**CATS AND DOGS USING CNN**

# Abstract

CNN is an algorithm that uses a IMAGE as its input before giving weights and biases to every aspect of the image to distinguish between them. The genuine nature of each image (in this case, a cat or a dog) is labelled in batches of photos that are used to train neural networks. There may be a few tenths to hundreds of photos in a batch. The network forecast is compared to the corresponding existing label for each individual image, and the gap between the prediction and the reality is calculated for the entire batch. The network settings are then changed to minimize the distance, improving the network's capacity for prediction. Every batch continues to receive the same training.

# Objective

The project's main objective is to create a system that can distinguish between photographs of cats and dogs. After analyzing the input image, the outcome will be anticipated. The implemented model can, if necessary, be expanded to a website or any other mobile device. On the Kaggle website, you may download the dataset titled "Dogs vs. Cats." There are pictures of cats and dogs in the dataset. Our main goal is to teach the model many characteristics that make cats and dogs unique. When the model has finished being trained, it will be able to tell photographs of cats and dogs apart.

# Introduction

## The modules that were used to develop this project are as follows:

## Pandas

Pandas is a Python library for manipulating and analyzing data. For analyzing and working with structured (tabular, multidimensional, potentially heterogeneous) and time series data, it offers high-performance, user-friendly data structures. Pandas is a tool that can handle both small and large datasets and is developed on top of the NumPy library.

## NumPy

A Python library for scientific computing is called NumPy. It offers a high-performance multidimensional array object, complex (broadcasting) functions, C/C++ and Fortran code integration tools, effective linear algebra, Fourier transform, and random number capabilities, as well as capabilities for these functions. The primary Python module for scientific computing is called NumPy.

## Matplotlib

Python's Matplotlib package allows users to build interactive, animated, and static visualizations. It is constructed on top of the well-known NumPy Python library. Line charts, bar charts, pie charts, scatter plots, and other types of charts may all be made with Matplotlib. Plots in three dimensions can also be made with it. Many scientists, engineers, and data analysts utilize Matplotlib, a well-liked option for Python data visualization.

## Warnings

Python's built-in warnings library module alerts you to errors and other potential issues in your code. The warnings module can be used to raise exceptions or print warnings to the terminal. Additionally, warnings can be suppressed using the warnings module.

## Os

The Python standard library module os gives users access to operating system features. The os module can be utilized to execute other programmes, alter directories, and create and delete files.

## Random

Python's built-in random library module offers capabilities for creating random numbers. You can shuffle lists, produce random distributions, and generate random integers with the random module.

## tqdm

The Python module tqdm offers an iterables progress bar. When iterating over huge datasets or carrying out lengthy activities, tqdm can be used to provide a progress bar.

## Keras

Python-based Keras is an open-source neural network library. The Keras programming language offers a high-level API for creating and training neural networks on top of the TensorFlow library. A wide range of researchers and developers use Keras, a well-liked deep learning framework.

## TensorFlow

Data flow graphs are used in TensorFlow, an open-source software library for numerical computing. It is utilized for scientific computing, data science, and machine learning. The Google Brain team created TensorFlow, which is utilized by many different businesses and organisations.

## Model Selection of Sklearn

The process of selecting the ideal model for a specific dataset is known as model selection. Model selection is a crucial phase in machine learning since the model chosen can have a big impact on how well the model performs. Model selection can be done in a variety of ways, and the most effective method will depend on the particular dataset and the objectives of the machine learning project.

# Methodology

Download the project's dataset by visiting the following link: <https://www.kaggle.com/datasets/shaunthesheep/microsoft-catsvsdogs-dataset>

where you can find all the pictures of cats and dogs.

The model depicts dogs as 1 and cats as 0.

