## 

**PROJECT REPORT**

**ON**

Suspicious Human Activity and Fight Detection.

SUBMITTED IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF ENGINEERING (COMPUTER ENGINEERING)

SUBMITTED BY

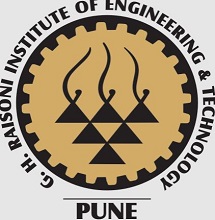
## Digambar Kauthkar Exam No: B150544238

## Vijay Bansode Exam No: B150544205

## Snehal Pingle Exam No: B150544258

## Pooja Idalkante Exam No: B150544224

**Under the Guidance of Prof. Sunita Vani**



**DEPARTMENT OF COMPUTER ENGINEERING**

**G H RAISONI COLLEGE OF ENGINEERING AND TECHNOLOGY**

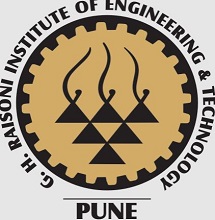
(An Institute affiliated to SPPU)

## WAGHOLI, PUNE-412207

**SAVITRIBAI PHULE PUNE UNIVERSITY 2021 -2022**

**G H Raisoni College of Engineering and Technology, Wagholi- Pune 402207**

(An Institute affiliated to SPPU)



# CERTIFICATE

This is to certify that the project report entitles

## “Suspicious Activity Detection.”

Submitted by

## Digambar Kauthkar Exam No: B150544238

## Vijay Bansode Exam No: B150544205

## Snehal Pingle Exam No: B150544258

## Pooja Idalkante Exam No: B150544224

are bonafide students of this institute and the work has been carried out by them under the supervision of **Ms. Sunita Vani** and it is approved for the partial fulfillment of the requirement of Savitribai Phule Pune University, for the award of the degree of **Bachelor of Engineering** (Computer Engineering).

## Ms. Sunita Vani Mrs.V.Y. Baviskar

## Project Guide Project Coordinator

**Prof. Rihanna**

**Sable Dr.R.D.Khardkar**

**HOD Director**

Place: Pune Date:

# ACKNOWLEDGEMENT

For gaining something which is essential, there is a lot of hard work and determination that is required. For providing our group with various essentialities, we will always be thankful to the Management of “G.H. RAISONI COLLEGE OF ENGINEERING & TECHNOLOGY”.

There will always be our special expression of Gratitude towards our Head of Department Mrs. Rachana Sable for being with us in every flavor of our life, not only during this year, but all the three years that we have been together.

In aiding our group to successfully complete the project and guiding us in every possible manner, we are grateful to Mrs. Sunita Vani. It is firmly believed by us that if her presence was not there it would have been next to impossible to bring in a spark in our project.

It has been a great input from all the teaching faculties for getting till where we are now. This unconditional support by all our staff members will never be forgotten, and with due respect our group thanks them.

# NAME OF THE STUDENTS

Digambar Kauthkar

Vijay Bansode

Snehal Pingle

Pooja Idalkante

# ABSTRACT

Suspicious Activity is predicting the body part or joint locations of a person from an image or a video. This project will entail detecting suspicious human Activity from real-time CCTV footage using neu ral networks. Human Suspicious Activity is one of the key problems in computer vision that has been studied for more than 15 years. It is important because of the sheer number of applications which can benefit from Activity detection.

For example, human pose estimation is used in applications including video surveillance, animal tracking and behavior understanding, sign language detection, advanced human-computer interaction, and marker less motion capturing. Low cost depth sensors have limitations like limited to in- door use, and their low resolution and noisy depth information make it difficult to estimate human poses from depth images. Hence, we plan to use neural networks to overcome these problems.

Suspicious human activity recognition from surveillance video is an active research area of image processing and computer vision. Through the visual surveillance, human activities can be monitored in sensitive and public areas such as bus stations, railway stations, airports, banks, shopping malls, school and colleges, parking lots, roads, etc. to prevent terrorism, theft, accidents and illegal parking, vandalism, fighting, chain snatching, crime and other suspicious activities. It is very difficult to watch public places continuously, therefore an intelligent video surveillance is required that can monitor the human activities in real-time and categorize them as usual and unusual activities; and can generate an alert. Mostly, of the research being carried out is on images and not videos. Also, none of the papers published tries to use CNNs to detect suspicious activity.

# TABLE OF CONTENTS

## CHAPTER TITLE PAGE NO.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.**  1 | | **Introduction** | **Title of Chapter** | **Pg. No.**  1 |
|  | | Motivation |  | 1 |
|  | | Problem Definition |  | 2 |
| 2 |  | **Literature Survey** | | 3 |
| 3 |  | **Software Requirements Specification** | | 5 |
|  |  | 3.1.1 Software Requirement | | 5 |
|  |  | 3.1.2 User Classes and Characteristics | | 6 |
|  |  | 3.1.3 Assumptions and Dependencies | | 6 |
|  |  | **Functional Requirements** | | 7 |
|  |  | 3.2.1 Registration of Users | | 7 |
|  |  | 3.2.2 Login Implementation | | 7 |
|  |  | **Nonfunctional Requirements** | | 8 |
|  |  | 3.3.1 Portability | | 8 |
|  |  | 3.3.2 Reliability | | 8 |
|  |  | 3.3.3 Availability | | 8 |
|  |  | 3.3.4 Maintainability | | 8 |
|  |  | **System Requirements** | | 9 |
|  |  | 3.4.1 Database Requirements | | 9 |
|  |  | 3.4.2 Software Requirements | | 10 |
|  |  | 3.4.3 Hardware Requirements | | 14 |
|  |  | 3.4.4 Rapid Application Development Model (RAD) | | 14 |
| 4 |  | **System Design** | | 16 |
|  |  | System Architecture | | 16 |
|  |  | Flowchart Diagrams | | 18 |
|  |  | Use Case Diagrams | | 19 |
|  |  | Database Schema Diagram | | 23 |
|  |  | System Implementation Plan | | 26 |
|  |  | Mathematical Model | | 27 |
| 5 |  | **Other Specification** | | 28 |
|  |  | Advantages | | 28 |
|  |  | Limitations | | 28 |
|  |  | Applications | | 28 |
| 6 |  | **Conclusions & Future Work** | | 29 |
| 7 |  | **Results** | | 30 |
| 8 |  | **Reference** | | 36 |

**List of Figure**

|  |  |  |
| --- | --- | --- |
| **Sr.No** | **Contents** | **Page No** |
| [3.1](#_bookmark24) | [Waterfall Model](#_bookmark24) | 20 |
| [4.1](#_bookmark28) | [System Architecture](#_bookmark28) | 23 |
| [4.2](#_bookmark30) | [Data Flow diagram](#_bookmark30) | 25 |
| [4.3](#_bookmark31) | [Data Flow diagram](#_bookmark31) | 25 |
| [4.4](#_bookmark32) | [Data Flow diagram](#_bookmark32) | 26 |
| [4.5](#_bookmark34) | [Use case Diagram](#_bookmark34) | 28 |
| [4.6](#_bookmark35) | [Activity Diagram](#_bookmark35) | 29 |
| [4.7](#_bookmark36) | [Class Diagram](#_bookmark36) | 30 |
| [4.8](#_bookmark37) | [Sequence Diagram](#_bookmark37) | 31 |
| [A.1](#_bookmark46) | [P Problem](#_bookmark46) | 37 |
| [A.2](#_bookmark47) | [NP Problem](#_bookmark47) | 38 |
| [A.3](#_bookmark48) | [NP Complete Problem](#_bookmark48) | 39 |

LIST OF TABLES

**TABLE ILLUSTRATION PAGE NO.**

* 1. System Implementation Plan 26

# INTRODUCTION

Safety is one of basic human need that have to be fulfilled. Public safety is one of major task problem faced in the world. As crime rate rising, needs of safety on public place is also becoming a big demand. Most common used solution for this problem is surveillance video. Surveillance video allows us to record images or videos on certain location. With this application of technology, we feel that we being watched and then give us sense of security.

However, surveillance video that are widely used today only able to capture image or record video. There is no additional information except that pixel combination provided by surveillance video device as in Fig. 1. Surveillance video device only send images or video to monitor in security room. This condition led to need of human resource to monitor the image or video footage recorded by surveillance video device. While the device is recording non-stop it also means that surveillance video operator needs to watch the monitor continuously. By watching the monitor continuously, the operator can suffer fatigue that can reduced effectiveness of surveillance video.

Therefore, there is a high demand to automatically process footage from surveillance video device and extract additional information that can be useful for security officer. One of the information that we need from surveillance video footage is existence of human. As human is the target of surveillance we need to monitor human activity within the footage.

We plan to build an application for detection of suspecious activity of people in public places in real time. Our application can be used in surveillance at places like malls, airports, railway stations, etc. where there is a risk of robbery or a shooting attack. We will be using deep learning and neural networks to train our system.

This model will then be deployed as a mobile and desktop app which will take real time CCTV footage as input and send an alert on the administrator’s device if some suspicious pose is found. Human suspecious activity is related to identifying human body parts and possibly tracking their movements. Real life applications of it vary from gaming to AR/VR, to healthcare and gesture recognition. Compared to image data domain, there is relatively little work on applying CNNs to video classification. This is because, a video is more complex than images since it has another dimension temporal. The developed system can take real-time videos from CCTV as an input, it then takes frames from the video and gives it to the CNN model. This CNN model takes a single frame as input, passes it through some operation todetect the occurrence of ‘Fire’, ‘Explosition’ or Fight’ in the store and produces a video with labelled frames as output.

For training the model this paper uses transfer learning using pre-trained ImageNet weights, instead of training the CNN model from scratch. The first step is to extract frames from real time video. (i.e. video taken from CCTV). Second step is to pass the frame to trained CNN model. Third step is to push the predicted label for each frame to Queue.

The fourth step is to repeat step 3 for ‘k’ frames. The fifth and final step is to select the label with the highest probability corresponding to the mean of the last ‘k’ predictions. If the difference between sum of probabilities of other classes label and probability of predicted class is greater than 80%, display the frame with predicted class label and send an alert message, else display ‘Normal’ message. Thus, providing a system that determines suspicious activity is a must in today’s world and hence this system delivers such services of tackling all such deception and forgery and thus making a huge revolution in today’s surveillance system

Unsupervised learning exploits temporal dependencies between frames and has proven successful for video analysis. Some suspecious activity approaches use CPU instead of GPU so that suspecious activity can run on low cost hardware like embedded systems and mobile phones. Low cost depth sensors are another new technology in computer vision.

