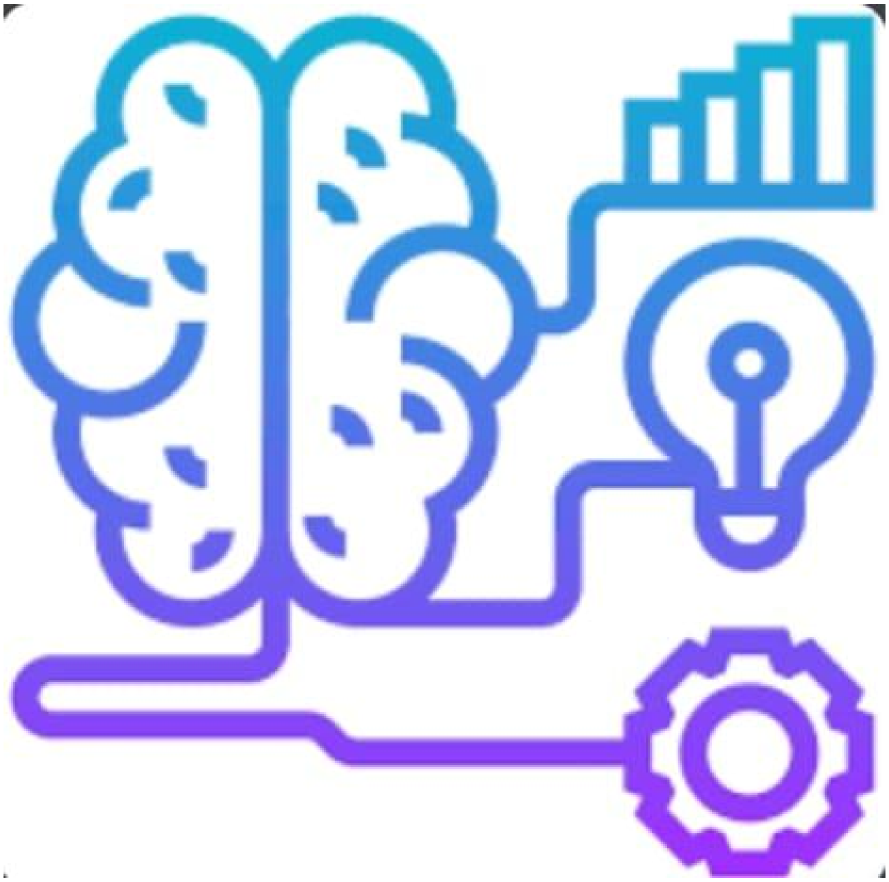
DEVELOP AN IMAGES CLASSIFICATIONS MODAL TO DISTINGUISH BETWEEN IMAGES OF CATS AND DOGS USING

DATA SCIENCE TECHINQUES IN PYTHON

DOGS VS CAT:-

TRANSFRE LEARNING:-

Deep learning technique where we use a per-trained modal. this per-trained modal is trained for on one task and can be re-trained for a similar task wait a smaller dataset.



Trans

fer learning gives higher accuracy compared training modal from scratch.