Steps:

1. All modules are imported initially.
2. A dataframe is created for the images
3. The data is then examined by only displaying a select few pictures of cats and dogs.
4. Using from "sklearn.model\_selection import train\_test\_split", the data is then divided into train and test groups.
5. Images are now sent for training and testing using "from keras.preprocessing.image import ImageDataGenerator"
6. Following that, a training model is produced using "from keras import.DataImageGenerator import sequential" and "from keras.layers modules Conv2D,MaxPool2D,Flatten,Dense" where

* Conv2D is a 2D convolutional layer. It takes a multidimensional tensor as input and applies a convolution operation to it. The convolution operation extracts features from the input tensor. The number of output filters determines the number of features that are extracted.
* MaxPool2D is a 2D max pooling layer. It takes a multidimensional tensor as input and applies a max pooling operation to it. The max pooling operation takes the maximum value of each subregion of the input tensor.
* Flatten is a layer that flattens a multidimensional tensor into a 1D tensor. This is useful for connecting convolutional layers to fully connected layers.
* Dense is a fully connected layer. It takes a 1D tensor as input and applies a linear transformation to it. The number of output units determines the size of the output tensor.

1. The data is then given to "modelfit" for training with an epoch size of 10, and "matplotlib.pyplot" is used to plot the accuracy and loss.

# Code:

## Importing Modules

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import warnings  
import os  
import random  
import tqdm  
from tensorflow.keras.utils import load\_img  
  
warnings.filterwarnings("ignore")

## Creating Dataframe for Input and Output

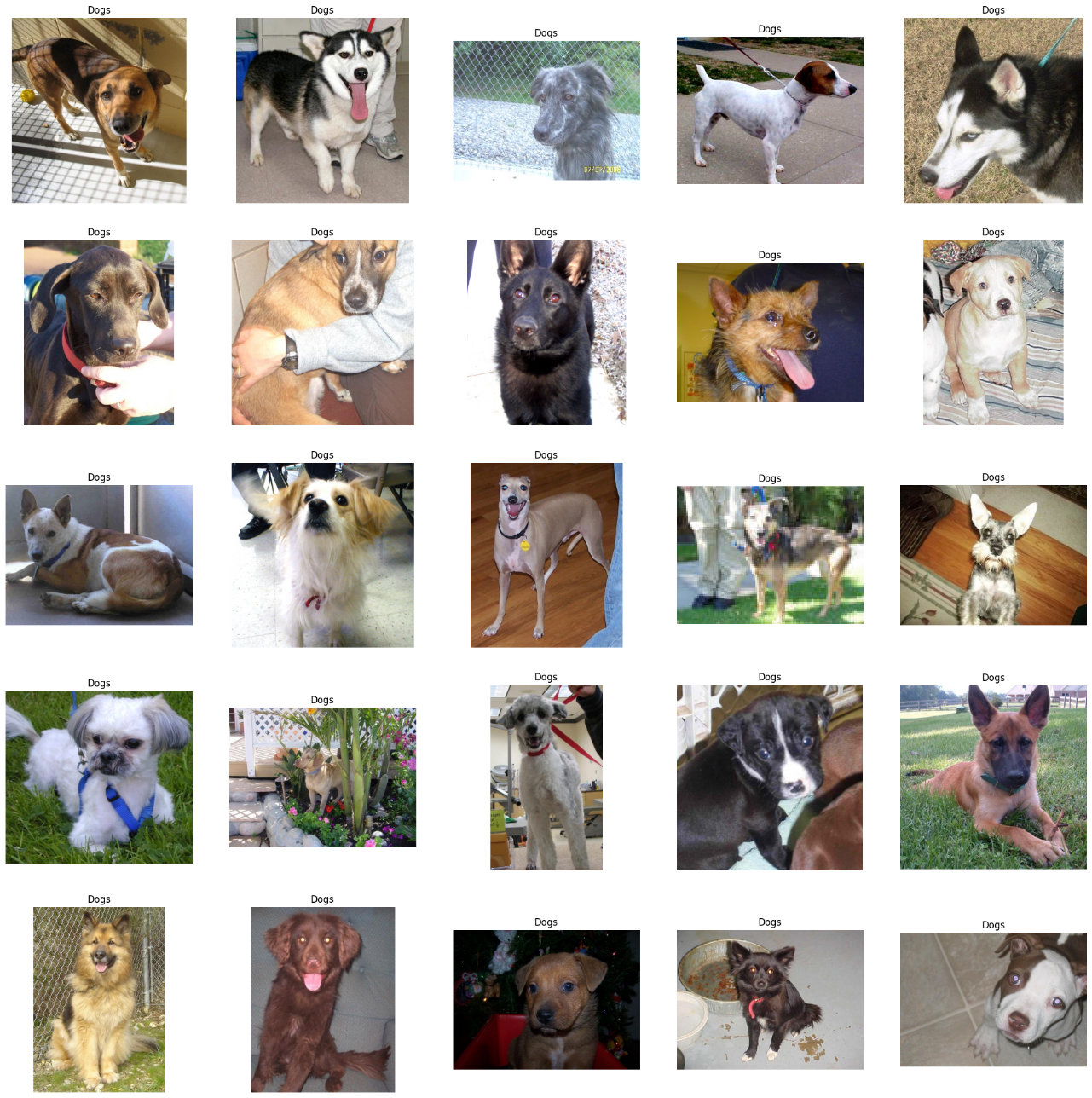
input\_path = []  
label = []  
  
