Assignment No 3

Aim :- Build the image classification model by dividing the model into 4 stages
a] Loading image data
b] Defining model's architecture
c] Training the model
d] Estimate model's performance

Objectives :- To learn about CNN & how to develop a CNN for image recognition

Infrastructure : Computer / Laptop

Software used :- Jupyter Notebook / Google Colab

Theory :-

CNN

Convolution networks, also known as convolution neural networks or CNN's are a specialized kind of neural network for processing data that has a known, grid-like topology. The name "convolution neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a special kind of linear operation. Convolution networks are simply neural networks that use convolution in place of general matrix multiplication in atleast one of their layers

There are two main parts to a CNN archi
· A convolution tool that seperates and identifi
the various features of that image for analysis
in a process called as Feature Extraction.
· The network of feature extraction consists of ma
pairs of convolutional or pooling layers
· A fully connected layer that utilizes the output fro
the convolutional process & predicts the class of
image based on feature extracted in previous s
· This CNN model of feature extraction aims to r
the number of features present in a dataset. It c
new features which summaries the existing fe
contained in an original set of features.

Convolution Layers

· There are three types of layers that make up
CNN. which are convolution layers, pooling lay
& fully connected layers.
· In addition to these three layers there are two m
important parameters which are dropout layer &
activation function.

1] Convolution layer :-

This is the first layer that is used to extract
various features from input images. In this layer,
mathematical operations are performed between i
image & a filter. By slider over the input image, th
dot product is taken between the filters and pa
input image with respect to size of filter.

· The output is termed as the Feature map which gives us information about image such as corners & edges.

2] Pooling Layers
The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers & independently operates on each feature map. In MaxPooling, the largest element is taken from feature map. Average pooling calculates the average of elements in predefinied sized image.

3] Fully Connected Layer
The fully connected layer consists of weights & biases along with neuron & is used to connect the neurons between two different layers. These layers are usually placed before the output layer & form the last few layers of an CNN architecture

4.] Dropout :-
Usually, when all features are connected to FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on training data causing a negative impact in model's performance when used on a new data.

5.] Activation Function
They are used to learn & approximate any of continuous & complex relationship between variables of network. In simple words, it decides which information of the model should fire in forward direction & which ones should not at end of network

Implementation
1.] Load necessary libraries
2.] Import dataset from respective library
3.] Design neural network architecture & mention number of layers, etc.
4.] The train the model with imported dataset
5.] Evaluate performance of model.

Conclusion:-
We learnt how to build & train a CNN to iden images.

In [1]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from tensorflow.keras.optimizers import SGD
import numpy as np
import random
import matplotlib.pyplot as plt
import tensorflow as tf
```

In [2]:
```
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
```

Load Dataset

Dataset avalaible on https://bit.ly/ImgClsKeras (https://bit.ly/ImgClsKeras)

In [3]:
```
x_train = np.loadtxt('input.csv',delimiter = ',')
y_train = np.loadtxt('labels.csv',delimiter = ',')

x_test = np.loadtxt('input_test.csv',delimiter = ',')
y_test = np.loadtxt('labels_test.csv',delimiter = ',')
```

In [4]:
```
x_train = x_train.reshape(len(x_train),100,100,3)
y_train = y_train.reshape(len(y_train),1)

x_test = x_test.reshape(len(x_test),100,100,3)
y_test = y_test.reshape(len(y_test),1)

x_train = x_train/255.0
x_test = x_test/255.0
```

In [5]:
```
print("Shape of X_train:", x_train.shape)
print("Shape of Y_train:", y_train.shape)
print("Shape of X_train:", x_test.shape)
print("Shape of X_train:", y_test.shape)
```

```
Shape of X_train: (2000, 100, 100, 3)
Shape of Y_train: (2000, 1)
Shape of X_train: (400, 100, 100, 3)
Shape of X_train: (400, 1)
```

In [6]:
```
idx = random.randint(0,len(x_train))
plt.imshow(x_train[idx,:])
plt.show()
```

Model Building

In [8]:
```
model = Sequential([
    Conv2D(256,(3,3),activation = 'relu',input_shape=(100,100,3)),
    BatchNormalization(),
    MaxPooling2D((4,4)),
    Conv2D(128,(3,3),activation = 'relu'),
    BatchNormalization(),
    MaxPooling2D((2,2)),
    Conv2D(64,(3,3),activation = 'relu'),
    BatchNormalization(),
    MaxPooling2D((2,2)),

    Flatten(),
    Dense(128,activation='relu'),
    Dropout(0.4),
    Dense(1,activation='sigmoid')
])
```

In [9]:
```
opt = SGD(momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics = ['accuracy'])
```

```
In [10]: model.fit(x_train,y_train,epochs=30,steps_per_epoch = 20,validation_data=(x_test,

