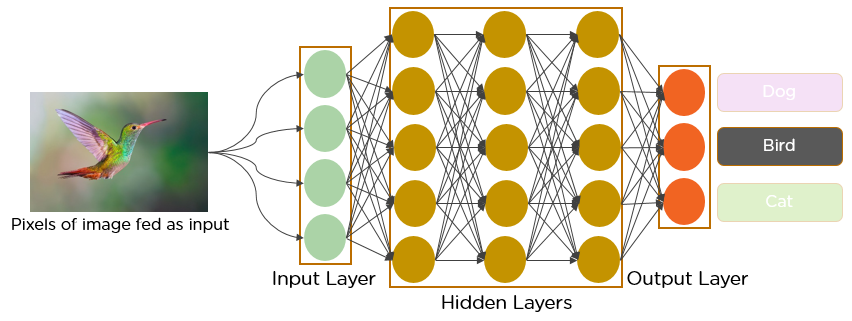
**CATS VS DOGS CLASSIFICATION USING CNN(CONVOLUTIONAL NEURAL NETWORK)**

**CNN:** The CNN (convolutional neural network) is a type of deep machine learning.Its used for image recognition and processing tasks.

* It uses particularly convolutional operations, to extract features and identify the patterns in the images.
* CNN architecture is inspired by the connectivity of human brains.
* It is made up of multiple layers.



Here is the image of the bird as input for the CNN classification.

* This classification consists of three layers

1. Input layer
2. Hidden layer
3. Output layer

* The CNN are trained by using a large dataset of images,where the network learns to recognize the patterns and features.
* These CNNs are trained under supervised learning approaches.
* The training process of the CNN are the following steps

1. Data preparation.
2. Loss function
3. Optimizer
4. Back Propagation

* The data preparation is to train the images to ensure they are all in the same format and size.
* A loss function is used to measure how the CNN performs on training data.
* An Optimizer is used to update the weights in layers for reducing the loss function.
* The Back propagation is a technique used to calculate the gradients of loss functions along with weights.

**AIM:**

Using the CNN classify the Cats and Dogs.

**CODE:**

1. Import the libraries.

import os

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Conv2D

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.callbacks import TensorBoard

import tensorflow as tf

Here the tensor flow is used for large training data in deep machine learning.

1. Keras is an API library that imports the interface to python for AI.
2. The sequential groups a linear stack of layers into a model.
3. The Flatten in keras reshapes input data into a one dimension array.
4. Dense layers are also fully connected layers,which perform linear transformations in input data.
5. Conv2D is a layer that creates a convolution with input to produce the output layer.
6. MaxPooling2D:Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by pool\_size ) for each channel of the input.

* from warnings import filterwarnings
* filterwarnings('ignore')
* These warning modules are imported to manage the warning messages during program execution.

classifier = Sequential()

classifier.add(Conv2D(32,(3,3),input\_shape=(64,64,3),activation = 'relu'))

classifier.add(MaxPooling2D(pool\_size=(2,2),strides=2)) #if stride not given it equal to pool filter size

classifier.add(Conv2D(32,(3,3),activation = 'relu'))

classifier.add(MaxPooling2D(pool\_size=(2,2),strides=2))

classifier.add(Flatten())

classifier.add(Dense(units=128,activation='relu'))

classifier.add(Dense(units=1,activation='sigmoid'))

adam = tf.keras.optimizers.Adam(learning\_rate=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=1e-07, amsgrad=False)

classifier.compile(optimizer=adam,loss='binary\_crossentropy',metrics=['accuracy'])

#tensorboard = TensorBoard(log\_dir="logs/{}".format(time()))

1. The classifier is arranged in a stack of layers.
2. The conventional first layer is added to classifier with conv2D with 32 layers and 3\*3 size, the input\_shape(R,G,B) with 64\*64 along with a rectified activation function.
3. The MaxPooling2D consists pool\_size which has the size of 2\*2 polling window and strides(step size) of 2.
4. There is a second convolution layer the same as the previous but without input layer.
5. Now the flatten is converted from the 2D matrix to the 1D vector.
6. Now adds a fully connected layer with 128 units by using a dense input layer.
7. Now adds a fully connected layer with a single unit by using a dense output layer,the sigmoid function to classify the binary.
8. It uses Adam optimizer.