EXAMPLES OF PER-TRAINED MODALS:-

*VGG-16

*INCEPTIONV3

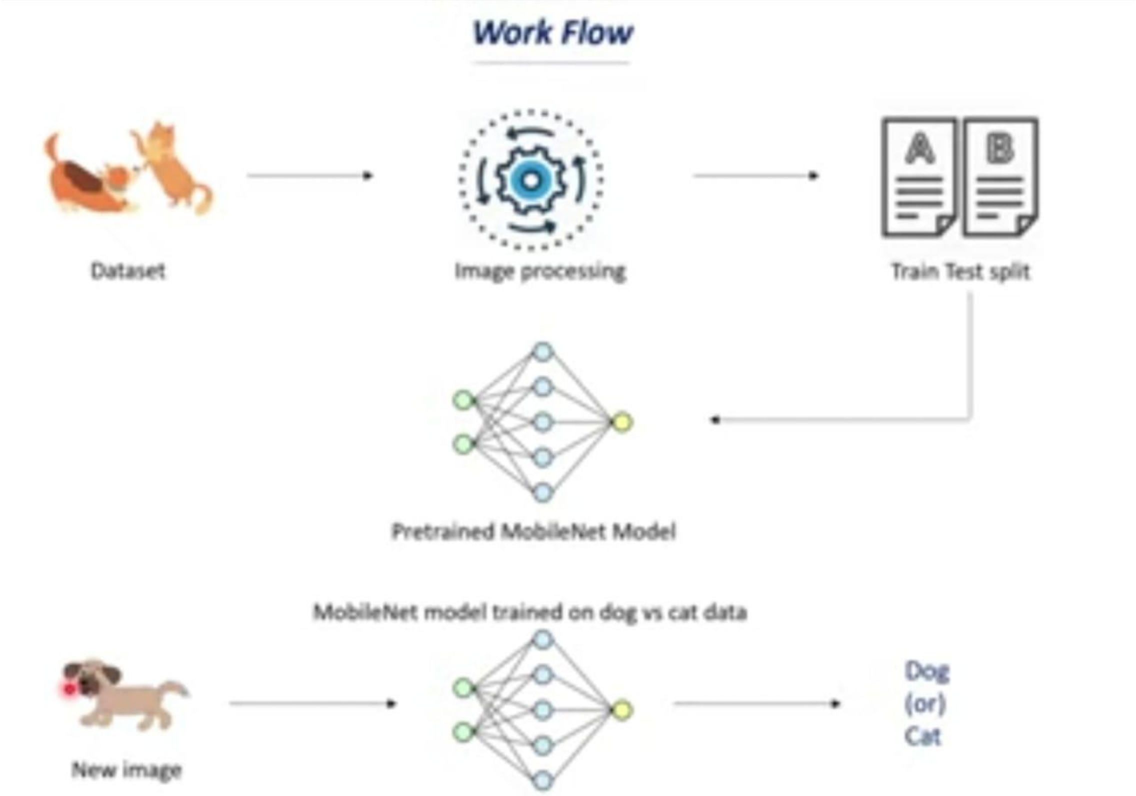
*RESNET50

*MOBILENETV2

WORKFLOW:-

DATASET----->IMAGE PROCESSING------>TRAIN TEST SPLIT

PRETRAINED MOBILE NET MODAL<------



MOBILE NET MODAL TRAINED ON DOG VS CAT DATA

NET IMAGE------>IMAGE------->DOG OR CAT

EXTRACTING DATASET USING KAGGLE APL:-

(1) #Installing the kaggle libary.

```
l pip install kaggle
```

(2) #Configuring the path of kaggle. json file

```
Image mkdir-p~/.kaggle.

icp Kaggle .json~/.kaggle/

i chord 600~/.kaggle. json
```

IMPORTING THE DOG VS CAT DATASET FORM KAGGLE:-

(3) #Kaggle api

```
 lkaggle competition download-dogs-vs-cats.
```

RESULT:-

```
Downloading dogs-vs-cats-zip to /content
90% 793M/812M[00.22<00:00,21.5MB/S]
100% 812M/812M[00.22<00:00,38.2MB/S]
```

(4) LIS

```
dogs-vs-cats.zip kaggle. Json  sample_ data
```

(5) #Enteracting the compressed dataset.

```
from zip file import zip file
with zip file (data set, 'r')as zip:
zip. Extract()
print('the dataset is extracted')
```

THE DATASET IS EXTRACTED:-

(6) #Extracting the compressed dataset.

```
from zip file import zip file.
dataset='/content/train.zip'
with zip file (data set, 'r')as zip:
zip. Extract()
print('the dataset is extracted')
```