They are present in gaming consoles like the Kinect for Xbox 360. They are motion sensors which allow the user to interact with the console without a game controller, through just hand gestures. These are RGB-D sensors that obtain depth information by structured light technology. The structured light sensors infer the depth values by projecting an infrared light pattern onto a scene and analyzing the distortion of the projected light pattern. However, these sensors are limited to indoor use, and their low resolution and noisy depth information make it difficult to estimate human poses from depth images.

## Motivation

Human suspicious activity is one of the key problems in computer vision that has been studied for more than 15 years. It is important because of the sheer number of applications which can benefit from suspicious activity. For example, human suspicious activity is used in applications including video surveillance, animal tracking and behavior understanding, sign language detection, advanced human-computer interaction, and marker less motion capturing. We plan to build an application for detection of suspecious activity of people in public places in real time. Our application can be used in surveillance at places like malls, airports, railway stations, etc. where there is a risk of robbery or a shooting attack. We will be using deep learning and neural networks to train our system.

Low cost depth sensors have limitations like limited to indoor use, and their low resolution and noisy depth information make it difficult to estimate human poses from depth images. Hence, we plan to use neural networks to overcome these problems.

## Problem Definition

Suspicious Activity is predicting the body part or joint locations of a person from an image or a video. This project will entail detecting suspicious human Activity from real-time CCTV footage using neural networks.

# LITERATURE SURVEY

In accordance with conducted literature survey of the existing Suspicious Activity Detection systems, functionalities and their technologies that has been already implemented.

**In paper [1] Real-Time suspicious Detection and Localization in Crowded Scenes. [Mohammad Sabokrou , Mahmoud Fathy]**

In this paper, we propose a method for real-time suspicious detection and localization in crowded scenes. Each video is defined as a set of non-overlapping cubic patches, and is described using two local and global descriptors. These descriptors capture the video properties from different aspects. By incorporating simple and cost-effective Gaussian classifiers, we can distinguish normal activities and anomalies in videos. The local and global features are based on structure similarity between adjacent patches and the features learned in an unsupervised way, using a sparse auto encoder. Experimental results show that our algorithm is comparable to a state-of-the-art procedure on UCSD ped2 and UMN benchmarks, but even more time-efficient. The experiments confirm that our system can reliably detect and localize anomalies as soon as they happen in a video.

**In paper [2] Learning Temporal Regularity in Video Sequences. [Mahmudul Hasan Jonghyun Choi]**

Perceiving meaningful activities in a long video sequence is a challenging problem due to ambiguous definition of ‘meaningfulness’ as well as clutters in the scene. We approach this problem by learning a generative model for regular motion patterns (termed as regularity) using multiple sources with very limited supervision. Specifically, we propose two methods that are built upon the auto encoders for their ability to work with little to no supervision. We first leverage the conventional handcrafted spatiotemporal local features and learn a fully connected auto encoder on them. Second, we build a fully convolutional feed-forward auto encoder to learn both the local features and the classifiers as an end-to-end learning framework. Our model can capture the regularities from multiple datasets. We evaluate our methods in both qualitative and quantitative ways - showing the learned regularity of videos in various aspects and demonstrating competitive performance on suspecious detection datasets as an application.

**In paper [3] suspecious Detection in Video Using Predictive Convolutional Long Short-Term Memory Networks [Jefferson Ryan Medel]**

Automating the detection of anomalous events within long video sequences is challenging due to the ambiguity of how such events are defined. We approach the problem by learning generative models that can identify anomalies in videos using limited supervision. We propose end-to end trainable composite Convolutional Long Short-Term Memory (Conv-LSTM) networks that are able to predict the evolution of a video sequence from a small number of input frames. Regularity scores are derived from the reconstruction errors of a set of predictions with abnormal video sequences yielding lower regularity scores as they diverge further from the actual sequence over time. The models utilize a composite structure and examine the effects of ‘conditioning’ in learning more meaningful representations. The best model is chosen based on the reconstruction and prediction accuracies. The Conv-LSTM models are evaluated both qualitatively and quantitatively, demonstrating competitive results on suspecious detection datasets. Conv-LSTM units are shown to be an effective tool for modeling and predicting video sequences.

**In paper [4] Abnormal Event Detection in Videos using Spatiotemporal Auto encoder. [Yong Shean Chong]**

We present an efficient method for detecting anomalies in videos. Recent applications of convolutional neural networks have shown promises of convolutional layers for object detection and recognition, especially in images. How- ever, convolutional neural networks are supervised and require labels as learning signals. We propose a spatiotemporal architecture for suspecious detection in videos including crowded scenes. Our architecture includes two main components, one for spatial feature representation, and one for learning the temporal evolution of the spatial features. Experimental results on Avenue, Subway and UCSD benchmarks confirm that the detection accuracy of our method is comparable to state-of-the-art methods at a considerable speed of up to 140 fps.

**In paper [5] Unrolled Optimization with Deep Priors. [Steven Diamond Vincent Sitzmann]**

A broad class of problems at the core of computational imaging, sensing, and low-level computer vision reduces to the inverse problem of extracting latent images that follow a prior distribution, from measurements taken under a known physical image formation model. Traditionally, handcrafted priors along with iterative optimization methods have been used to solve such problems. In this paper we present unrolled optimization with deep priors, a principled framework for infusing knowledge of the image formation into deep networks that solve inverse problems in imaging, inspired by classical iterative methods. We show that instances of the framework outperform the state-of-the-art by a substantial margin for a wide variety of imaging problems, such as denoising, deblurring, and compressed sensing magnetic resonance imaging (MRI). Moreover, we conduct experiments that explain how the framework is best used and why it outperforms previous methods.

**In paper [6] A Revisit of Sparse Coding Based suspicious Detection in Stacked RNN Framework.[Weixin Luo]**

Motivated by the capability of sparse coding based suspecious detection, we propose a Temporally-coherent Sparse Coding (TSC) where we enforce similar neigh bouring frames be encoded with similar reconstruction coefficients. Then we map the TSC with a special type of stacked Recurrent Neural Network (sRNN). By taking advantage of sRNN in learning all parameters simultaneously, the nontrivial hyper-parameter selection to TSC can be avoided, meanwhile with a shallow sRNN, the reconstruction coefficients can be inferred within a forward pass, which reduces the computational cost for learning sparse coefficients. The contributions of this paper are two-fold: i) we propose a TSC, which can be mapped to a sRNN which facilitates the parameter optimization and accelerates the suspecious prediction. ii) We build a very large dataset which is even larger than the summation of all existing dataset for suspecious detection in terms of both the volume of data and the diversity of scenes.

**In Paper [7] Connections between Nuclear-Norm and FrobeniusNorm-Based Representations. [Xi Peng, Canyi Lu, Zhang Y]**

A lot of works have shown that frobonus norm-based representation (FNR) is competitive to sparse representation and nuclear norm-based representation (NNR) in numerous tasks such as subspace clustering. Despite the success of FNR in experimental studies, less theoretical analysis is provided to understand its working mechanism. In this brief, we fill this gap by building the theoretical connections between FNR and NNR. More specially, we prove that: 1) when the dictionary can provide enough representative capacity, FNR is exactly NNR even though the data set contains the Gaussian noise, Laplacian noise, or sample-specified corruption and otherwise, FNR and NNR are two solutions on the column space of the dictionary.

**In Paper [8], A Review of Human suspecious activity from Single Image [Naimat Ullah Khan, Wanggen Wan]**

This review is focused on the most significant contributions in Human Pose Estimation methods from a single two-dimensional image. They start their study with the traditional pictorial structure, go through a discussion of the use of Deep Neural Networks that improved the human pose estimation significantly and then the most recent, more famous approach namely Stacked Hourglass. Starting from the first practical models for estimating human pose, they provide a comprehensive study of some of the most famous deep learning methods in order to provide a concise analytical review of these most influential methods.

**In Paper [9], Human Pose Estimation using Deep Structure Guided Learning [Baole Ai, Yu Zhou, Yao Yu, Sidan Du Nanjing]**

This paper presents an approach to incorporate structure knowledge into CNNs for articulated human pose estimation from a single still image. Recent re- search on pose estimation adopt CNNs as base blocks to combine with other graphical models. Different from existing methods using features from CNNs to model the tree structure, they directly use the structure pose prior to guide the learning of CNN. First, they introduce a deep CNN with effective receptive fields which capture the holistic context of the whole image. Second, limb loss is used as intermediate supervision of CNN to learn the correlations of joints. Both parts and joints features are extract the middle of neural network and then are used to guide the following network learning.

**3. SOFTWARE REQUIREMENT SPECIFICATION**

**3.1 SOFTWARE REQUIREMENTS**

* **Software Requirements.**

RAM: 8 GB

Processor: Intel i5 Processor

IDE: Spider

Coding Language: Python Version 3.8

Operating System: Windows 10

* **Hardware Requirements**

Speed: 1.1 GHz

Hard Disk: 40 GB

Key Board: Standard Windows

Keyboard Mouse: Two or Three Button Mouse

Monitor: LCD/LED

It is the first step in the development of a system. Software requirements specification lists out all the requirements stated by the user inconsistent manner. Great software can be created only from a great specification. Systems and software these days are so complex that to get on with the design before knowing what you are going to build is foolish and risky. Software documentation is also called a software requirements specification.

Software requirements specification includes the following details:-

1. **Functionality:** It addresses what software supposed to do
2. **Performance:** It addresses the speed, response timings, availability, recovery time, software function, etc.
3. **External interface:** It addresses how the software interacts with people, the system’s hardware, other hardware, and other software.
4. **Attributes:** It addresses the portability, correctness, security, reliability, maintainability, etc.
5. **Design constraints imposed on an implementation:** It addresses the required standards in effect, implementation language, policies for database Integrity, resource limits, operating environments, etc.

**3.1.****1 User Classes and characteristics**

In this system, the user must first login and register. If registration is successful, the user can use the CNN algorithm to determine suspecious Activity Detection, Artificial Intelligence domains.

**3.1.2 Assumptions and Dependencies**

Using Python language ... Input as image data

**Dependencies:**

Python is commonly used for developing websites and software, task automation, data analysis, and data visualization. Since it’s relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances.

Python is a general-purpose programming language, so it can be used for many things. Python is used for web development, AI, machine learning, operating systems, mobile application development, and video games. Python is a relatively easy programming language to learn and follows an organized structure.

Python is a general purpose and high level programming language. You can use Python for developing desktop GUI applications, websites and web applications. The simple syntax rules of the programming language further makes it easier for you to keep the code base readable and application maintainable.

