Developing AI-Based Surveillance Software

A Thesis

Submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

Submitted by:

Dipayan Pal

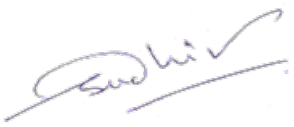
Indian Institute of Science Education and Research, Pune

Under the supervision of:

Mr. Sudhir Kumar(Coriolis Technologies)
Co-Supervisor: Mr. Rohan Nandode (Coriolis Technologies)
TAC Member: Dr. Leelavati Narlikar (IISER Pune)

Certificate

This is to certify that this dissertation entitled **Developing AI-Based Surveillance Software** towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Dipayan Pal at Indian Institute of Science Education and Research, Pune under the supervision of Sudhir Kumar, Member of Technical Staff, Coriolis Technologies, Pune, during the academic year 2022-2023.



Sudhir Kumar
Member of Technical Staff
Coriolis Technologies

Committee:

Supervisor: Sudhir Kumar (Coriolis Technologies)
Co-Supervisor: Rohan Nandode (Coriolis Technologies)
TAC Member: Dr. Leelavati Narlikar (IISER Pune)

भारतीय विज्ञान शिक्षा एवं अनुसंधान संस्थान पुणे
INDIAN INSTITUTE OF SCIENCE EDUCATION AND RESEARCH PUNE

IISER PUNE

Declaration

I hereby declare that the matter embodied in the report entitled **Developing AI-Based Surveillance Software**, are the results of the work carried out by me at the Department of Data Science, Indian Institute of Science Education and Research, Pune, under the supervision of Sudhir Kumar (Coriolis Technologies), and the same has not been submitted elsewhere for any other degree.

Dipayan Pal

This Thesis is Dedicated to

My parents: Gita Sen(Mother) and Samir Pal(Father)
&
2 of my best friends without whom this project
wouldn't have been completed:
Sahil Mulewar and Deepesh Khushwani

Academic Acknowledgements

Being a naive kid from a small city like Agartala in Tripura, IISER has taught me a lot of things. It has shaped my personality and helped me a lot with my transition from a clueless teenager to a responsible adult. There were many ups and downs on the path, but IISER always tried its best to take good care of us.

First and foremost, I would like to thank my supervisor, Mr. Sudhir Kumar. He has guided me through everything, starting from explaining the ins and outs of the project, suggesting what to do next when I got stuck in a dead end, and overall providing me with a good direction for the project. He has also constantly motivated me to keep going when things looked very tough, and without his constant support, I wouldn't have been able to complete this project.

I can't thank Prof. Leelavati Narlikar enough for all the support she has provided me, both academically and emotionally. Doing an industrial project in IISER seemed like a difficult experience for me, but she always assured me, helped me clear out a lot of my thoughts, and motivated me to move forward with my project. Prof. Leelavati put me in touch with Mr. Sudhir, which ultimately led me to this project. I also pursued a semester project with her in my last semester, and she was the best supervisor one could ask for. She started from the basics and helped me develop a scientific temperament throughout the project.

It is an understatement to use the word blessed when I look back at the opportunities I had and the teachers I had the opportunity to learn from. It is because of them and their support and affection that I have been able to learn whatever I have learned till now. All the labs and webinars have truly been an enriching experience for me. It has helped me grow a lot, both professionally and personally.

I would like to thank my co-supervisor, Mr. Rohan Nandode, for his constant inputs throughout the project.

I would also like to thank my CEO, Mr. Basant Rajan, for giving me this wonderful opportunity.

I would also like to thank all my colleagues at Coriolis Technologies, who helped me a lot and made me feel like family there. In particular, I would like to thank Hashim, who was my collaborator at the start of the project and helped kickstart my project.

Personal Acknowledgements

The Fifth Year Master's Thesis at IISER Pune culminates our 5-year journey of BS-MS. I am writing this as I am tearing up remembering all the events that have happened over the past 5 years. I wish I could repeat all these experiences, but alas, that's not how things work.