THE DATASET IS EXTRACCED:-

(7) Import OS

```
#Counting the number of files in train folder.
 path, dirs. ,files=next(OS. walk('/content/train'))
```

```
     files-count I=lean(files)
     print('number of images: ',file-count)
     number of images:2500
```

PRINTING THE NAME OF IMAGES:-

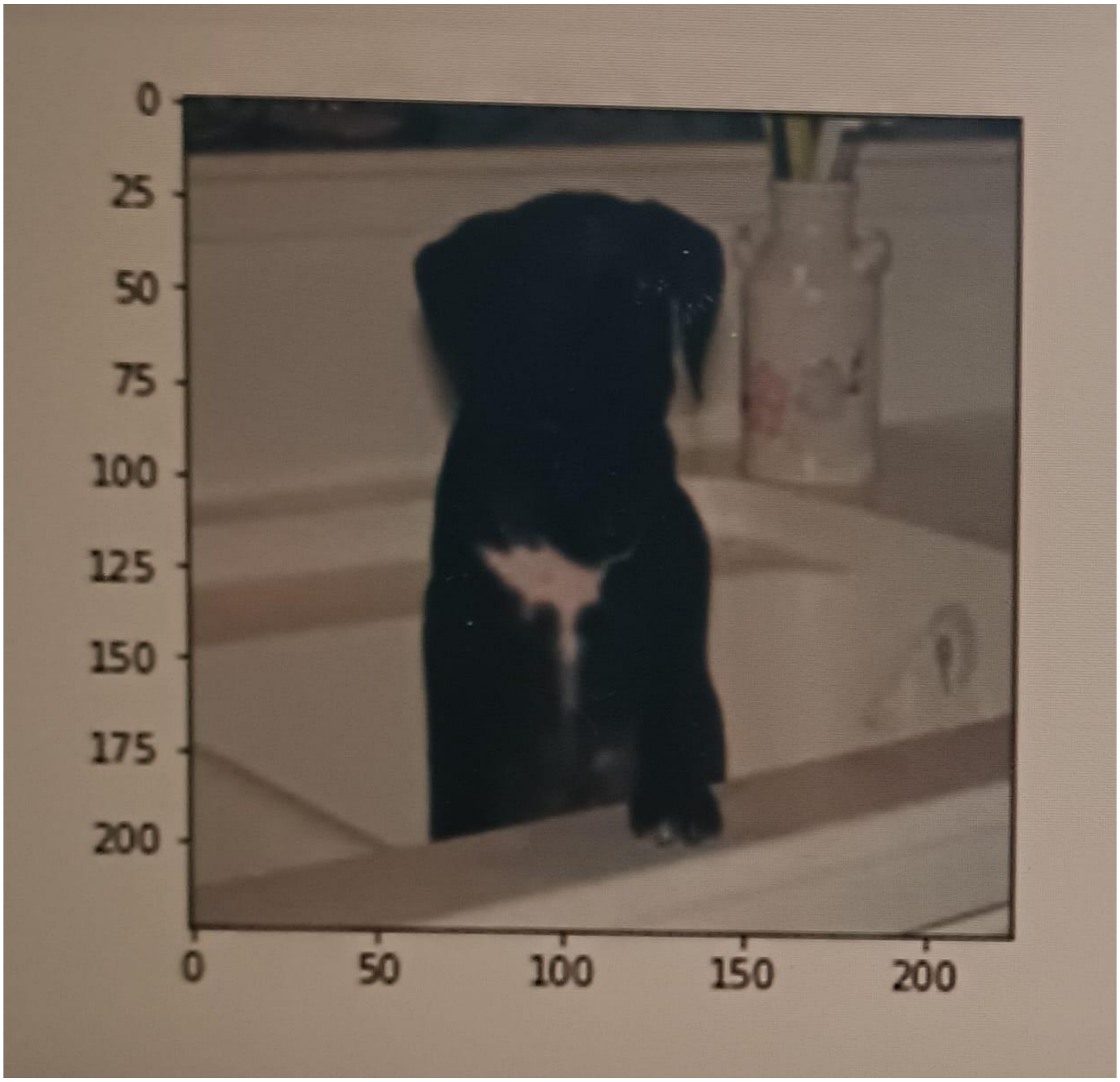
```
(8) Files-names=OS. listdir('/count/train/')
    print(file-names)
   ['dog.12067.jpg','cat.1189.jpg','cat.7561.jpg','cat.7760.jpg','cat.7076.jpg','dog.8298.jpg',
    'cat.4352.jpg', 'cat.1873.jpg','cat.7969.jpg;,;dog.3687.jpg',]
```