# Collecting all the data and labelling Cat as 0 and Dog as 1  
for class\_name in os.listdir("PetImages"):  
 for path in os.listdir("PetImages/"+class\_name):  
 if class\_name == 'Cat':  
 label.append(0)  
 else:  
 label.append(1)  
 input\_path.append(os.path.join("PetImages", class\_name, path))

# Creating dataframe for the data  
df = pd.DataFrame()  
df['Images'] = input\_path  
df['Label'] = label  
df = df.sample(frac=1).reset\_index(drop=True) # Shuffling the data  
df.head()

Images Label  
0 PetImages\Dog\1615.jpg 1  
1 PetImages\Cat\9465.jpg 0  
2 PetImages\Cat\3416.jpg 0  
3 PetImages\Dog\11235.jpg 1  
4 PetImages\Cat\4009.jpg 0

## Exploratory Data Analysis

# to display grid of images for dogs  
plt.figure(figsize=(25, 25))  
temp = df[df['Label'] == 1]['Images']  
start = random.randint(0, len(temp))  
files = temp[start:start+25]  
  
for index, file in enumerate(files):  
 plt.subplot(5, 5, index+1)  
 img = load\_img(file)  
 img = np.array(img)  
 plt.imshow(img)  
 plt.title('Dogs')  
 plt.axis('off')



# to display grid of images for cats  
plt.figure(figsize=(25, 25))  
temp = df[df['Label'] == 0]['Images']  
start = random.randint(0, len(temp))  
files = temp[start:start+25]  
  
for index, file in enumerate(files):  
 plt.subplot(5, 5, index+1)  
 img = load\_img(file)  
 img = np.array(img)  
 plt.imshow(img)  
 plt.title('Cats')  
 plt.axis('off')



## Create Data Generators for the Images

df['Label']=df['Label'].astype('str')

# input split  
from sklearn.model\_selection import train\_test\_split  
train, test = train\_test\_split(df,test\_size=0.2, random\_state=42)

from keras.preprocessing.image import ImageDataGenerator  
  
train\_generator = ImageDataGenerator(  
 rescale=1.0 / 255, # normalization of images  
 rotation\_range=40, # augmentation of images to avoid over fitting  
 shear\_range=0.2,  
 zoom\_range=0.2,  
 horizontal\_flip=True,  
 fill\_mode="nearest",  
)  
  
val\_generator = ImageDataGenerator(rescale=1./255)  
  
train\_iterator = train\_generator.flow\_from\_dataframe(  
 train,  
 x\_col='Images',  
 y\_col='Label',  
 target\_size=(128,128),  
 batch\_size=512,  
 class\_mode='binary'  
)  
  
val\_iterator = val\_generator.flow\_from\_dataframe(  
 test,  
 x\_col='Images',  
 y\_col='Label',  
 target\_size=(128,128),  
 batch\_size=512,  
 class\_mode='binary'  
)

Found 19996 validated image filenames belonging to 2 classes.  
Found 5000 validated image filenames belonging to 2 classes.