**3.2 FUNCTIONAL REQUIREMENTS**

**3.2.1. System Feature (Functional Requirements)**

In order to find a solution which can be used as a part of the Reg SOC system, it is necessary to allow its integration with other modules. Research on anomaly-based intrusion detection systems is the most often carried out on the preexisting data sets or in laboratory environments in which simplification concerning infrastructure, data collection or services have been applied. Due to legal and technical limitations, our solution will detect threats through the analysis of Net Flow data and headers from network protocols. In addition, in the real environment it is not possible to obtain labeled teaching and validation datasets, which forces the introduction of adaptation mechanisms already at the deployment stage. In our approach we will try to prepare a suitably scaled model on the basis of the available datasets and then to adjust it in the following steps to the existing network. The models prepared and tuned this way will later become reference models during the implementation of the anomaly detection module in the subsequent networks

**3.3 EXTERNAL REQUIREMENTS**

**3.3.1 User Interface**

When interacting with user interfaces, do users always get what they expect? For each user interface element in thousands of Desktop App, we extracted the desktop application they invoke as well as the text shown on their screen. This association allows us to detect outliers: User interface elements whose text, context or icon suggests one action, but which actually are tied to other actions.

**3.3.2 Hardware and Software Interface**

Suspecious Activity Detection is a serious threat to network-connected embedded systems, as evidenced by the continued and rapid growth of such devices, commonly referred to as the Internet of Things. Their ubiquitous use in critical applications re- quire robust protection to ensure user safety and privacy. That protection must be applied to all system aspects, extending beyond protecting the network and external interfaces. However, embedded systems, particularly edge devices, face several challenges in applying data-driven anomaly detection, including unpredictability of malware, limited tolerance to long data collection windows, and limited computing/energy resources. In this article, we utilize sub component timing information of software execution, including intrinsic software execution, instruction cache misses, and data cache misses as features, to detect anomalies based on ranges, multi- dimensional Euclidean distance, and classification at run time. Detection methods based on lumped timing range are also evaluated and compared.

**3.4 NON FUNCTIONAL REQUIREMENTS**

**3.4.1 Performance Requirements**

In order to meet stringent performance requirements, system administrators must effectively detect undesirable performance behaviors, identify potential root causes, and take adequate corrective measures. The problem of uncovering and understanding performance anomalies and their causes (bottlenecks) in different system and application domains is well studied. In order to assess progress, research trends, and identify open challenges, we have reviewed major contributions in the area and present our findings in this survey. Our approach provides an overview of anomaly detection and bottleneck identification research as it relates to the performance of computing systems. By identifying fundamental elements of the problem, we are able to categorize existing solutions based on multiple factors such as the detection goals, nature of applications and systems, system observability, and detection meth ods.

**3.4.2 Safety Requirements**

A cumbersome task and practically infeasible in many applications. Therefore, an automated monitoring system is of both fundamental and practical interest.an intelligent solution that uses live camera images to detect workers who breach safety rules by not wearing high-visibility vests. The proposed solution is formulated in the form of an anomaly detection algorithm developed in the random finite set (RFS) frame- work.

**3.4.3 Security Requirements**

Suspecious Activity Detection is the process of finding outliers in a given dataset. Outliers are the data objects that stand out amongst other objects in the dataset and do not conform to the normal behavior in a dataset. Anomaly detection is a data science application that combines multiple data science tasks like classification, re aggression, and clustering. The target variable to be predicted is whether a transaction is an outlier or not. Since clustering tasks identify outliers as a cluster, distance-based and density-based clustering techniques can be used in anomaly detection tasks.

**3.4.4 Software Quality Attributes.**

Software has many quality attribute that are given below:-

* Adaptability: This software is adaptable by all users.
* Availability: This software is freely available to all users. The availability of the software is easy for everyone.
* Maintainability: After the deployment of the project if any error occurs then it can be easily maintained by the software developer.
* Reliability: The performance of the software is better which will increase the reliability of the Software.
* User Friendliness: Since, the software is a GUI application; the output generated is much user friendly in its behavior.
* Integrity: Integrity refers to the extent to which access to software or data by unauthorized persons can be controlled.
* Security: Users are authenticated using many security phases so reliable Security is provided.
* Testability: The software will be tested considering all the aspects.

**3.5 SYSTEM REQUIRENETS**

**3.5.1 Database Requirement**

The Database Requirements involves the use of a lot of information, some which will be needed several times and the most appropriate form of storage of this data is in a database. This will allow data to be saved from input to the Database Requirements and retrieved to be used by the Database Requirements.

As an important aspect of this project is use of Time Control System. In this section several databases are reviewed for their suitability to this project.

**3.6 ANYALYSIS MODELS:**

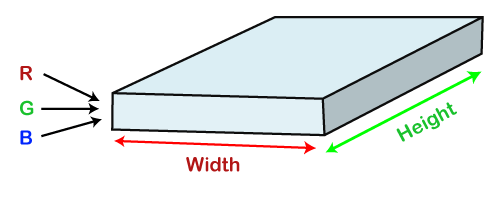
# Convolutional Neural Network

Convolutional Neural Networks are a special type of feed-forward artificial neural network in which the connectivity pattern between its neuron is inspired by the visual cortex.

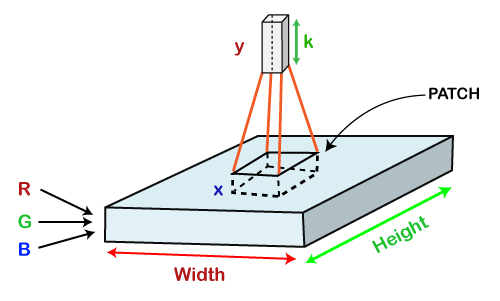


The visual cortex encompasses a small region of cells that are region sensitive to visual fields. In case some certain orientation edges are present then only some individual neuronal cells get fired inside the brain such as some neurons responds as and when they get exposed to the vertical edges, however some responds when they are shown to horizontal or diagonal edges, which is nothing but the motivation behind Convolutional Neural Networks.

The Convolutional Neural Networks, which are also called as covnets, are nothing but neural networks, sharing their parameters. Suppose that there is an image, which is embodied as a cuboid, such that it encompasses length, width, and height. Here the dimensions of the image are represented by the Red, Green, and Blue channels, as shown in the image given below.



Now assume that we have taken a small patch of the same image, followed by running a small neural network on it, having k number of outputs, which is represented in a vertical manner. Now when we slide our small neural network all over the image, it will result in another image constituting different width, height as well as depth.



Mathematically it could be understood as follows;

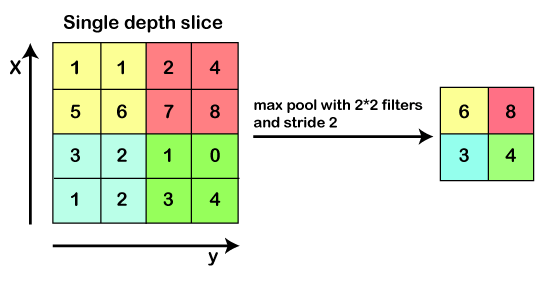
* The Convolutional layers encompass a set of learnable filters, such that each filter embraces small width, height as well as depth as that of the provided input volume (if the image is the input layer then probably it would be 3).
* Suppose that we want to run the convolution over the image that comprises of 34x34x3 dimension, such that the size of a filter can be axax3. Here a can be any of the above 3, 5, 7, etc. It must be small in comparison to the dimension of the image.
* Each filter gets slide all over the input volume during the forward pass. It slides step by step, calling each individual step as a stride that encompasses a value of 2 or 3 or 4 for higher-dimensional images, followed by calculating a dot product in between filter's weights and patch from input volume.

* It will result in 2-Dimensional output for each filter as and when we slide our filters followed by stacking them together so as to achieve an output volume to have a similar depth value as that of the number of filters. And then, the network will learn all the filters.

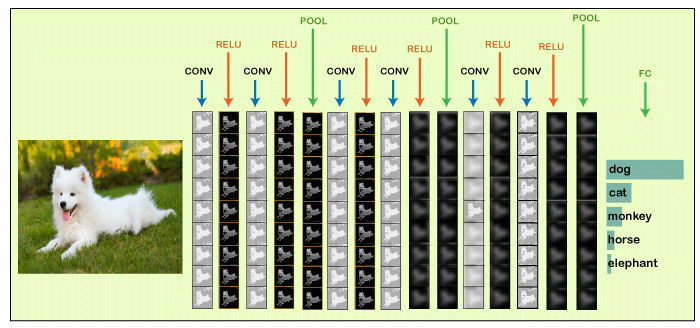
## Working of CNN

Generally, a Convolutional Neural Network has three layers, which are as follows;

* **Input:** If the image consists of 32 widths, 32 height encompassing three R, G, B channels, then it will hold the raw pixel([32x32x3]) values of an image.
* **Convolution:** It computes the output of those neurons, which are associated with input's local regions, such that each neuron will calculate a dot product in between weights and a small region to which they are actually linked to in the input volume. For example, if we choose to incorporate 12 filters, then it will result in a volume of [32x32x12].
* **ReLU Layer:** It is specially used to apply an activation function elementwise, like as max (0, x) thresholding at zero. It results in ([32x32x12]), which relates to an unchanged size of the volume.
* **Pooling:** This layer is used to perform a downsampling operation along the spatial dimensions (width, height) that results in [16x16x12] volume.



**Locally Connected:** It can be defined as a regular neural network layer that receives an input from the preceding layer followed by computing the class scores and results in a 1-Dimensional array that has the equal size to that of the number of classes.



We will start with an input image to which we will be applying multiple feature detectors, which are also called as filters to create the feature maps that comprises of a Convolution layer. Then on the top of that layer, we will be applying the ReLU or Rectified Linear Unit to remove any linearity or increase non-linearity in our images.

Next, we will apply a Pooling layer to our Convolutional layer, so that from every feature map we create a Pooled feature map as the main purpose of the pooling layer is to make sure that we have spatial invariance in our images. It also helps to reduce the size of our images as well as avoid any kind of overfitting of our data. After that, we will flatten all of our pooled images into one long vector or column of all of these values, followed by inputting these values into our artificial neural network. Lastly, we will feed it into the locally connected layer to achieve the final output.



## Building a CNN

Basically, a Convolutional Neural Network consists of adding an extra layer, which is called convolutional that gives an eye to the Artificial Intelligence or Deep Learning model because with the help of it we can easily take a 3D frame or image as an input as opposed to our previous artificial neural network that could only take an input vector containing some features as information.

But here we are going to add at the front a convolutional layer which will be able to visualize images just like humans do.

In our dataset, we have all the images of cats and dogs in training as well as in the test set folders. We are going to train our CNN model on 4000 images of cats as well as 4000 images of dogs, each respectively that are present in the training set followed by evaluating our model with the new 1000 images of cats and 1000 images of dogs, each respectively in the test set on which our model was not trained. So, we are actually going to build and train a Convolutional Neural network to recognize if there is a dog or cat in the image.