IMPORTING THE DEPENDENCIES:-

```
(9) Import numpy as np
    from PIL import image
    import matplotlib .pyplot as plt
    import matplotlib. image as mage ping
    from sklearn .modal _selection import train_ test_ split
    from google. colab. patches import CV2_imshow
```

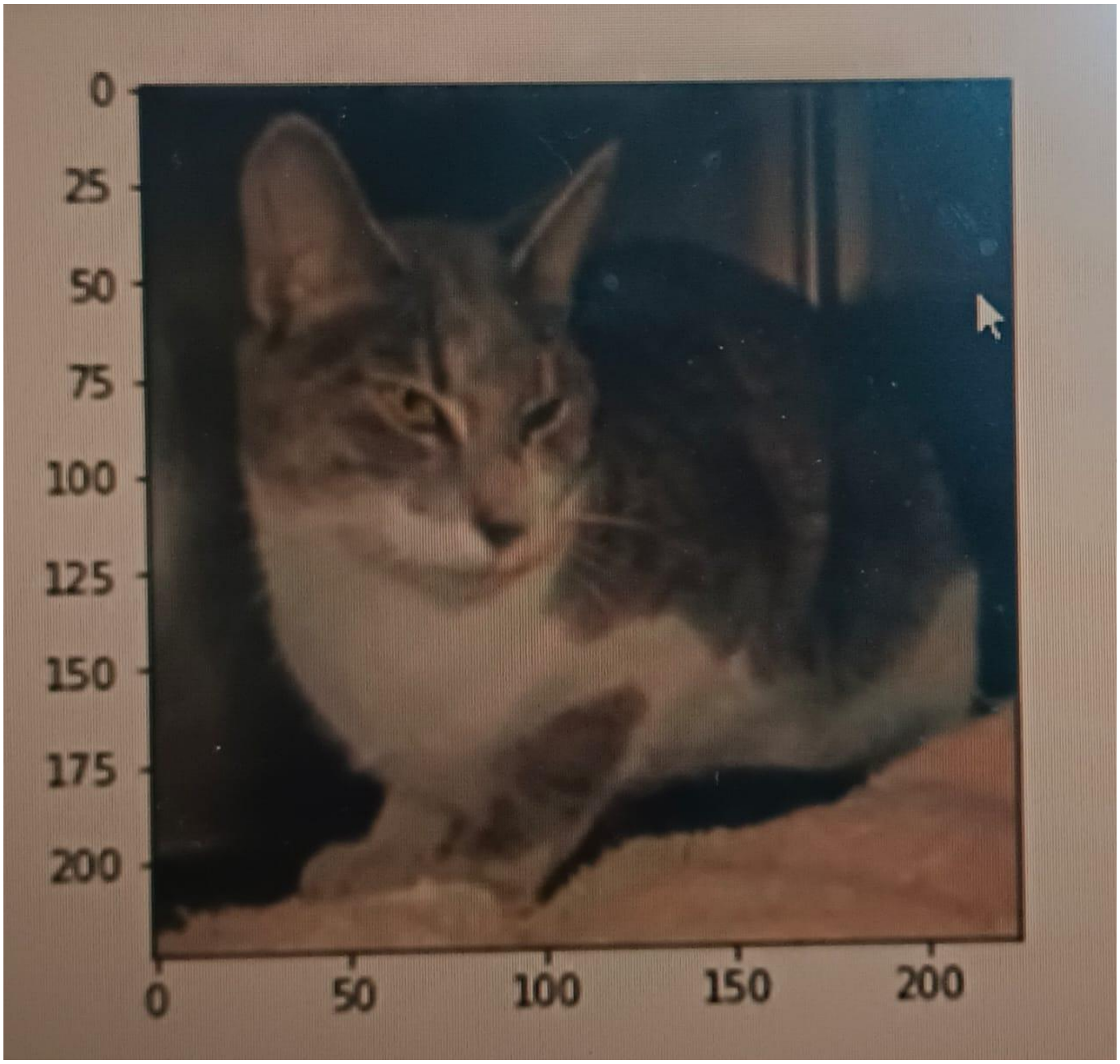
DISPLAYING THE IMAGES OF DOGS AND CAT:-

```
(10) #disply dog image
      Image =mping. Image read('/context/train/dog.8298.jpg')
      image plt=plt .show(image)
      plt ,show()
```