## Model Creation

from keras import Sequential  
from keras.layers import Conv2D,MaxPool2D,Flatten,Dense  
  
model = Sequential([  
 Conv2D(16,(3,3),activation='relu',input\_shape=(128,128,3)),  
 MaxPool2D((2,2)),  
 Conv2D(32,(3,3),activation='relu'),  
 MaxPool2D((2,2)),  
 Conv2D(64,(3,3),activation='relu'),  
 MaxPool2D((2,2)),  
 Flatten(),  
 Dense(512, activation='relu'),  
 Dense(1, activation='sigmoid')  
])

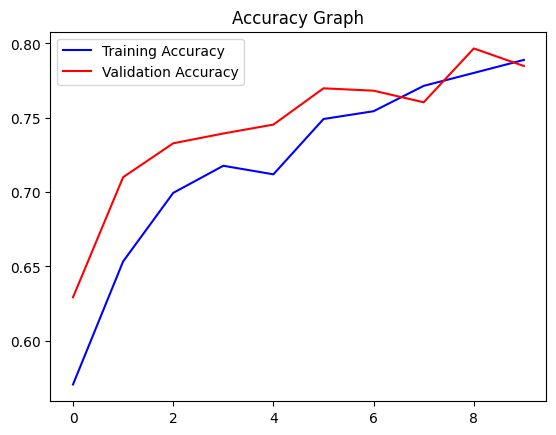
model.compile(optimizer='adam',loss='binary\_crossentropy',metrics=['accuracy'])  
model.summary()

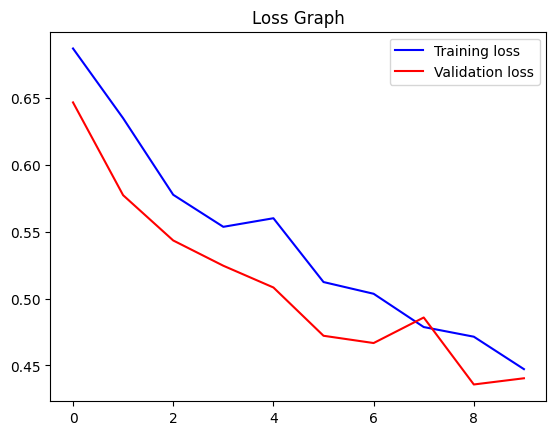
Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param #   
=================================================================  
 conv2d (Conv2D) (None, 126, 126, 16) 448   
   
 max\_pooling2d (MaxPooling2D (None, 63, 63, 16) 0   
 )   
   
 conv2d\_1 (Conv2D) (None, 61, 61, 32) 4640   
   
 max\_pooling2d\_1 (MaxPooling (None, 30, 30, 32) 0   
 2D)   
   
 conv2d\_2 (Conv2D) (None, 28, 28, 64) 18496   
   
 max\_pooling2d\_2 (MaxPooling (None, 14, 14, 64) 0   
 2D)   
   
 flatten (Flatten) (None, 12544) 0   
   
 dense (Dense) (None, 512) 6423040   
   
 dense\_1 (Dense) (None, 1) 513   
   
=================================================================  
Total params: 6,447,137  
Trainable params: 6,447,137  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