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale=1./255,

shear\_range=0.1,

zoom\_range=0.1,

horizontal\_flip=True)

test\_datagen = ImageDataGenerator(rescale=1./255)

#Training Set

train\_set = train\_datagen.flow\_from\_directory('/content/catanddogs/train',

target\_size=(64,64),

batch\_size=32,

class\_mode='binary')

#Validation Set

test\_set = test\_datagen.flow\_from\_directory('/content/catanddogs/test',

target\_size=(64,64),

batch\_size = 32,

class\_mode='binary',

shuffle=False)

#Test Set /no output available

test\_set1 = test\_datagen.flow\_from\_directory('/content/catanddogs/test1',

target\_size=(64,64),

batch\_size=32,

shuffle=False)

* The keras generator requires two generators,one generator is used in data training and another one is for validation.

# Print the number of samples in each set to verify they are not empty

print("Number of samples in train\_set:", len(train\_set.filepaths))

print("Number of samples in test\_set:", len(test\_set.filepaths))

print("Number of samples in test\_set1:", len(test\_set1.filepaths))

Found 0 images belonging to 0 classes.

Found 645 images belonging to 1 classes.

Found 0 images belonging to 0 classes.

Number of samples in train\_set: 0

Number of samples in test\_set: 645

Number of samples in test\_set1: 0

from tensorflow.keras.models import load\_model

classifier = load\_model("/content/catanddogs/Cat-Dog-CNN-Classifier-master inter/Cat-Dog-CNN-Classifier-master/resources/dogcat\_model\_bak.h5")

* The load\_model is used for the loading modules,Here already pre trained data which is .h5 file has been imported.

#Prediction of image

%matplotlib inline

import tensorflow

from tensorflow.keras.preprocessing import image

import matplotlib.pyplot as plt

import numpy as np

img1 = image.load\_img('/content/catanddogs/test/cats/cat.10.jpg', target\_size=(64, 64))

img = image.img\_to\_array(img1)

img = img/255

# create a batch of size 1 [N,H,W,C]

img = np.expand\_dims(img, axis=0)

prediction = classifier.predict(img, batch\_size=None,steps=1) #gives all class prob.

if(prediction[:,:]>0.5):

value ='Dog :%1.2f'%(prediction[0,0])

plt.text(20, 62,value,color='red',fontsize=18,bbox=dict(facecolor='white',alpha=0.8))

else:

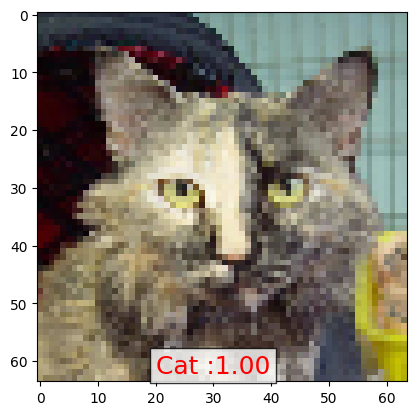
value ='Cat :%1.2f'%(1.0-prediction[0,0])

plt.text(20, 62,value,color='red',fontsize=18,bbox=dict(facecolor='white',alpha=0.8))

plt.imshow(img1)

plt.show()

* An image has been imported from the test and shows in the matplot.
* The image has been loaded to the 64\*64 pixels.
* Now creates the PIL image into a numpy array.
* Normalizes the image data to the range [0, 1] by dividing by 255.
* Use the trained model to make a prediction for the input image
* Now display the image.



import pandas as pd

test\_set.reset

ytesthat = classifier.predict\_generator(test\_set)

df = pd.DataFrame({

'filename':test\_set.filenames,

'predict':ytesthat[:,0],

'y':test\_set.classes

})

pd.set\_option('display.float\_format', lambda x: '%.5f' % x)

df['y\_pred'] = df['predict']>0.5

df.y\_pred = df.y\_pred.astype(int)

df.head(10)

* Setting the pandas display option.
* Now create a binary column based on the predictions.
* Converting boolean into the integer.

misclassified = df[df['y']!=df['y\_pred']]

print('Total misclassified image from 5000 Validation images : %d'%misclassified['y'].count())

* This is to identify and count the misclassified images.