(11) #Disply dog image

```
Image =noping .image read('/context/train/cat.4352.jpg')

Image pit=pit. Image show(image)

pit. show()
```



```
(12) File_ names=0S.listdir('/content/train/')
     for in range (5):
     name=file_ names[i]
     print(name[0.3])
```

```
RESULT:-DOG
        CAT
        CAT
        CAT
```

CAT

```
(13) File _names=06.listdir('/content/train/')
     dog_ count=0
     cat_ count=0
     for image file in file_ name:
     name=image_ file[0.3]
     if name=='dog',:
     dog_ count+=1
     else:
     cat. count+=1
     print('number of dog images=',dog_ count)
     print('number of cat images=',cat_ count)
```

RESULT:-Number of dog images=12500

Number of cat images=12500

RESIZING ALL THE IMAGES:-

```
(14) #Creating a directory for resized images
      OS. mkdir ('/content/images resized')
```

```
(15) orginal_ folder='/content/train/'
     Realized _ folder='/content/image resized/'
     for in range(2000):
     filename=0S.listdir(original_ folder)[i]
     image _path=original _folder +filename
     image=[image. open[image . path]]
     image =image .realize((224,224))
     image=image .convert('RGB')
     new image path=resized _folder +filename
```

```
    image. save(new image path)
```

```
(16) #Disply resized dog image
     img=mping.im read('/content/image resized/dog.8298.jpg')
     image plt =plt.im show (image)
     plt. show()
```

```
(17) #Disply cat image
     Image =mping. imread('/content/train/cat.4352.jpg')
     imgplt =plt. imshow(image)
     plt show ()
```

CREATING LABELS FOR REAIZED IMAGES OF DOGS AND CATS:-

Cat-->0

Dog-->1

```
(18) #Creating a for to assign labels
     filenames=OS. listdir('/count/image/reaized/')
     labels=[]
     for i in range(2000):
     file_ names=filenames[i]
     label=file_ name[0,3]
     else:
     labels. appened(0)
```

```
RESULT:-Print(filename[0,5])
        Print (len(filename))

       ['dog.1267.jpg','cat.1189.jpg','cat.7561.jpg','cat.7760.jpg','cat.7076.jpg']

       2000.
```

```
RESULT:-Print (lables[0.5])
        Print(len  (lables))
        [1,0,0,0,0]
         2000.
```

```
(19) #Counting the images of dogs and cats out of 2000 images.
     values, counts=np .unique(lables,  return_ counts=true)
     print (values)
     print (counts)
```

```
RESULT:-[0,1]
       [992,1008]
```