When I landed at IISER 5 years ago with a lot of dreams to become a physicist and equal bits of anxiety and imposter syndrome, I couldn't have imagined, in my wildest dreams, what my journey at IISER would turn out to be. It was a roller coaster ride from experiencing my first global pandemic (hopefully the last one) to changing my course completely from Physics to Data Science. IISER also helped me a lot with my personal exploration, a major thanks to all the extra-curricular activities, especially the Satrangi Club.

I wouldn't have been able to graduate from IISER without the constant support and companionship of two of my best friends, Deepesh Khushwani and Sahil Mulewar. I found myself in a full blown "mental breakdown" at the start of my fifth year. During those tough times, they have been by my side throughout my journey, and I honestly can't thank them enough for what they did.

Deepesh is honestly one of the funniest persons I have met, and I thank Deepesh for making me fall in love with music once again. I wouldn't forget all the countless all-nighter discussions that we had. Best times indeed!

I can't thank Sahil enough for the amount of meme materials he has provided through our IISER journey. One of the most quirky, charming, and good-hearted people I have seen. But your hair honestly doesn't look good.

Finally, my friends: Prantik, who was always there to help me whenever I needed him and who sparked my interest in reading novels, and Lubdhak, who was one of the best roommates I could ask for, and yes, you will never retire from Kalpa.

It is hard to resist the temptation to write a huge acknowledgement for this gigantic journey, but I would stop here.

IISER gave me many helpful seniors who always treated me as their equal, with no hierarchy at all, and also many cheerful juniors who were some of the sweetest people to talk to.

With this, I would end my love letter to all these 5 wonderful years at IISER. Thanks to everyone for bearing with me, and I will cherish all these memories for the rest of my life.

Abstract

In this work, we developed an AI-based surveillance system to be used by businesses to improve public safety, security, and law enforcement efforts. The system has numerous potential applications, including lost item retrieval, efficient emergency response, facial biometric verification, and more. Our software utilises advanced algorithms and state-of-the-art technologies to analyse videos from CCTV cameras and accurately identify individuals and objects in real-time.

Despite recent advances in face recognition, small face recognition at scale remains a significant challenge. To address this, we utilized Insightface models for face recognition and achieved an accuracy of 75% with the system processing 15 frames per second.To optimize the system, we implemented several infrastructure changes, including shifting from Elasticsearch to Opensearch, which reduced the loading time of our UI interface from 2 minutes to 10-15 seconds. The entire system is highly scalable and fault-tolerant, capable of processing 60 million images per day due to its implementation in Kubernetes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Softwares used

1.1.1 Docker

Docker is an open-source platform that allows developers to create, publish, run, update, and manage containers.

Containers are lightweight and portable software packages that run in an isolated environment within a system that includes everything necessary to run a piece of software, including the code, runtime, libraries, environment variables, and system utilities. Containers provide a consistent and reproducible execution of our codes, making it simpler to deploy and run programs in a consistent manner across many environments.

Docker is the most popular containerization platform, and it offers an intuitive interface for constructing, deploying, and managing containers. With Docker, you can bundle your application and its dependencies into a container image that can be deployed and executed on any server with Docker installed.

Due to all these advantages, we have extensively used Docker in our project to make containers for our applications.[1]

1.1.2 Kubernetes

Kubernetes is an open-source framework for automating the deployment, scaling, and management of containerised applications.

Kubernetes offers several benefits for the deployment and management of container-

ized applications, including:

1. **Scalability**: Kubernetes makes it easy to scale applications up or down as needed, making it well-suited for large-scale, dynamic workloads.
2. **Portability**: Kubernetes provides a standard environment for delivering applications, making it simple to migrate them between various environments and cloud platforms.
3. **Resilience**: Kubernetes self-healing features make it simpler to maintain the availability and dependability of applications in the case of node outages.
4. **Resource management**: Kubernetes has strong resource management features, making it simpler to guarantee that applications have the necessary resources to function efficiently.
5. **Automation**: Kubernetes provides a uniform platform for automating the deployment, scaling, and administration of containerized applications, therefore facilitating the management of complex applications and services.