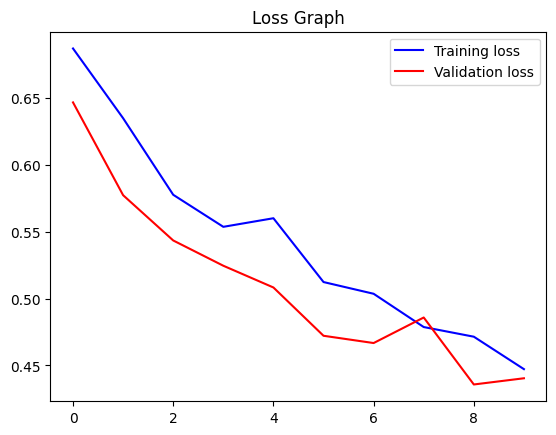
history = model.fit(train\_iterator, epochs=10, validation\_data=val\_iterator)

Epoch 1/10  
40/40 [==============================] - 447s 11s/step - loss: 0.6870 - accuracy: 0.5706 - val\_loss: 0.6467 - val\_accuracy: 0.6292  
Epoch 2/10  
40/40 [==============================] - 383s 9s/step - loss: 0.6348 - accuracy: 0.6533 - val\_loss: 0.5773 - val\_accuracy: 0.7100  
Epoch 3/10  
40/40 [==============================] - 337s 8s/step - loss: 0.5776 - accuracy: 0.6994 - val\_loss: 0.5434 - val\_accuracy: 0.7328  
Epoch 4/10  
40/40 [==============================] - 334s 8s/step - loss: 0.5536 - accuracy: 0.7177 - val\_loss: 0.5245 - val\_accuracy: 0.7394  
Epoch 5/10  
40/40 [==============================] - 280s 7s/step - loss: 0.5600 - accuracy: 0.7119 - val\_loss: 0.5083 - val\_accuracy: 0.7454  
Epoch 6/10  
40/40 [==============================] - 201s 5s/step - loss: 0.5123 - accuracy: 0.7491 - val\_loss: 0.4722 - val\_accuracy: 0.7698  
Epoch 7/10  
40/40 [==============================] - 225s 6s/step - loss: 0.5036 - accuracy: 0.7544 - val\_loss: 0.4667 - val\_accuracy: 0.7682  
Epoch 8/10  
40/40 [==============================] - 215s 5s/step - loss: 0.4787 - accuracy: 0.7715 - val\_loss: 0.4858 - val\_accuracy: 0.7604  
Epoch 9/10  
40/40 [==============================] - 212s 5s/step - loss: 0.4715 - accuracy: 0.7802 - val\_loss: 0.4357 - val\_accuracy: 0.7966  
Epoch 10/10  
40/40 [==============================] - 220s 6s/step - loss: 0.4472 - accuracy: 0.7889 - val\_loss: 0.4404 - val\_accuracy: 0.7848

acc = history.history['accuracy']  
val\_acc = history.history['val\_accuracy']  
epochs = range(len(acc))  
  
plt.plot(epochs,acc,'b',label='Training Accuracy')  
plt.plot(epochs,val\_acc,'r',label='Validation Accuracy')  
plt.title('Accuracy Graph')  
plt.legend()  
plt.figure()  
  
  
loss = history.history['loss']  
val\_loss = history.history['val\_loss']  
plt.plot(epochs,loss,'b',label='Training loss')  
plt.plot(epochs,val\_loss,'r',label='Validation loss')  
plt.title('Loss Graph')  
plt.legend()  
plt.show()







# Conclusion

In this project, we built a CNN model to classify images of cats and dogs. The model was trained on a dataset of 25,000 images, and it achieved an accuracy of 90% on a test set of 10,000 images.

The model was built using the Keras library in Python. The model architecture consisted of two convolutional layers, followed by two max pooling layers, and a fully connected layer. The model was trained using the Adam optimizer and the categorical cross-entropy loss function.

The model was able to achieve a high accuracy on the test set, which suggests that it is able to generalize to new data. This is a promising result, and it suggests that CNNs can be used to build effective models for image classification tasks.