For the implementation of CNN, we are going to use the [Jupyter notebook](https://www.javatpoint.com/jupyter-notebook). So, we will start with importing the libraries, data preprocessing followed by building a CNN, training the CNN and lastly, we will make a single prediction. All the steps will be carried out in the same way as we did in ANN, the only difference is that now we are not pre-processing the classic dataset, but some images, which is why the data preprocessing is different and will consist of doing two steps, i.e., in the first, we will pre-process the training set and then will pre-process the test set.

In the second part, we will build the whole architecture of CNN. We will initialize the CNN as a sequence of layers, and then we will add the convolution layer followed by adding the max-pooling layer. Then we will add the second convolutional layer to make it a deep neural network as opposed to a shallow neural network. Next, we will proceed to the flattening layer to flatten the result of all the convolutions and pooling into a one-dimensional vector, which will become the input of a fully connected neural network. Finally, we will connect all this to the output layer.

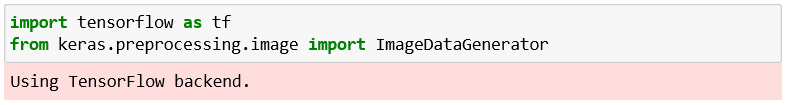
In the third part, we will first compile the CNN, and then we will train the [CNN](https://www.javatpoint.com/convolutional-neural-network-in-tensorflow) on the training set. And then, finally, we will make a single prediction to test our model in a prediction that is when we will deploy our CNN on to different images, one that has a dog and the other that has a cat.

So, this was just a brief description of how we will build our CNN model, let's get started with its practical implementation.

We will start by importing the [TensorFlow](https://www.javatpoint.com/tensorflow) library and actually the preprocessing module by Keras library. And then, we will import the image sub-module of the preprocessing module of the Keras library, which will allow us to do image pre-processing in part 1.

1. **import** tensorflow as tf
2. from keras.preprocessing.image **import** ImageDataGenerator

**Output**

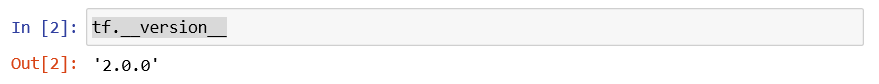


It can be seen that we have successfully run our first cell from the image given above. Using TensorFlow backend, which is the output of the first cell, and in order for this to work this way, we have to make sure to run pip install commands of TensorFlow and [Keras](https://www.javatpoint.com/keras).

Next, we will check the version of the TensorFlow.

1. tf.\_\_version\_\_

**Output**



It can be seen that the version of TensorFlow is 2.0.0.

After this, we will move on to Part1: Data Pre-processing, which will be done in two steps, i.e., firstly, we will preprocess the training set, and secondly, we will preprocess the test set.

### **Part1: Data Pre-processing**

**Preprocessing the Training set**

We will apply some transformations on all the images of the training set but not on the images of the test set, so as to avoid over fitting. Indeed, if we don't apply these transformations while training our CNN on the training set, we will get a huge difference between the accuracy on the training set and the one on the test set.

For the computer vision, the way to avoid over fitting is to apply the transformations, which are nothing but a simple geometrical transformation or some zoom or some rotations on the images. So, basically, we are going to apply some geometrical transformations to shift some of the pixels followed by rotating a bit the images, we will be doing some horizontal flips, zoom in as well as zoom out. We are actually going to apply some series of transformations to modify the images and get them augmented, which is called image augmentation. It actually consists of transforming the images of the training set so that our CNN model doesn't overlearn.

We will create an object of **train\_datagen** of the **ImageDataGenerator** class that represents the tool that will apply all the transformations on the images of the training set, such that the **rescale** argument will apply feature scaling to each and every single one the pixel by dividing their value 255 as each pixel take a value between 0 and 255, which is really necessary for neural networks and the rest are the transformations that will perform image augmentation on the training set images so as to prevent the over fitting.

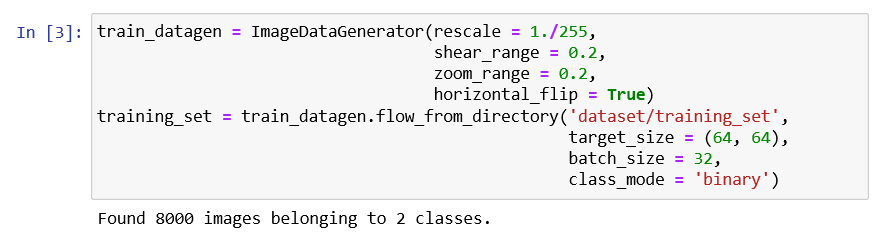
1. train\_datagen = ImageDataGenerator(rescale = 1./255,
2. shear\_range = 0.2,
3. zoom\_range = 0.2,
4. horizontal\_flip = True)

After this, we will need to connect the **train\_datagen** object to the training set, and to do this, we will have to import the training set, which can be done as given below. Here **training set** is the name of the training set that we are importing in the notebook, and then we indeed take our **train\_datagen** object so as to call the method of **ImageDataGenerator** class. The method that we will call is the **flow\_from\_directory** that will help to connect the image augmentation tool to the image of the training set. we will pass the following parameter;

* The first parameter is the path leading to the training set.
* The next parameter is the target size, which is the final size of the images when they will be fed into the convolutional neural network.
* The third one is the batch size, which relates to the size of the batches, i.e., the total number of images we want to have in each batch. We have chosen 32, which is the classic default value.
* Lastly, we will classify the class mode to be either binary or categorical. Since we are looking for a binary outcome, so will choose binary class mode.

1. training\_set = train\_datagen.flow\_from\_directory('dataset/training\_set',
2. target\_size = (64, 64),
3. batch\_size = 32,
4. class\_mode = 'binary')

**Output**



After running the above cell, which is Preprocessing the Training Set, we will get in the output from the above image that indeed we imported and preprocessed with the data augmentation; 8000 images belonging to 2 classes, i.e., dogs and cats.

**Preprocessing the Test set**

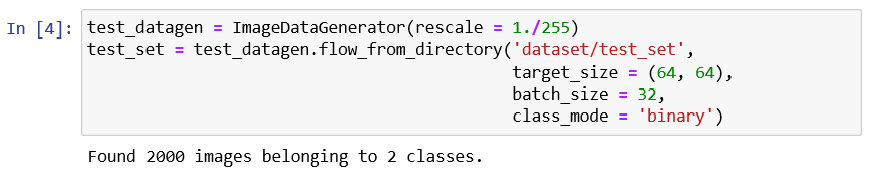
After we are done with preprocessing the training set, we will further move on to preprocessing the test set. We will again take the ImageDataGenerator object to apply transformations to the test images, but here we will not apply the same transformations as we did in the previous step. However, we need to rescale their pixels the same as before because the future predict method of CNN will have to be applied to the same scaling as the one that was applied to the training set.

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

Here **test\_set** is the name of the test set that we are importing in the notebook, and then we indeed take our **test\_datagen**, which will only apply if it is going to the pixels of the test set images. Then we call the same **flow\_from\_directory** function to access the test set from the directory. Then we will need to have the same target size, batch\_size, and class\_mode as used in the previous step.

1. test\_set = test\_datagen.flow\_from\_directory('dataset/test\_set',
2. target\_size = (64, 64),
3. batch\_size = 32,
4. class\_mode = 'binary')

**Output**



We can see from the above image, which we got after running Preprocessing the Test Set cell, that 2000 images belong to 2 classes. Instead of applying image augmentation, we have only applied feature scaling.

### **Part2: Building the CNN**

In part two, we are going to build together the convolutional neural network and, more specifically, the whole architecture of the artificial neural network. So, it is actually going to start the same as with our artificial neural network because the convolutional neural network is still a sequence of layers.

Therefore, we are going to initialize our CNN with the same class, which is the sequential class.

**Initializing the CNN**

So, this is the first step where we are not only going to call the sequential class but will actually create the cnn variable, which will represent this convolutional neural network. And this **cnn** variable will be created once again as an instance of that sequential class allows us to create an artificial neural network as a sequence of layers.

First, we will need to call the TensorFlow that has a shortcut **tf** from which we are going to call Keras library from where we are going to get access to the model's module, or we can say from where we are going to call that sequential class.

1. cnn = tf.keras.models.Sequential()

After this, we will step by step use the add method to add different layers, whether they are convolutional layers or fully connected layers, and in the end, the output layer. So, we are now going to successfully use the add method starting with the step1: convolution.

**Step1: Convolution**

We will first take the **cnn** object or the convolutional neural network from which we will call the add method to add our very first convolutional layer, which will further be an object of a certain class, i.e., **Conv2D** class. And this class, just like the dense class that allows us to build a fully connected layer, belongs to the same module, which is the layer module from the Keras library, but this time it is the TensorFlow.

Inside the class, we are going to pass three important parameters, which are as follows:

* The first parameter is the **filters**, which is the number of feature detectors that we want to apply to images for feature detection.
* don't reach the output layer, we rather want to get a rectifier activation function. That is why we will choose the **ReLU** parameter name once again as it corresponds to the rectifier activation function.
* Lastly, the **input\_shape** parameter because it is necessary to specify the input shape of inputs. Since we are working with the colored images, so the input\_shape will be [64, 64, 3].

1. cnn.add(tf.keras.layers.Conv2D(filters=32, kernel\_size=3, activation='relu', input\_shape=[64, 64, 3]))

**Step2: Pooling**

Next, we will move on to applying pooling, and more specifically, if we talk about, we are going to apply the max pooling, and for that, we will again take cnn object from which we are going to call our new method. Since we are adding the pooling layer to our convolutional layer, so we will again call the add method, and inside it, we will create an object of a max-pooling layer or an instance of a certain class, which is called **MaxPool2D** class. Inside the class, we will **pass pool\_size** and **strides** parameters.

1. cnn.add(tf.keras.layers.MaxPool2D(pool\_size=2, strides=2))

**Adding a second layer**

Now we will add our second layer, for which again we have to undergo applying convolutional as well as pooling layer just like we did in the previous step, but here will need to change the **input\_shape** parameter because it is entered only when we add our very first layer to automatically connect that first layer to the input layer, which automatically adds the input layer.

Since we are already here adding the second convolution layer, so we can simply remove that parameter. So, we are all set to move on to step3.