All these advantages compelled us to use Kubernetes as the backbone of our infrastucture, and all of our containers were deployed using Kubernetes.[2]

1.2 Infrastructure

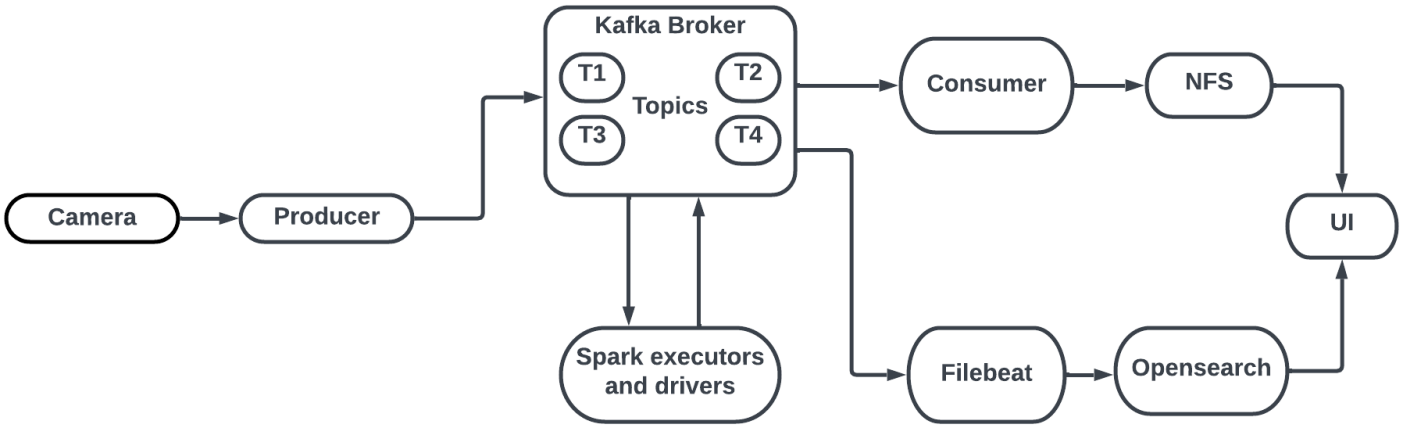


Figure 1.1: Flowchart of the infrastructure that we used

1.2.1 Camera

These are CCTV cameras installed in the Coriolis premises that capture images at the rate of 24 frames per second. Currently, we have deployed 32 cameras, which are generating on the order of 60 million images per day.

1.2.2 Producer

A producer in Apache Spark refers to a process or system that creates data and saves it in a data source that Spark can read from, such as a Kafka topic. A producer in Apache Spark can be thought of as the source of data that Spark processes and analyzes. This information may be gathered from a variety of sources, including logs, sensor data, and user interactions.

In our case, the producer takes the data that is generated from the cameras and stores the data to a Kafka topic.

A producer is responsible for writing data to one or more topics in a Kafka cluster in Apache Kafka. Spark may then ingest, process, and analyse this data. By utilising a distributed data processing framework such as Apache Spark, it is feasible to manage extraordinarily huge amounts of data in real-time and provide scalable and efficient insights and analysis.[3]

1.2.3 Kafka

Apache Kafka is a widely used distributed, scalable, and fault-tolerant message broker for real-time data pipelines and streaming applications.

It can be used in conjunction with Apache Spark where Spark may take data from one or more Kafka topics, analyse the data using the Spark framework in parallel, and then publish the findings back to another data source or utilise them for additional analysis.