CONVERTING ALL THE REALIZED IMAGES TO NUMPY ARRAYS:-

```
(20) Import CV2
     Import glob
```

```
(21)Images_ directly='/content/images resized/'
    images_ extension=['png  ',jpg']
    file=[]
    [files .extend(glob. glob(image_ directly + '*', +e))for e in image_ extension]
    dog_ cat_ images = np .asarray ([CV2.imread(file)for file in files])
```

```
RESULT:-[[[[ 79 93 142]
          [ 67 79 127]
          [ 77 87 135]
          ...
         [121 113 113]
         [130 117 119]
         [129 116 118]

       [[ 43 54 106]
        [ 45 56 106]
        [ 72 79 128]
        ...
       [121 113 113]
       [131 119 119]
       [131 119 119]
[[ 64  71  126]
 [ 77  85  138]
 [103 119  160]
 ...
 [120 113 110]
 [131 119 117]
 [132 120 118]]
  ...
[[ 73 79  98]
 [ 73 79  98]
 [ 76 82 101]
  ...
 [ 121  99  88]
 [ 115  93  81]
 [ 130 108  96]]

[[ 59 65  84]
 [ 66 72  91]
 [ 80 86 105]
  ...
 [ 121 101  90]
 [ 134 115 102]
 [ 175 156 143]]

[[ 84 90 109]
 [ 83 89 108]
 [ 83 89 108]
  ...
[ 118  98 109]
[ 149 130 117]
[ 212 193 180]]]
```

```
  [[[  1 248 228]
    [ 13 255 237]
    [ 20 251 242]
     ...
    [ 8 177 235]
    [ 2 174 232]
    [ 0 172 229]]
```

```
(22) 1 type(dog_ cat_ images)
       numps. ndarray
```

```
(23) 1 print(dog_ cat_ images. shapes)
       (2000,224,224,3)
```

```
(24) X=DOG_CAT_ images.
     Y=np .ass array (lables)
```

```
(25) X_Train,X-Test,Y_Train,Y_Test=Train_Test_split(X,Y,Test_Size=0.2,Random_state=2)
```

```
(26) Print(X,  Shape, X_ train. shape, X_ test. shape)
     (2000,224,224,3) (1600,224,224,3) (400,224,224,3)
     1600-->training images
      400-->test images
```

```
(27) #Scolling the data
      X_ train_ scaled=X_ train/225
      X_ test_ scaled=X_ test/225
```

```
(28) print(X_TRAIN_SCALED)
```

```
RESULT:-[[0.36862745 0.4627451  0.50980392]

         [0.4745098  0.56862745 0.61568627]

         [0.48627451 0.58039216 0.62745098]

         ...

         [0.34117647 0.41176471 0.43921569]

         [0.35686275 0.42754098 0.43921569]

         [0.33333333 0.40392157 0.43137255]]]


      [[[0.01960784  0.02745098 0.02745098]

        [0.01960784  0.02745098 0.02745098]

        [0.01960784  0.02745098 0.02745098]

        ...

        [0.00784314  0.03137255 0.08235294]

        [0.01568627  0.03529412 0.02745098]

        [0.01960784  0.03529412 0.10588235]]


      [[0.01960784   0.02745098 0.02745098]

       [0.01960784   0.02745098 0.02745098]

       [0.01960784   0.02745098 0.02745098]

       ...

       [0.00784314   0.03137255 0.08235294]

       [0.01960784   0.03529412 0.09411765]

       [0.01960784   0.03529412 0.10588235]]


     [[0.01568627    0.02352941 0.02352941]

      [0.01568627    0.02352941 0.02352941]

      [0.01568627    0.02352941 0.02352941]

      ...

      [0.01176471    0.03529412 0.08627451]

      [0.00784314    0.03529412 0.09627451]

      [0.01176471    0.03921569 0.09803922]]
```

```
[[0.16862745    0.28627451 0.42352941]
 [0.18039216    0.29803922 0.43529412]
 [0.19607843    0.31372549 0.45098039]
 ...
[0.83529412     0.85490196 0.85098039]
[0.84313725     0.8627451  0.85882353]
[0.84705882     0.86666667 0.8627451 ]]
```