1. cnn.add(tf.keras.layers.Conv2D(filters=32, kernel\_size=3, activation='relu'))
2. cnn.add(tf.keras.layers.MaxPool2D(pool\_size=2, strides=2))

**Step3: Flattening**

In the third step, we will undergo flattening the result of these convolutions and pooling into a one-dimensional vector, which will become the input of a fully connected layer neural network in a similar way as we did in the previous section. instance of the **Flatten** class, such that Keras will automatically understand that this is the result of all these convolutions and pooling, which will be flattened into the one-dimensional vector.

So, we just need to specify that we want to apply flattening and to do this we will have to call once again the layers module by the Keras library by TensorFlow from which we are actually going to call the flatten class, and we don't need to pass any kind of parameter inside it.

1. cnn.add(tf.keras.layers.Flatten())

**Step4: Full Conversion**

In step 4, we are exactly in the same situation as before building a fully connected neural network. So, we will be adding a new fully-connected layer to that flatten layer, which is nothing but a one-dimensional vector that will become the input of a fully connected neural network. And for this, we will again start by taking a **cnn** neural network from which we are going to call the **add** method because now we are about to add a new layer, which is a fully connected layer that belongs to **tf.keras.layers**. But this time, we will take **a Dense** class followed by passing **units**, which is the number of hidden neurons we want to have into this fully connected layer and **activation function** parameter.

1. cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

**Step5: Output Layer**

Here we need to add the final output layer, which will be fully connected to the previous hidden layer. Therefore, we will call the Dense class once again in the same way as we did in the previous step but will change the value of the input parameters because the numbers of units in the output layer are definitely not 128. Since we are doing binary classification, it will actually be one neuron to encode that binary class into a 'cat' or 'dog'. And for the activation layer, it is recommended to have a sigmoid activation function. Otherwise, if we were doing multiclass classification, we would have used the SoftMax activation function.

1. cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

### **Part3: Training the CNN**

In the previous steps, we built the brain the, which contained in the eyes of the Artificial Intelligence and now we are going to make that brain smart with the training of CNN on all our training set images, and at the same time, we will evaluate our same model on the test set over the epochs. Now we are going to train our CNN over 25 epochs, and at each epoch, we will actually see how our model is performing on our test set images. This is a different kind of training as we did before because we always used to separate the training and evaluation, but here this will happen at the same time as we are making some specific application, i.e., computer vision.

**Compiling the CNN**

Now we are going to compile the CNN, which means that we are going to connect it to an optimizer, a loss function, and some metrics. As we are doing once again a binary classification, so we are going to compile our CNN exactly the same way as we complied our ANN model because indeed, we are going to choose once again **adam** optimizer to perform stochastic gradient descent to update the weights in order to reduce the loss error between the predictions and target. Then we will choose the same loss, i.e., the **binary\_crossentrophy** loss because we are doing exactly the same task binary classification. And then same for the metrics, we will choose **accuracy** metrics because it is the most relevant way to measure the performance of the classification model, which is exactly our case of CNN.

So, we will take our cnn from which we will be calling the compile method that will take as input the optimizer, loss function, and the metrics.

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

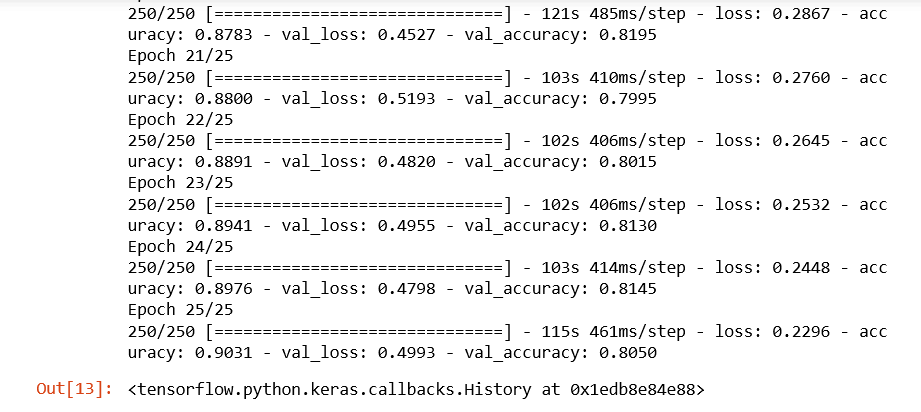
**Training the CNN on the Training set and evaluation on the Test set**

After the compilation, we will train the CNN on the training set followed by evaluating at the same time on the test set, which will not be exactly the same as before but will be somewhat similar. Basically, the first two steps are always the same, i.e., in the first step, we will take cnn followed by taking the fit method in the second step that will train the cnn on the training set. Inside it, we will pass the following parameters:

* The first parameter is the set, which is off course the dataset (**training set**) on which we are going to train our model, and the name for that parameter is simply X, created in part1.
* The second parameter is the difference with what we did before. So, it has to do, of course with the fact that we are not only training the CNN on the training set but also evaluating it at the same time on the test set. And that is what exactly our second parameter corresponds to, so we will be specifying here the **validation data** (test set), which is the set on which we want to evaluate our CNN.
* Lastly, the epochs parameter, which is the number of epochs. Here we are choosing 25 epochs to converge the accuracy not only on the training set but also on the test set.

1. cnn.fit(x = training\_set, validation\_data = test\_set, epochs = 25)

**Output**



From the image given above, it can be seen that we ended with **90%** of final accuracy on the training set and final accuracy of **80.50%**on the test set. Let's remind it again that if we had not done image augmentation preprocessing in part1, we would have ended up with an accuracy of around **98%** or even **99%** on the training set, which clearly indicates **overfitting** and lower accuracy here on the test set around **70%**. This is the reason why we insisted image augmentation is absolutely fundamental.

### **Part4: Making a single prediction**

In part4, we will make a single prediction, which actually consists of deploying our model on the two separate images of this single prediction folder for which our model will have to recognize for both the dog and cat, respectively. So, basically, we will deploy our CNN model on each of these single images, and we will hope that our CNN successfully predicts a dog as well as a cat. And for this, we will start with importing [**NumPy**](https://www.javatpoint.com/numpy-tutorial). Next, we will import a new module that we actually imported earlier, i.e., we imported the **ImageDataGenerator** from the image submodule of the preprocessing module of the **Keras** library. And in fact, what we are going to import now is that image module. But because we specifically imported something specific from that module, well, we need to import it again.

So, we will start with **Keras**, which we will help us to get access to the preprocessing module from which we will further import that image module. The next is, of course, to load that single image on which we want to deploy our model to predict if there is a cat or dog inside. We will create a new variable, i.e., the **test\_set** that will be initialized with loading the image on which we want to test out model from the same single prediction folder. It can be done by first calling the **image** submodule from which we will call the **load\_img** function, and inside this function, we will simply pass two arguments, i.e.,

Since we actually resized our images into the size target of (64, 64), whether it was for the training set or test set and we also specify it again while building the CNN with the same input shape, so the size of the image we are going to work with either for training the CNN or calling the predict method has to be (64, 64). So, in order to specify it here, we will enter our second parameter, which is the **target\_size.**

1. **import** numpy as np
2. from keras.preprocessing **import** image
3. test\_image = image.load\_img('dataset/single\_prediction/cat\_or\_dog\_1.jpg', target\_size = (64, 64))

But to make our first test \_set image accepted by the predict method, we need to convert the format of an image into an array because the predict method expects its input to be a 2D array. And we will do this with the help of another function of the image preprocessing module, i.e., **img\_to\_array** function, which indeed converts **PIL image** instance into a **NumPy array** that is exactly the format of array expected by the predict method. We will again use our image submodule from which we will call **img\_to\_array()**, and inside, it will take the test\_size image in PIL format that we are looking forward to convert it into the NumPy array format.

1. test\_image = image.img\_to\_array(test\_image)

Since the predict method has to be called on the exact same format that was used during the training, so if we go back into the preprocessing phase of both training set as well as the test set, we created batches of images. Therefore, our CNN was not trained in any single image; rather, it was trained on the batches of images. So, as we have an extra dimension of batch and we are about to deploy our model on a single image, then that single image still has to be into the batch even if we are going to have one image in the batch, it has to be into this batch so that the predict method of our CNN model can recognize the batch as that extra dimension.

Next, we will add an extra dimension, which will correspond to the batch that will contain that image into the batch, and it can be simply done by updating our test image by adding extra dimensions corresponding to batch. And the way to do it is with **NumPy** as the NumPy arrays can be easily manipulated, so we will first call the NumPy from which we will call this function that allows exactly to add a fake dimension, or we can say a dimension corresponding to the batch, which is called **expand\_dims** function inside of which we will input the image to which we want to add this extra dimension corresponding to the batch followed by adding an extra argument, i.e., where we want to add that extra dimension such that the dimension of the batch is always the first dimension to which we always give our first batch of images, and then inside of each batch we get the different images. So, it seems natural to have the batch as the first dimension and to specify this is exactly what we need to enter as a second argument, which is **an axis** that we have to set equal to zero. That is why the dimension of a batch that we want to add to our image will be the first dimension.

1. test\_image = np.expand\_dims(test\_image, axis = 0)

After this, we can call the predict method because, indeed, that test set image, which is not only in the right NumPy array but also which has the extra dimension corresponding to the batch, has exactly the right format expected by the predict method.

Therefore, we can create a new variable which will call result as it will actually predict our CNN model with the test image. Here we are not calling it prediction because it will only return or zero or one, which is why we are required to encode so as to represent 0 relates to cat and 1 is a dog. So, we will call our first result variable, which will actually be the output of the predict method called from our CNN. Inside the predict method, we will pass the **test\_image**, which now has the right format expected by that predict method.

1. result = cnn.predict(test\_image)

To figure out in between what relates to 0 and what narrates about 1, we will call either the **training\_set** or **test\_set** and then from which we will further call **class\_indices**, such that by printing this, we will get the right class\_indices. And with this, we indeed get that dog corresponds to 1 and cat relates to 0.

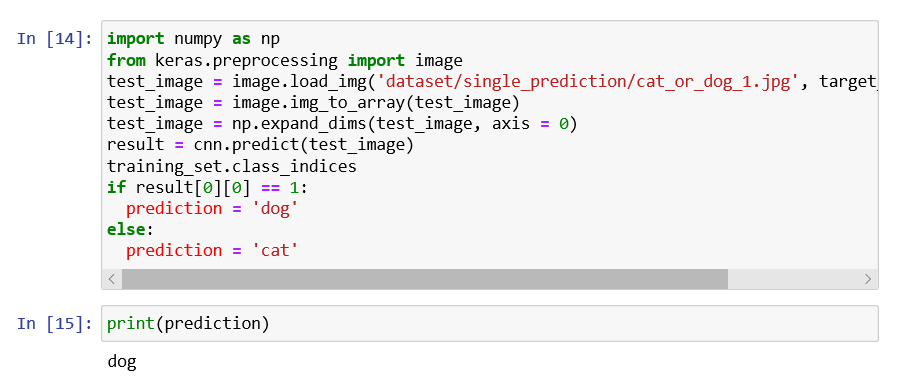
1. training\_set.class\_indices

In the end, when the two single predictions are made on these two single images, we will finish it with the if condition. Since we already know that result contains the outcome in batches because it was called on a test image that was into a batch, so results also have a batch dimension, and we are going to get access to the batch.