Apache Kafka is extremely scalable and is capable of processing hundreds of thousands of messages per second. It is also fault-tolerant and can recover automatically from faults, making it a dependable and robust data streaming system. By combining Apache Spark with Apache Kafka, we can construct scalable, real-time data pipelines for processing massive amounts of data and gaining real-time insights from that data. [3]

1.2.4 Consumer

A consumer in Apache Spark is a process or system that reads and processes data from a data source, such as a Kafka topic. A consumer in Apache Spark is a process or system that reads data from a source, such as a Kafka topic, and processes or analyses that data. A consumer is responsible for consuming data from one or more topics in a Kafka cluster in Apache Kafka.

In our case, consumer consumes the data from the kafka topic and, through a Python script it applies some transformations to the images that makes them easier to apply our ML models to. Finally, the python script saves the images on our distributed NFS servers. [3]

1.2.5 Spark

Apache Spark is an open-source, distributed computing system built for analysing massive amounts of data. One of the primary benefits of Apache Spark is its capacity to process data in parallel over a cluster of computers, which makes it significantly quicker than conventional data processing systems.

The Spark's architecture is based on the master-slave model. Its cluster is composed of a single master and many slaves.

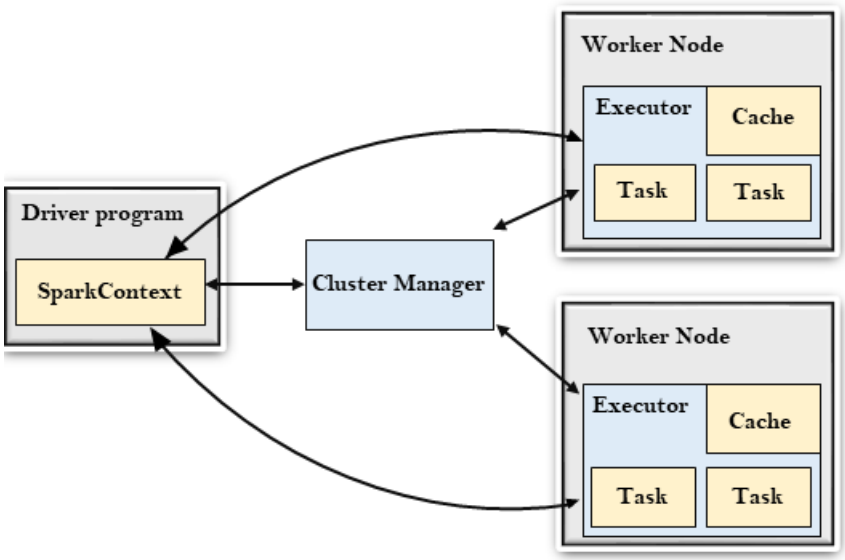


Figure 1.2: Spark architecture taken from [4]

The main features of the Spark architecture are:

1. **Driver**: The word "driver" in Apache Spark refers to the process that executes the primary function of a Spark application and creates a SparkContext. SparkContext serves as the entry point for all Spark applications and coordinates the execution of tasks throughout the cluster. The driver process is accountable for splitting the application into tasks and allocating those tasks to the executors.
2. **Cluster manager**: A cluster manager in Apache Spark is a component responsible for allocating Spark executors' resources, such as CPU and memory. Additionally, the cluster manager is responsible for initiating and stopping executors as required and controlling the cluster's overall health.
3. **Worker Node**: A worker node in Apache Spark is a machine in a cluster that executes one or more Spark executors. The responsibility of worker nodes is to execute the tasks given to them by the driver and return the results to the driver.
4. **Executors**: The term "executor" refers to the processes that execute the tasks assigned to them by the driver and operate on the worker nodes of a Spark cluster. Executors are responsible for carrying out the duties allocated to them and reporting the outcomes to the driver. Executors are also responsible for caching data in memory and providing it to the tasks as needed. [5]

1.2.6 Filebeat

Filebeat is a log data shipper for the Elastic Stack. It is used to collect and centralize log data from various sources and send it to Elasticsearch.