BULINDING THE NETURAL NETWORK:-

```
(29) import tensor flow as it
     import flow_ hub as hub
```

```
(30) mobilent _modal=-------
     pvetraind_ modal=hub. keraslayer (mobilenet_ modal, input_
shape=(224,224,3),trainable=false)
```

```
(31) number _of_ classes=2
     modal=tf .keras .sequential l[
     prentained_ modal,
     tf. keras. layer. dense (number_ of_ classes)
     ])
     Modal .summary()
```

MOBILENETV2 ARCHITECTURE:-

```
(32) Modal: "sequential"
     ________________________________________________________________________
     Layer (type)            Output shape        Param #
     ========================================================================
     Keras_ layer (Kera salayer)    (None, 1280)       2257984


     dense (Dense)               (None, 2)          2562


     ==============================================================================
     Total params: 2,60,546
     Trainable params: 2,562
     Non-trainable params: 2,257,984
```

```
(33) Modal .complie (
     optimizer='adom',
     loss _tf. Keras .losses. sparse categorical crossen tropy (from_ logits=true),
     mareic =['acc']
```

```
(34) Modal. fit(X_ train_ scaled, Y train, epochs=5)


    Epoch 1/5
    50/50 [==============================] - 47s 861ms/step - loss: 0.2163 - acc: 0. 9162
    Epoch 2/5
    50/50 [==============================] - 42s 838ms/step - loss: 0.0746 - acc: 0. 9756
    Epoch 3/5
    50/50 [==============================] - 44s 872ms/step - loss: 0.0532 - acc: 0. 9825
    Epoch 4/5
    50/50 [==============================] - 41s 824ms/step - loss: 0.0417 - acc: 0. 9894
    Epoch 5/5
    50/50 [==============================] - 41s 825ms/step - loss: 0.0345 - acc: 0. 9937
    <Kereas. callbacks. history at 0x7faedc598090>
```

```
(35) Score, acc= modal. Evalute (X_ test_ scaled, y_ test)
     print('test loss=',score)
     print('test accuracy=', acc)
```

```
  13/13[============] - 12s 866ms/step - loss: 0.0812 - acc: 0.9775

  Test loss = 0,0812455490231514

  Test Accuracy = 0.9775000214576721
```

PREDICTIVE SYSTEM:-

```
(36) input_ image_ path = input('path of the image to be predicated:')

     input_ image = cv2.imread(input_ image _path)

     cv2_imshow(input_ image)

     input_ image_ resize = cv2.resize(input_ image,(224,224))

     input_ image-scaled = input_ image_ resize/225

     image_ resaped = np. reshape(input_ image_ scaled,[1,224,224,3])

     input_ predication = modal. predict(image_ reshaped)

     input_ pred_ label = np .argmax(input_ predication)

     if input_ pred_ label == 0:

       print('The image represents a cat')

     else

       print('The image represents a dog')
```