After this, inside the batch, we are going to get access to the first element of the batch that corresponds to the prediction of that same **cat\_or\_dog\_1** image. As we are dealing with a single image, so a single prediction is needed, and to get that, we will need to get inside the batch of index zero, the first and only prediction once again, which has a [0] index. So, that is how we get our prediction by first accessing the batch followed by accessing the single element of the batch, and if that prediction equals to one, then we already know that it corresponds to the dog, then we will create a new variable which we will call as prediction and will set that prediction variable equals to the dog. Likewise, in the else condition, if the result prediction equals to 1, then the prediction will be a cat. Now we will wrap it up by simply printing the prediction.

1. **if** result[0][0] == 1:
2. prediction = 'dog'
3. **else**:
4. prediction = 'cat'
6. print(prediction)

**Output**

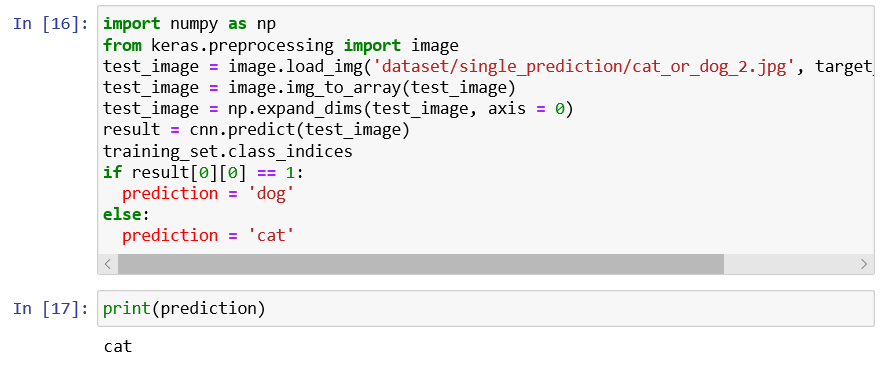


We can see our Convolution Neural Network predicted that there is a dog inside the image. So, it can be concluded that our first test is passed successfully.

Now we will check for the other image which is of the cat, so for that we will need to deploy our model on this single image and check that indeed, our CNN returns a cat. To do this, we need to change the name here, i.e. **cat\_or\_dog\_2.jpg** and then play this cell again by clicking on the Run button.

1. **import** numpy as np
2. from keras.preprocessing **import** image
3. test\_image = image.load\_img('dataset/single\_prediction/cat\_or\_dog\_2.jpg', target\_size = (64, 64))
4. test\_image = image.img\_to\_array(test\_image)
5. test\_image = np.expand\_dims(test\_image, axis = 0)
6. result = cnn.predict(test\_image)
7. training\_set.class\_indices
8. **if** result[0][0] == 1:
9. prediction = 'dog'
10. **else**:
11. prediction = 'cat'
12. print(prediction)

**Output**



So, it's clear now that our CNN model is successful in predicting cat in the output of the console. Hence our CNN got all the answers correct.

**4. SYSTEM DESIGN**

**4.1 System Architecture**



Fig4.1 .System Architecture

WORKING:

1. Data Collection: First of all, the information for different Websites and Social Media applications based on certain parameters is extracted data.

2. Pre-processing: Then we will apply various pre-processing steps such as Noise removal, resizing, binary conversion and Gray scaling in order to make our dataset proper.

3. Noise removal: Noise is removed from the input video. In image processing, the key process for denoising is filtering. Generally average filters, median filters, Wiener filters and Kalman filters are utilized to reduce noise.

4. Resizing: Image resizing is necessary when we need to increase or decrease the total number of pixels, whereas remapping can be done when we are adjusting for lens distortion or rotating an image.

5. Binary conversion: A binary image is one that holds the pixels that can have any one of precisely two colors, classically black and white. Binary images are also well known as bi-level or as two-level. This means that each and

Every single pixel is put in storage as a solitary bit – i.e. in value of 0and 1.

6. Gray scaling: Gray-scaling is the method of transforming a continuous-tone image to an image that a computer can manipulate effortlessly.

7. Segmentation: Image segmentation is the significant process in which isolation of a digital image into multiple segments is carried out i.e. (sets of pixels, also recognized as image objects).

6. Data Training: We compile artificial as well as real time using online news data and provide training with any machine learning classifier.

8. Feature extraction: Feature extraction is a part of the dimensionality decrease procedure, in which, an initial set of the raw data is separated and compact to more controllable groups.

9. Classification: Classification is the method of sorting and labeling groups of pixels or vectors with in an image based on definite rules and instruction

10. Data Training: We gathered artificial as well as real time using social media data and provide training with any machine learning classifier.

11. Testing with machine learning: We give testing dataset to system and apply machine learning algorithm to detect the activity accordingly.

12. Analysis: We determine the accuracy of proposed system and estimate with other existing systems

**4.2 Data Flow Diagram (DFD)**

A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on the flow of information, where data comes from, where it goes and how it gets stored.

**Data Flow Diagram Levels**

Context Diagram. A context diagram is a top level (also known as "Level 0") data flow diagram. It only contains one process node ("Process 0") that generalizes the function of the entire system in relationship to external entities.

**DFD Layers.** Draw data flow diagrams can be made in several nested layers. A single process node on a high level diagram can be expanded to show a more detailed data flow diagram. Draw the context diagram first, followed by various layers of data flow diagrams.

**DFD Levels.** The first level DFD shows the main processes within the system. Each of these processes can be broken into further processes until you reach pseudo code.

**0-Level DFD**

It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows. Then the system is decomposed and described as a DFD with multiple bubbles.

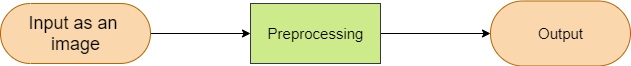


Fig.4.2.1 0-Level DFD

**1-Level DFD**

In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into sub processes.

Fig.4.2.2 1-Level DFD



**2-Level DFD**

2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.



Fig.4.2.3 2-Level DFD

**4.3 UML Diagrams**

The UML stands for Unified modeling language, is a standardized general-purpose visual modeling language in the field of Software Engineering. It is used for specifying, visualizing, constructing, and documenting the primary artifacts of the software system. It helps in designing and characterizing, especially those software systems that incorporate the concept of Object orientation. It describes the working of both the software and hardware systems.

The UML has the following features:

1. It is a generalized modeling language.
2. It is distinct from other programming languages like C++, Python, etc.
3. It is interrelated to object-oriented analysis and design.

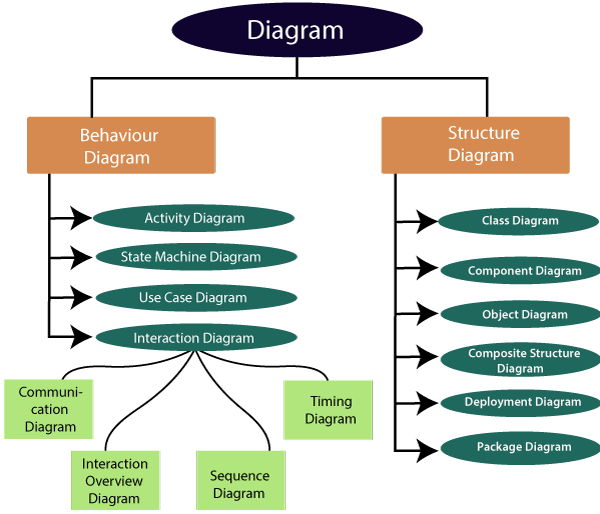


Fig.4.3.1 Classification of UML Diagrams

**4.3.1 Use Case Diagrams**

It represents the functionality of a system by utilizing actors and use cases. It encapsulates the functional requirement of a system and its association with actors. It portrays the use case view of a system.

In Use Case Diagram we shows communication between user and system,In the first step we done registration by user and then login .take the video as an input then detect the person,then do prepocessing on Detected images,system use CNN Model for classifiy the image,after the classification process suspicious activity is detected

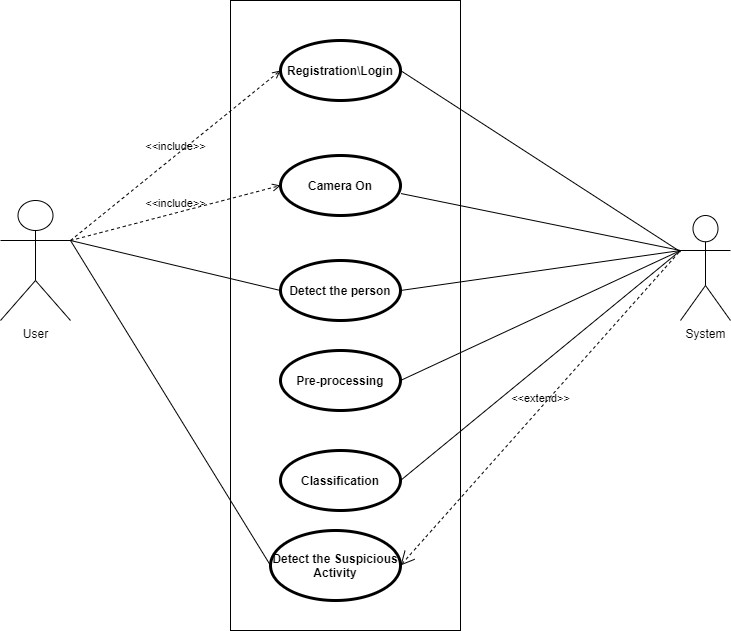


Fig.4.3.2 Use Case Diagram

**4.3.2 Activity Diagram**

It models the flow of control from one activity to the other. With the help of an activity diagram, we can model sequential and concurrent activities. It visually depicts the workflow as well as what causes an event to occur.