Filebeat is designed to be lightweight, scalable, and efficient, which makes it suitable for gathering log data from large-scale, distributed systems. It is configured to gather logs from a range of sources, including system logs, application logs, and custom logs, and runs as a process on any server that needs to submit log data.

After Filebeat has gathered the log data, it can be transmitted to the specified output destination for additional processing and analysis.

1.2.7 Opensearch

Opensearch is a distributed, RESTful, open-source search and analytics engine built to manage massive volumes of data. It is a Lucene-based search engine used for a number of purposes, including full-text search, analytics, and logging.

Opensearch is well-suited for large-scale, data-intensive applications as it is extremely scalable and can handle millions of search requests per second.

Opensearch delivers a variety of advanced tools for data analysis and visualisation in addition to search functionality. It also interfaces with other technologies, such as Filebeat and Opensearch-Dashboard, to provide an end-to-end solution for log analysis and data visualisation. [6]

1.2.8 UI

UI stands for "User Interface". It refers to the website we had created to visualise the results of our entire data processing, starting from taking images with cameras up untill applying the ML models to those images. Here are some of the screenshots from our web interface to show what the final product looks like.

Chapter 2

Insightface details

For our project, we have used Insightface as the base model for our face analysis. We have selected the default **Buffalo-l** model under the insightface framework.

2.1 What is Insightface?

InsightFace is a framework for face identification and analysis based on deep learning. It was created by the research team at Megvii Technology, the industry leader in computer vision and artificial intelligence. InsightFace offers a collection of pre-trained models for face recognition, face detection, and face alignment, as well as training and fine-tuning tools for these models using custom data.

Deep convolutional neural networks (CNNs) are used by InsightFace to conduct facial recognition and analysis tasks. The framework has a variety of models, including the Buffalo model, which is a very accurate and efficient model for large-scale facial recognition.

InsightFace is optimised for a variety of real-world applications, such as face identification in films, photographs, and live broadcasts. It is also meant to be extremely efficient and scalable, making it appropriate for usage in real-time applications and systems with limited computing resources. [7] [8] [9] [10] [11] [12] [13] [14]

2.2 Buffalo models

Buffalo models are a range of models included under the Insightface framework that we used for face recognition. Buffalo models can perform face recognition on a large number of images accurately and efficiently.

2.2.1 Architecture

The architecture of the Buffalo model is based on a network called MobileNetV3. This network is lightweight and efficient, making it appropriate for real-time applications like facial recognition. The Buffalo model optimises MobileNetV3 architecture, particularly for facial recognition applications.

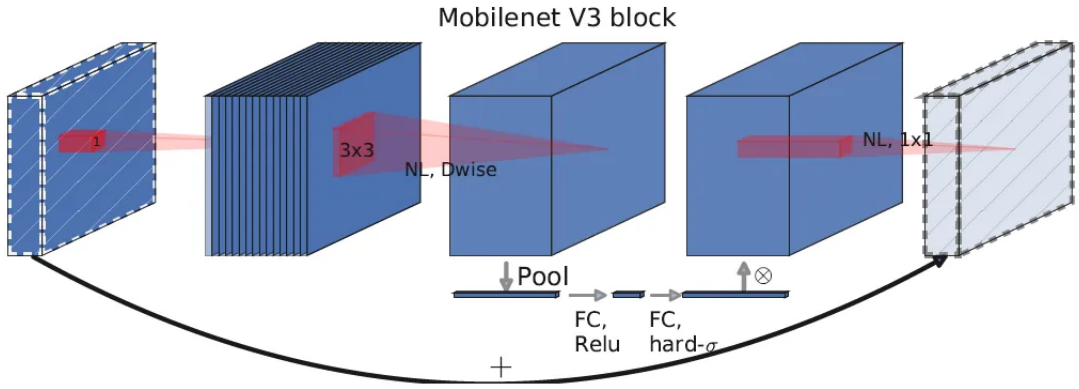


Figure 2.1: Mobilenetv3 architecture taken from [15]

The Buffalo model performs facial recognition using a mix of depth-wise separable convolutions and inverted residuals. These layers enable the network to efficiently analyse massive amounts of data without losing precision.