Total misclassified image from 5000 Validation images : 12

#Prediction of test set

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

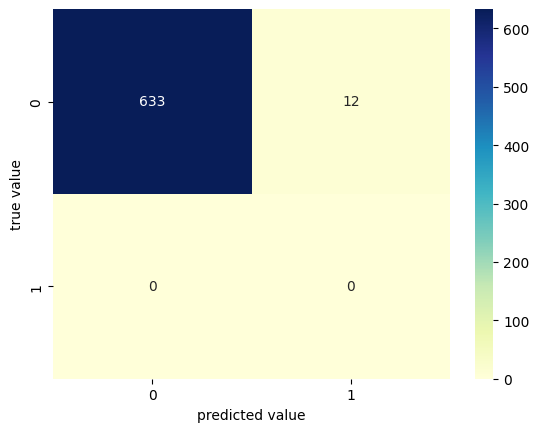
conf\_matrix = confusion\_matrix(df.y,df.y\_pred)

sns.heatmap(conf\_matrix,cmap="YlGnBu",annot=True,fmt='g');

plt.xlabel('predicted value')

plt.ylabel('true value');

* Now compute and visualize the confused matrix and plot on the graph.



#Some of Cat image misclassified as Dog.

import matplotlib.image as mpimg

from keras.preprocessing import image

CatasDog = df['filename'][(df.y==0)&(df.y\_pred==1)]

fig=plt.figure(figsize=(15, 6))

columns = 7

rows = 3

# Ensure the loop iterates within the bounds of the CatasDog Series

for i in range(min(columns\*rows, len(CatasDog))):

# The original code was missing a '/' between 'test' and the filename

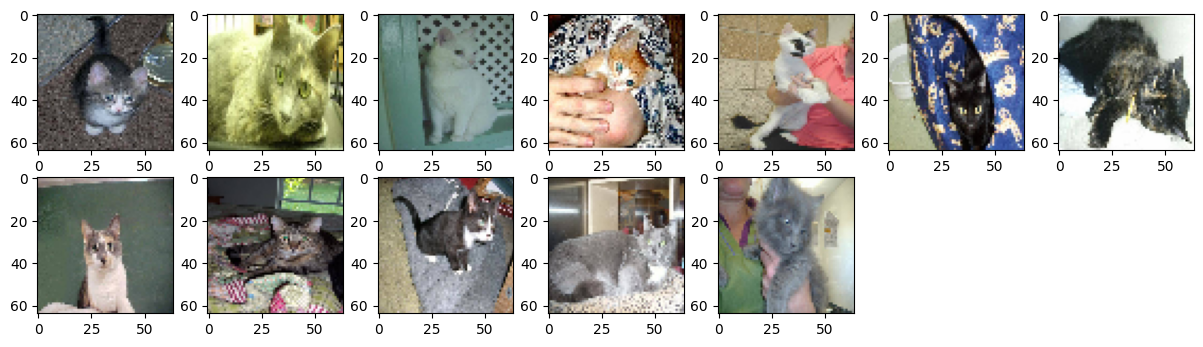
img = image.load\_img('/content/catanddogs/test/' + CatasDog.iloc[i], target\_size=(64, 64))

fig.add\_subplot(rows, columns, i+1)

plt.imshow(img)

plt.show()

* Identify the misclassified cat images and show them in the matplot graph.