         Epoch 1/30
         20/20 [==============================] - 61s 3s/step - loss: 0.8490 - accuracy:
         0.5650 - val_loss: 0.6948 - val_accuracy: 0.5250
         Epoch 2/30
         20/20 [==============================] - 70s 4s/step - loss: 0.6606 - accuracy:
         0.6240 - val_loss: 0.6801 - val_accuracy: 0.5975
         Epoch 3/30
         20/20 [==============================] - 68s 3s/step - loss: 0.6057 - accuracy:
         0.6830 - val_loss: 0.6797 - val_accuracy: 0.5625
         Epoch 4/30
         20/20 [==============================] - 64s 3s/step - loss: 0.5674 - accuracy:
         0.6990 - val_loss: 0.7254 - val_accuracy: 0.5050
         Epoch 5/30
         20/20 [==============================] - 64s 3s/step - loss: 0.5089 - accuracy:
         0.7605 - val_loss: 0.7999 - val_accuracy: 0.5050
         Epoch 6/30
         20/20 [==============================] - 62s 3s/step - loss: 0.4879 - accuracy:
         0.7720 - val_loss: 0.8164 - val_accuracy: 0.4950
         Epoch 7/30
         20/20 [==============================] - 61s 3s/step - loss: 0.4430 - accuracy:
         0.7960 - val_loss: 0.8590 - val_accuracy: 0.5125
         Epoch 8/30
         20/20 [==============================] - 60s 3s/step - loss: 0.4141 - accuracy:
         0.8130 - val_loss: 0.8978 - val_accuracy: 0.5050
         Epoch 9/30
         20/20 [==============================] - 65s 3s/step - loss: 0.3859 - accuracy:
         0.8270 - val_loss: 1.0060 - val_accuracy: 0.5025
         Epoch 10/30
         20/20 [==============================] - 59s 3s/step - loss: 0.3405 - accuracy:
         0.8470 - val_loss: 1.0907 - val_accuracy: 0.5200
         Epoch 11/30
         20/20 [==============================] - 59s 3s/step - loss: 0.3248 - accuracy:
         0.8555 - val_loss: 0.9742 - val_accuracy: 0.5300
         Epoch 12/30
         20/20 [==============================] - 59s 3s/step - loss: 0.2908 - accuracy:
         0.8820 - val_loss: 0.8911 - val_accuracy: 0.5650
         Epoch 13/30
         20/20 [==============================] - 59s 3s/step - loss: 0.2615 - accuracy:
         0.8920 - val_loss: 0.6965 - val_accuracy: 0.6225
         Epoch 14/30
         20/20 [==============================] - 59s 3s/step - loss: 0.2234 - accuracy:
         0.9145 - val_loss: 0.7834 - val_accuracy: 0.6000
         Epoch 15/30
         20/20 [==============================] - 60s 3s/step - loss: 0.1961 - accuracy:
         0.9245 - val_loss: 1.3311 - val_accuracy: 0.5475
         Epoch 16/30
         20/20 [==============================] - 60s 3s/step - loss: 0.1657 - accuracy:
         0.9470 - val_loss: 1.0894 - val_accuracy: 0.5900
         Epoch 17/30
         20/20 [==============================] - 60s 3s/step - loss: 0.1325 - accuracy:
         0.9590 - val_loss: 1.3612 - val_accuracy: 0.5525
         Epoch 18/30
         20/20 [==============================] - 60s 3s/step - loss: 0.1255 - accuracy:
         0.9565 - val_loss: 1.0850 - val_accuracy: 0.6050
         Epoch 19/30
```

```
20/20 [==============================] - 60s 3s/step - loss: 0.1015 - accuracy:
0.9620 - val_loss: 1.1502 - val_accuracy: 0.5850
Epoch 20/30
20/20 [==============================] - 63s 3s/step - loss: 0.0923 - accuracy:
0.9660 - val_loss: 1.0095 - val_accuracy: 0.6275
Epoch 21/30
20/20 [==============================] - 61s 3s/step - loss: 0.0725 - accuracy:
0.9770 - val_loss: 1.1029 - val_accuracy: 0.6150
Epoch 22/30
20/20 [==============================] - 60s 3s/step - loss: 0.0776 - accuracy:
0.9725 - val_loss: 0.7184 - val_accuracy: 0.7275
Epoch 23/30
20/20 [==============================] - 61s 3s/step - loss: 0.1011 - accuracy:
0.9610 - val_loss: 0.9714 - val_accuracy: 0.6525
Epoch 24/30
20/20 [==============================] - 61s 3s/step - loss: 0.0831 - accuracy:
0.9705 - val_loss: 0.9042 - val_accuracy: 0.6775
Epoch 25/30
20/20 [==============================] - 61s 3s/step - loss: 0.0670 - accuracy:
0.9760 - val_loss: 1.0374 - val_accuracy: 0.6825
Epoch 26/30
20/20 [==============================] - 61s 3s/step - loss: 0.0483 - accuracy:
0.9835 - val_loss: 0.8355 - val_accuracy: 0.7250
Epoch 27/30
20/20 [==============================] - 61s 3s/step - loss: 0.0448 - accuracy:
0.9860 - val_loss: 1.0373 - val_accuracy: 0.7000
Epoch 28/30
20/20 [==============================] - 61s 3s/step - loss: 0.0401 - accuracy:
0.9870 - val_loss: 0.8931 - val_accuracy: 0.7225
Epoch 29/30
20/20 [==============================] - 61s 3s/step - loss: 0.0376 - accuracy:
0.9900 - val_loss: 1.1096 - val_accuracy: 0.7050
Epoch 30/30
20/20 [==============================] - 61s 3s/step - loss: 0.0272 - accuracy:
0.9940 - val_loss: 1.0204 - val_accuracy: 0.7100
```