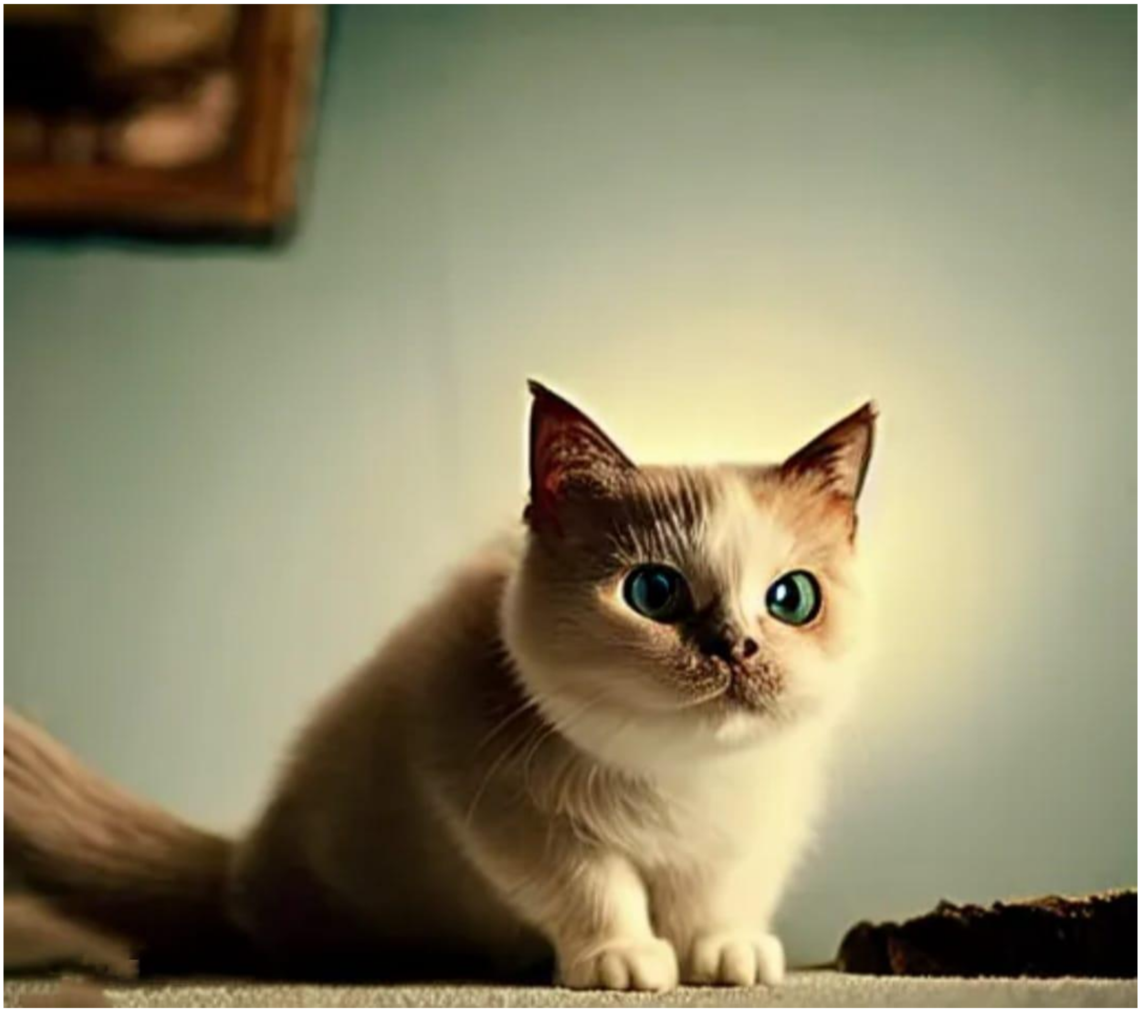
```
 path of the image to be predicated: /content/dog.jpg
```

```
 [[-4.6012597 3.784018 ]]

 1

The image represents a dog
```

```
(40) input_ image_ path = input('path of the image to be predicated:')
     input_ image = cv2.imread(input_ image_ path)
     cv2_imshow(input_ image)
     input_ image_ resize = cv2.resize(input_ image, (224,224)
     input_ image_ scaled + input_ image_ resize/225
     image_ reshaped = np .reshape(input_ image_ scaled, [1,224,224,3])
     input_ predication = modal .predict(image_ reshaped)
     print(input_ predication)
     input_ pred_ label = np. argmax(input_ predication)
     print(input_ pred_ label)
     if input_ pred_ label == 0:
       print('The image represents a Cat')
     else:
```

```
    print('The image represents a Dog')
```

```
Path of the image to be predicted:/content/cat.jpg
```



```
[[ 4.302739 -4.893738]]

0

The image represents a cat.
```