In activity diagaram we shows communication between user and system,In the first step we done registration by user and then login .take the video as an input then detect the person,then do prepocessing on Detected images,system use CNN Model for classifiy the image,after the classification process suspicious activity is detected

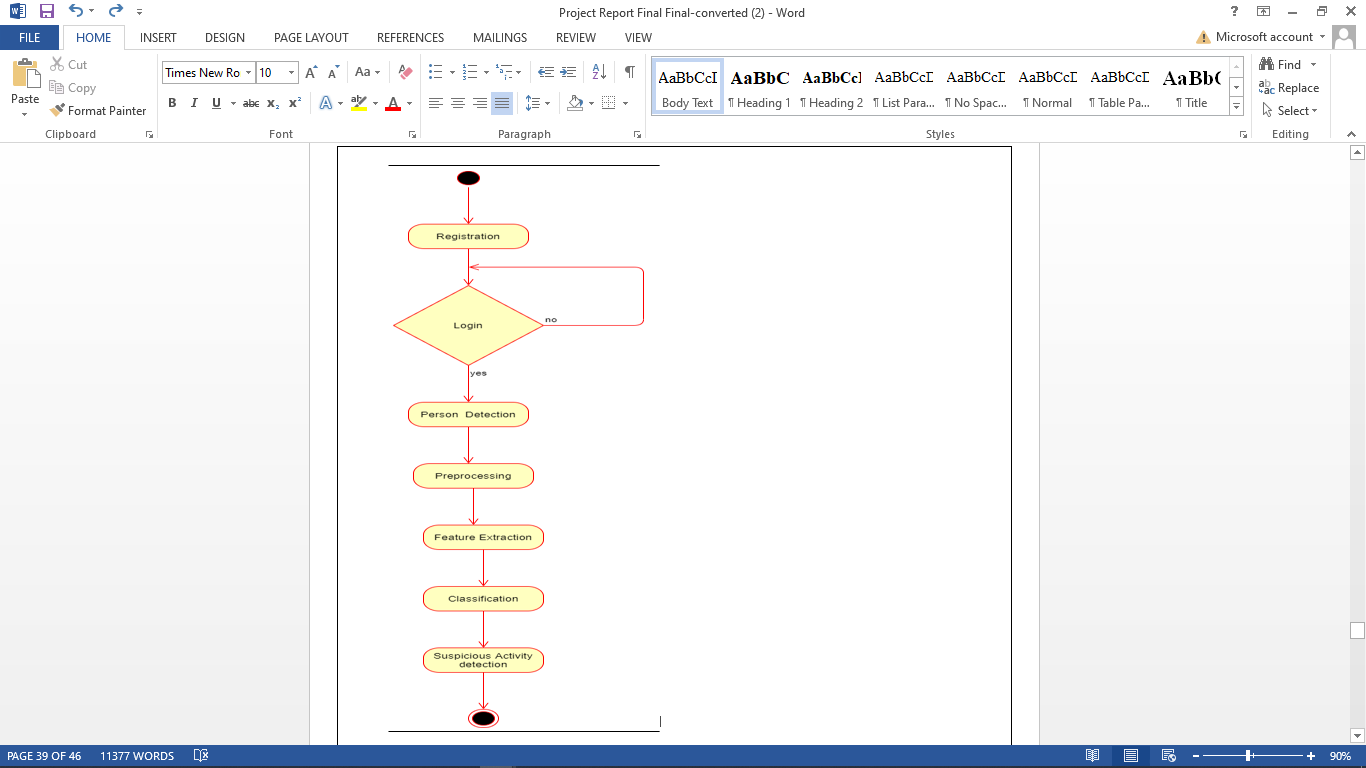


Fig.4.3.3 Activity Diagram

**4.3.3 Class Diagram**

Class diagrams are one of the most widely used diagrams. It is the backbone of all the object-oriented software systems. It depicts the static structure of the system. It displays the system's class, attributes, and methods. It is helpful in recognizing the relation between different objects as well as classes.we created four classes Camera,User,Notification,System.In camera classes we use the functions that is dectect person,detect suspicious activity

In user class we used user id,username,password as a string functions.,and we can do edit and add functions.In notification class we use Notification fumction,In system class we detect the suspicious Activity and person.

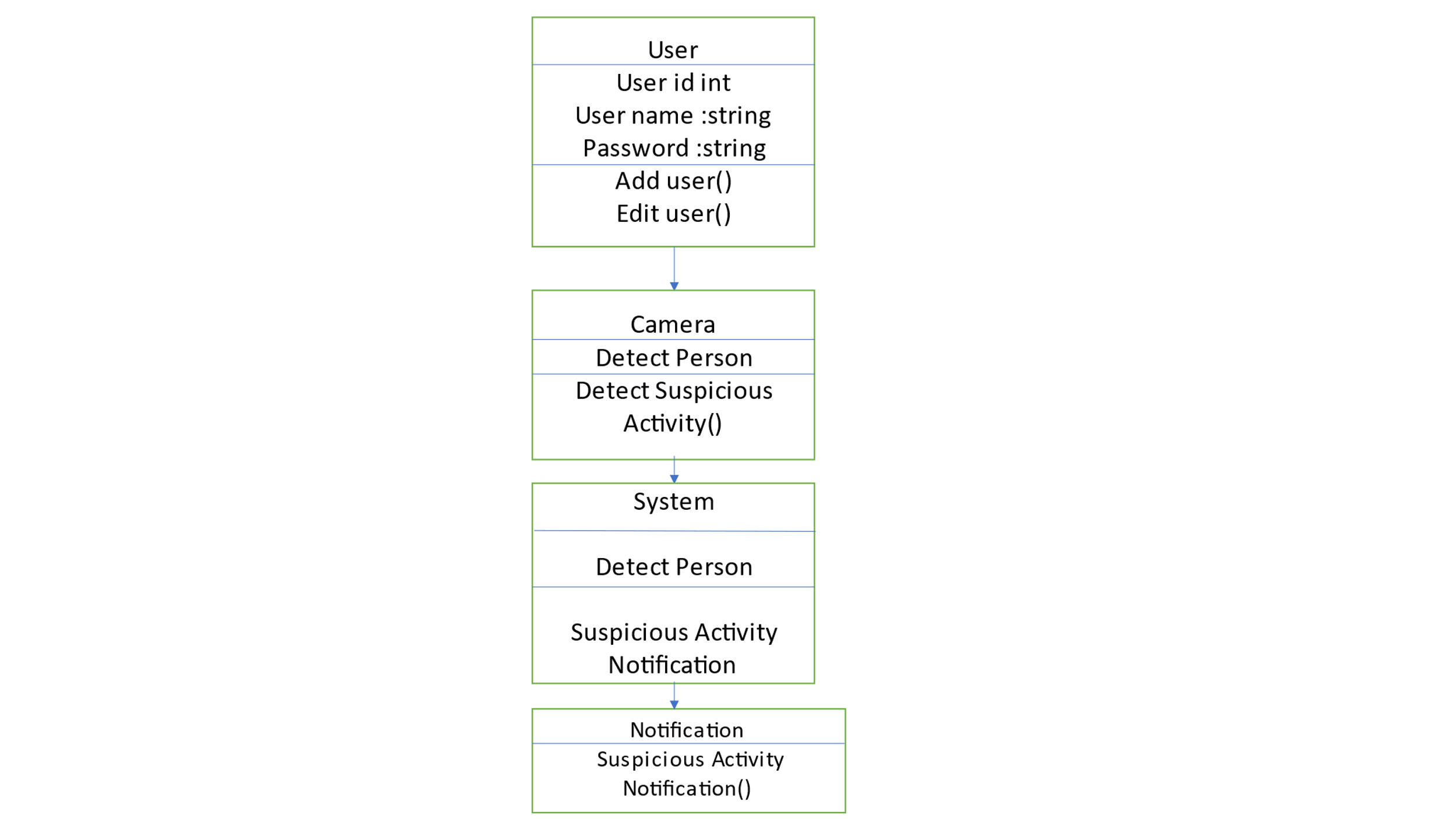


Fig.4.3.4 Class Diagram

**4.3.4 Sequence Diagram**

It shows the interactions between the objects in terms of messages exchanged over time. It delineates in what order and how the object functions are in a system.

WORKING PRINCIPLE:

STEP 1: User have to register first so for registration first enter the User name combination of any alphabetical char and any numbers up to 10 char size.

STEP 2: User need to set a password using any number, char and special char up to 8 char size.

STEP 3: After registration click on login button.

STEP 4: Give the Username and Password that previously created.

STEP 5: Click on submit button.

STEP 6: After step 5 User will redirected to the Video submission page. User need to click on Select Video Button.

STEP 7: After Clicking on Select Video Button user will go to the database were all the video are saved user need to select on of these videos.

STEP 8: After selecting the video Click on ok Button. the video will be open in new window and it will start showing the frames numbers and if the suspicious activity is detected then it will show red colour of frames

STEP 9: If user want to exit from the Application so use have to Click on Exit Button.

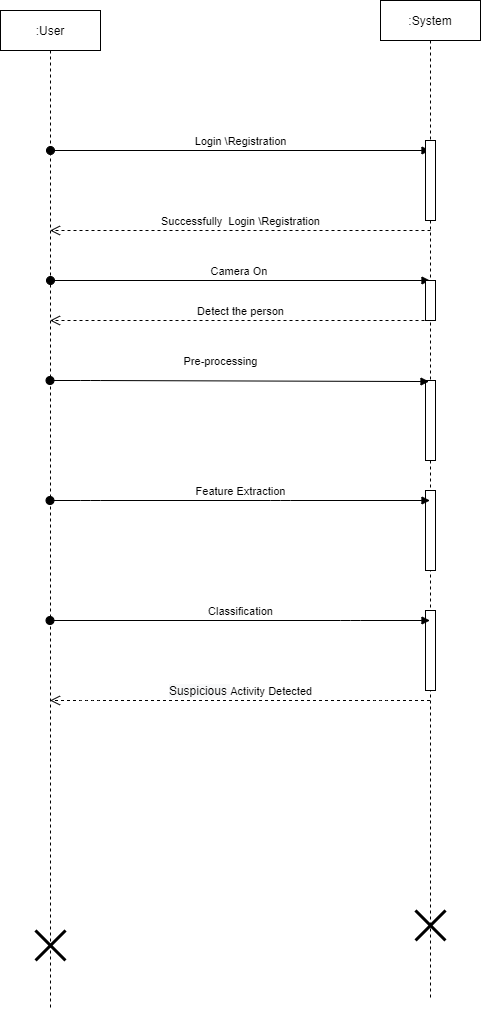


Fig.4.3.5 Sequence Diagram

**6. OTHER SPECIFICATIONS**

### ADVANTAGES

### Used in Surveillance at places like **Malls,** Railway **Stations,** And Hotel.

### Risk of Fighting or Shooting Attack.

### Accident Detection.

### Fire Detection

### Easily detect the suspicious activity.

### Decrease the number of crime.

### LIMITATIONS

* If the training not get successful or get interrupt because of any reason then system can not work proper.
* If the accuracy of training less then system can not work properly.