```
Out[10]: <keras.callbacks.History at 0x1c39aeacee0>
```

```
In [11]: model.evaluate(x_test,y_test)
```

```
13/13 [==============================] - 2s 164ms/step - loss: 1.0204 - accurac
y: 0.7100
```

```
Out[11]: [1.0204460620880127, 0.7099999785423279]
```

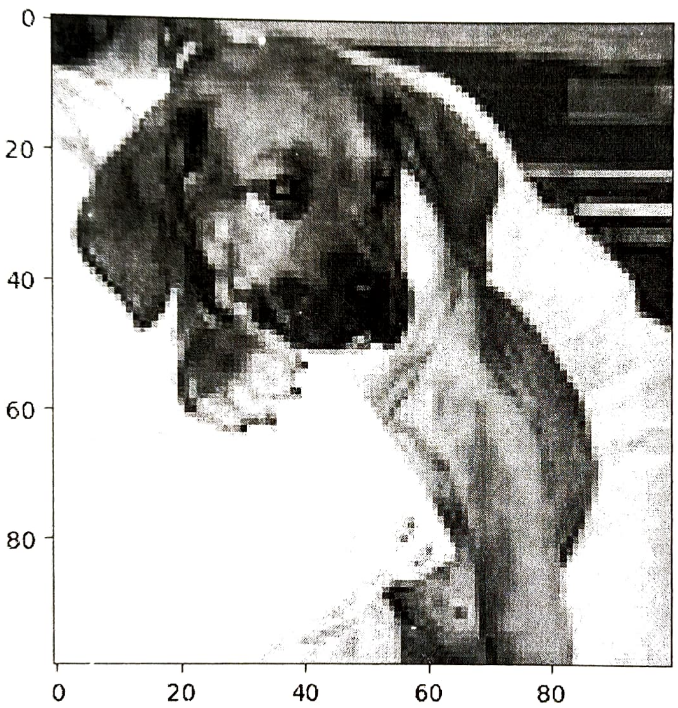
Making Predictions

In [14]:
```
idx2 = random.randint(0,len(y_test))
plt.imshow(x_test[idx2,:])
plt.show()

y_pred = model.predict(x_test[idx2,:].reshape(1,100,100,3))
y_pred = y_pred > 0.5

if(y_pred==0):
    pred = 'dog'
else:
    pred = 'cat'

print("Our model says it is a",pred)
```




```
1/1 [==============================] - 0s 28ms/step
Our model says it is a dog
```

In [15]:
```
score = model.evaluate(x_test, y_test, verbose = 0 )
print("Test Score: ", score[0])
print("Test accuracy: ", score[1])
```

```
Test Score:  1.0204460620880127
Test accuracy:  0.7099999785423279
```

In [16]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 256)       7168

 batch_normalization (BatchN  (None, 98, 98, 256)      1024
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 24, 24, 256)      0
 )

 conv2d_1 (Conv2D)           (None, 22, 22, 128)       295040

 batch_normalization_1 (Batc  (None, 22, 22, 128)      512
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 11, 11, 128)      0
 2D)

 conv2d_2 (Conv2D)           (None, 9, 9, 64)          73792

 batch_normalization_2 (Batc  (None, 9, 9, 64)         256
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 128)               131200

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 509,121
Trainable params: 508,225
Non-trainable params: 896
_________________________________________________________________
```

In [17]:
```
val = model.fit(x_train,y_train, epochs=5,validation_data=(x_test,y_test),batch_:
```

```
Epoch 1/5
10/10 [==============================] - 61s 6s/step - loss: 0.0246 - accuracy:
0.9925 - val_loss: 0.9134 - val_accuracy: 0.7250
Epoch 2/5
10/10 [==============================] - 59s 6s/step - loss: 0.0173 - accuracy:
0.9970 - val_loss: 0.8590 - val_accuracy: 0.7275
Epoch 3/5
10/10 [==============================] - 58s 6s/step - loss: 0.0139 - accuracy:
0.9970 - val_loss: 0.8365 - val_accuracy: 0.7425
Epoch 4/5
10/10 [==============================] - 59s 6s/step - loss: 0.0101 - accuracy:
0.9995 - val_loss: 0.8777 - val_accuracy: 0.7425
Epoch 5/5
10/10 [==============================] - 59s 6s/step - loss: 0.0116 - accuracy:
0.9975 - val_loss: 0.8869 - val_accuracy: 0.7650
```

In [18]:
```
plt.title("Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel('epoch')
plt.plot(val.history['accuracy'])
plt.plot(val.history['val_accuracy'])
plt.legend(['train','val'])
plt.show()
```