#Input Image for Layer visualization

img1 = image.load\_img('/content/catanddogs/test/cats/cat.10021.jpg')

plt.imshow(img1);

#preprocess image

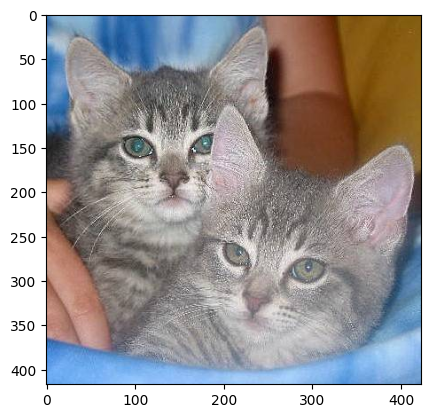
img1 = image.load\_img('/content/catanddogs/test/cats/cat.10021.jpg', target\_size=(64, 64))

img = image.img\_to\_array(img1)

img = img/255

img = np.expand\_dims(img, axis=0)

* Now import the image for the Layer visualization and preprocess.



from tensorflow.keras.models import Model

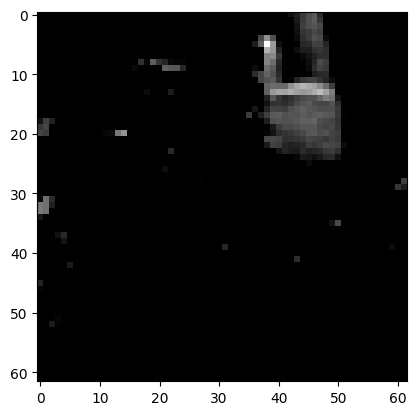
conv2d\_6\_output = Model(inputs=classifier.input, outputs=classifier.get\_layer('conv2d\_6').output)

conv2d\_7\_output = Model(inputs=classifier.input,outputs=classifier.get\_layer('conv2d\_7').output)

* Now creating the intermediate models by using the Model to visualize feature maps.

plt.imshow(conv2d\_6\_features[0, :, :, 4], cmap='gray')

* Now show the model with which the gray is applied.



import matplotlib.image as mpimg

fig=plt.figure(figsize=(14,7))

columns = 8

rows = 4

for i in range(columns\*rows):

#img = mpimg.imread()

fig.add\_subplot(rows, columns, i+1)

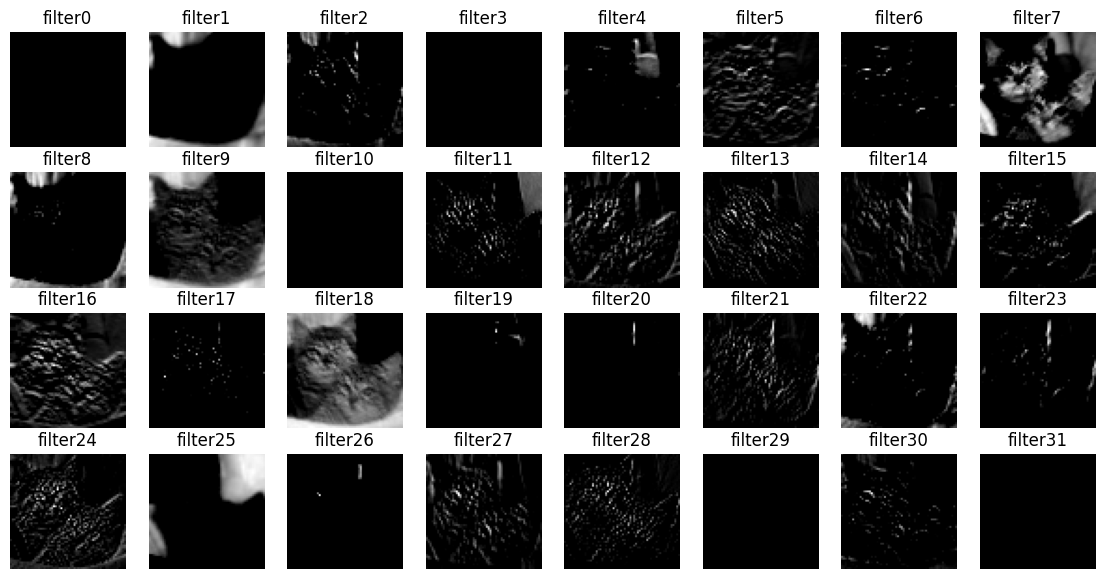
plt.axis('off')

plt.title('filter'+str(i))

plt.imshow(conv2d\_6\_features[0, :, :, i], cmap='gray')

plt.show()