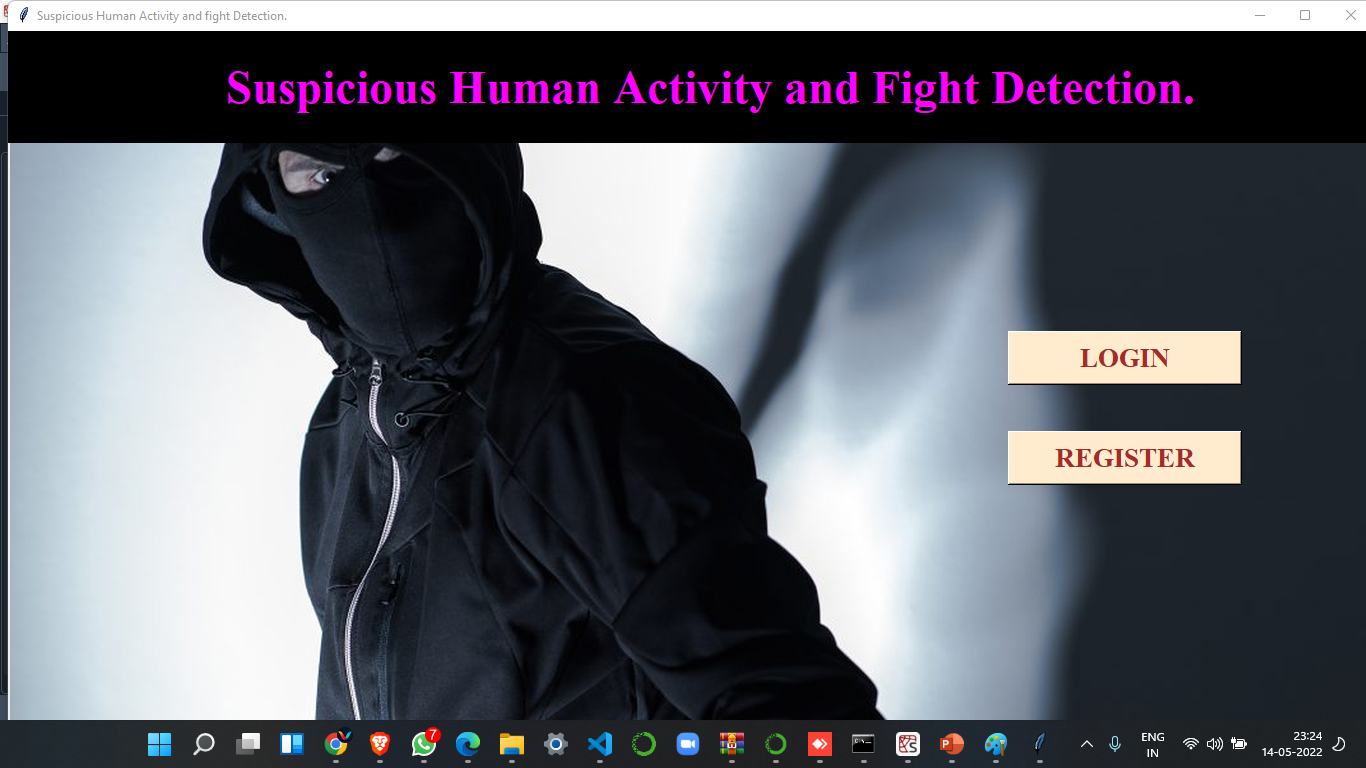
### APPLICATIONS

1. Crowd areas
2. Hospitals
3. Offices
4. Roads

**Result**

Step 1:

This is the first Main page of the System Here User want to login or Register to the System.



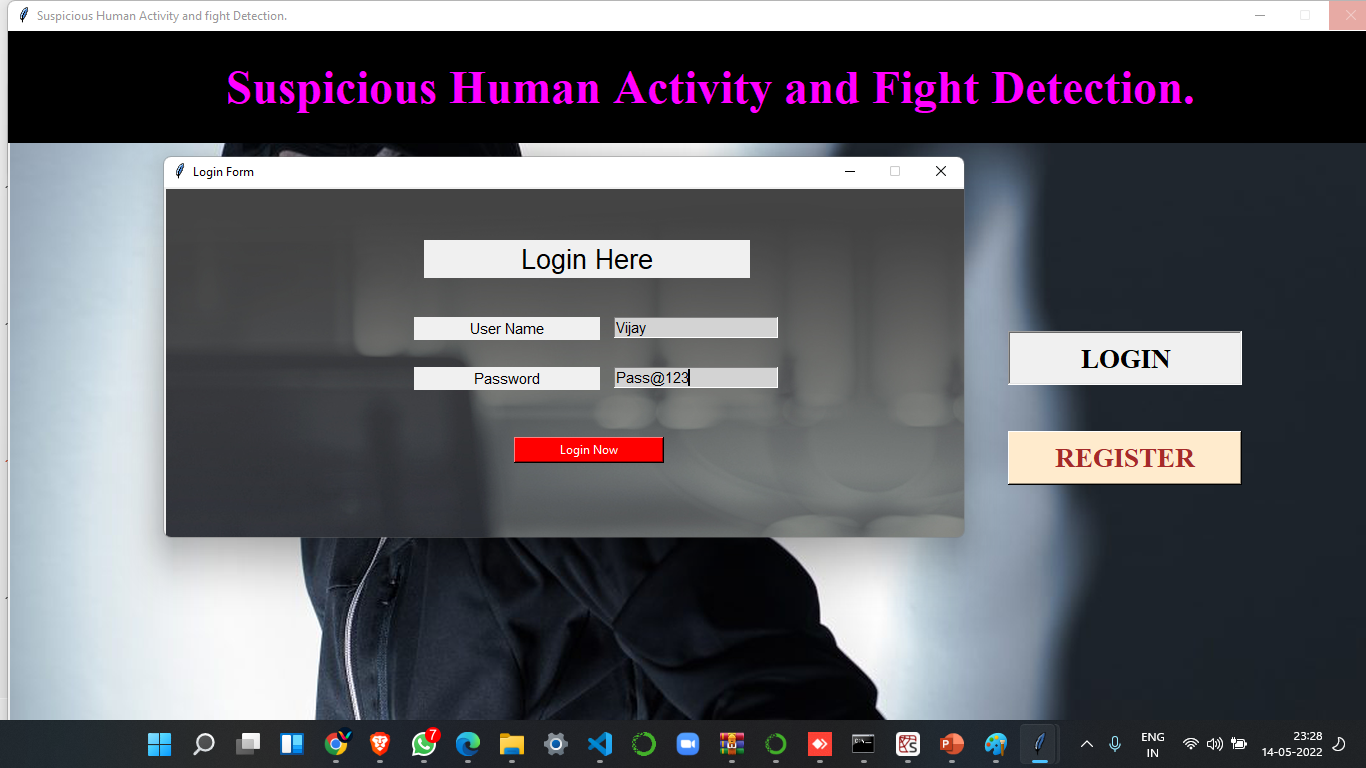
Step 2:

The user is register first to the system by click the Register Button and fill the required details in the form and after that submit it .



Step 3:

After Successful registration user is going to login page user must know the user name and password which is used at the time of registration.



Step4:

After Successful registration user want to continue in the system to press the video selection button otherwise exit to the system.



Step5:

After selecting the Video button User need to select the particular video and open it.



Step6:

After open video the video is playing smoothly and from video in background framing is created from the video.



Step7:

When the any suspicious thing is recognizing by the system it will show on the output screen is label and the frame in the red color.



Step8:

After that system admin or user need to take the particular action against the which suspicious activity is found in the location.



# Conclusions & Future Work

**7.1 Conclusion**

A system to process real-time CCTV footage to detect any suspicious activity will help to create better security and less human intervention. Great strides have been made in the field of human suspicious Activity, which enables us to better serve the myriad applications that are possible with it. Moreover, research in related fields such as Activity Tracking can greatly enhance its productive utilization in several fields system that gives CCTV cameras the ability to detect suspicious activity, without human intervention. The goal of this paper is achieved which was to generate alert on detection of suspicious activity. It is achieved by taking real-time videos from CCTV as an input and pass it to the CNN model and predict ‘Fight’, ‘Fire’ or ‘Explosion’-In’ in the store and notify it to the owners as soon as it occurs.

The collected images from Google are split into training and testing set in a ratio of 80:20. An accuracy of 89% on the testing set was seen in the result. Various other measures like precision, recall, f1-score were also considered. The only limitation of the system is the flickering effect, which can be further minimised by making a proper selection of a subset of frames from the queue. From the overall observed results

it can be said that the model achieved better accuracy than the previously tested results and can be used for detecting suspicious activity of customer. Finally, it is concluded that providing a system that determines customer behavior and detect suspicious activities without human intervention is a huge revolution in today’s surveillance system

.

**7.2 Future Scope**

In our model we take number of images form video if video is large then it will take more time to create frames. In future we try to improve accuracy and make sure that it take less time to capture human suspicious activity. In future we will add more images data set to detect suspicious activity and weapon

We implement this system on android.

Also we can try to improve accuracy.

**References**

* Eralda Nishani, Betim Cico : “Computer Vision Approaches based on DeepLearning and Neural Networks” Deep Neural Networks for Video Analysisof Human Pose Estimation- 2017 6th MEDITERRANEAN CONFERENCEON EMBEDDED COMPUTING (MECO), 11-15 JUNE 2017, BAR, MONTENEGRO
* Naimat Ullah Khan , Wanggen Wan : “A Review of Human Pose Estimation from Single Image”- 978-1-5386-5195-7/18/ 2018 IEEE
* Qiuhui Chen, Chongyang Zhang, Weiwei Liu, and Dan Wang, ”Surveillance Human Pose Dataset And Performance Evaluation For Coarse-Grained Pose Estimation”, Athens 2018.
* Baole Ai, Yu Zhou, Yao Yu : “Human Pose Estimation using Deep Structure Guided Learning”-978-1-5090-4822-9/17 2017 IEEE DOI 10.1109/WACV.2017.141
* Zhe Cao, Tomas Simon, Shih-En Wei, Yaser Sheikh The Robotics Institute, Carnegie Mellon University ”Real time Multiperson 2D Pose Estimation using part affinity fields” - 1063-6919/17 2017 IEEE DOI 10.1109/CVPR.2017.143
* Hanguen Kim, Sangwon Lee, Dongsung Lee, Soonmin Choi, JinsunJu and Huyun Myung “Real-Time Human Pose Estimation and Gesture Recognition from depth Images Using Superpixels and SVM classifier.”- Sensors 2015, 15, 12410-12427; doi:10.3390/s150612410