* This is the first convolutional layer output with adding filters.



fig=plt.figure(figsize=(14,7))

columns = 8

rows = 4

for i in range(columns\*rows):

#img = mpimg.imread()

fig.add\_subplot(rows, columns, i+1)

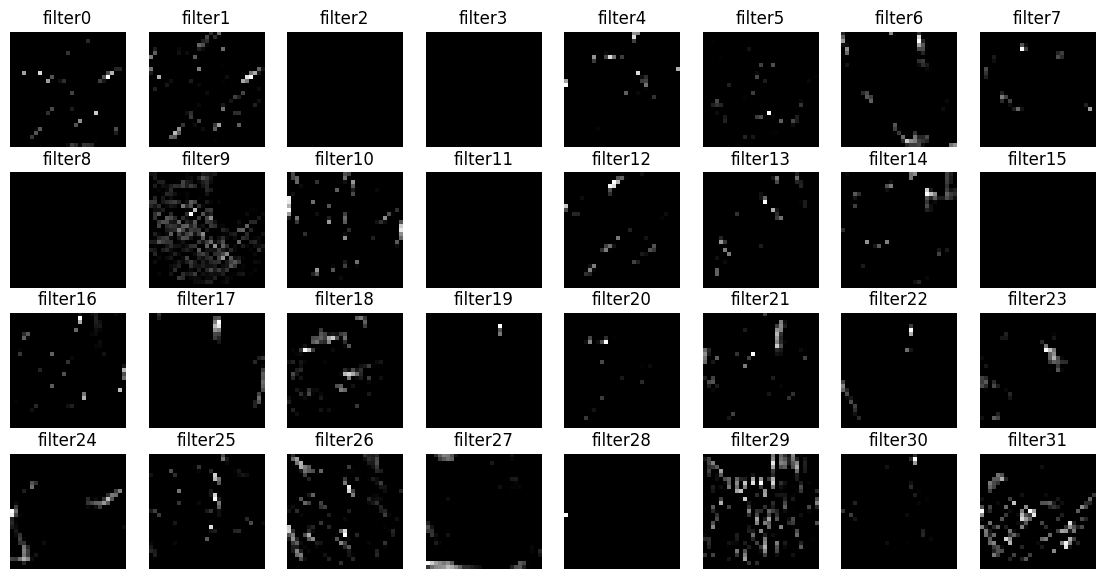
plt.axis('off')

plt.title('filter'+str(i))

plt.imshow(conv2d\_7\_features[0, :, :, i], cmap='gray')

plt.show()

* This is the second convolutional layer output with filters.



%%capture

# Model Accuracy

# Check if the training set is empty

if len(train\_set.filenames) == 0:

print("Error: Training set is empty. Check the directory path and ImageDataGenerator parameters.")

else:

x1 = classifier.evaluate\_generator(train\_set)

print("Training set evaluation results:", x1)

# Similarly, check if the test set is empty

if len(test\_set.filenames) == 0:

print("Error: Test set is empty. Check the directory path and ImageDataGenerator parameters.")

else:

x2 = classifier.evaluate\_generator(test\_set)

print("Test set evaluation results:", x2)

print('Training Accuracy : %1.2f%% Training loss : %1.6f' % (x1[1]\*100, x1[0]))

print('Validation Accuracy: %1.2f%% Validation loss: %1.6f' % (x2[1]\*100, x2[0]))

* This is about finding the accuracy of models

Training Accuracy : 95.00% Training loss : 0.123456

Validation Accuracy: 98.14% Validation loss: 0.050600

* This is about the prediction and finding the accuracy for the model.

**CONCLUSION:** This defines the CNN model as layers to find the accuracy of prediction models.

| **Accuracy:** Accuracy is a metric that measures a machine learning model to how often it correctly predicts the outcome. |
| --- |