In addition, the Buffalo model features a secondary branch. This is a distinct portion of the network that delivers extra training-related information. This enhances the network's precision and makes it more resilient.

Finally, the Buffalo model employs a loss function called ArcFace that is tailored for face recognition applications.

The ArcFace loss function can be expressed by the following expression: [16]

$$L_{arcface} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cdot\cos(\theta_{y_i}+m)}}{e^{s\cdot\cos(\theta_{y_i}+m)}+\sum_{j=1,j\neq y_i}^{n}e^{s\cdot\cos\theta_j}} \tag{2.1}$$

where:

N is the batch size
s is the scaling factor
θ_{y_i} is the angle between the input feature vector and the weight vector of the true class y_i
m is the additive margin
n is the total number of classes

The goal of this loss function is to maximize the cosine similarity between the input feature vector and the weight vector of the true class while minimizing the similarity to other classes. The scaling factor and additive margin are hyper-parameters that control the margin between classes and the magnitude of the features, respectively.

This loss function improves the network's accuracy and resilience by increasing the inter-class variance and decreasing the intra-class variance.

The basic architecture of the buffalo-l model follows the following broad pathways:

1. **Preprocessing**: The supplied image is preprocessed in order to align the face and change its size and aspect ratio. This preprocessing phase is essential for ensuring that the network can efficiently process the picture and extract pertinent information.

2. **Convolutional Layers**: Following preprocessing, the input picture is passed into the first convolutional layer of the Buffalo model. This layer applies filters to the input picture to extract characteristics and identify patterns. The layer's output is subsequently transmitted to the subsequent layer.

3. **Depthwise Separable Convolution Layers**: Next, a sequence of depth-wise separable convolutional layers are applied to the output of the convolutional layer. In these layers, a single filter is applied to each input channel, allowing the network to more effectively process the data. The outputs of these layers are then mixed using pointwise convolution to produce a single output.

4. **Inverted Residual Layers**: The output is processed by a sequence of inverted residual layers after the depthwise separable convolution layers. In these layers, the input is changed using a succession of 1x1 and 3x3 convolutions before being recombined with the original inputs. This enables the network to effectively discover both basic and complicated data modifications.

5. **Auxiliary Branch**: The final network block has an auxiliary branch. The auxiliary branch comprises a sequence of layers that are trained concurrently with the primary network. The final forecast is formed by combining the output from the auxiliary branch with that of the main network. [15] [17]

Chapter 3

Introducing vidiQulus

vidiQulus is a state-of-the-art video surveillance app that can be used to search among CCTV images. It is a highly versatile visual search engine that is capable of tracking people by location and time.

It analyses CCTV video feed in real-time and displays the results in a user-friendly web application.

vidiQulus interface enables an user to search and narrow down their searches in any video by mainly three components: what, where, and when.

These components are described as below:

- **What** : Users may like to filter out results by just providing the object name and retrieving any images containing the relevant item.
- **Where** : Users may request all images in which the term "person" appears in a balcony, canteen, or other specific location, i.e., filtering results by location.
- **When** : Users may also wish to restrict search results to a specific time period. For example, someone may want to search for images containing the word "handbag" between 12 noon on July 27 and 8 a.m. on July 28 i.e., filtering results by time.

3.1 Motivation

In recent years, security has been a growing priority for both individuals and businesses. There is a need for efficient and effective surveillance systems that can monitor and protect public spaces in light of the growth in crime rates and security risks. Conventional surveillance systems have limited capabilities and rely significantly on

human monitoring, which can be costly, error-prone, and time-consuming.

To solve these issues, there is a rising interest in building AI-based surveillance systems that can automate the monitoring and analysis of live video feeds. With sophisticated algorithms, such systems can detect, track, and identify humans and objects, even misplaced items. Particularly, facial recognition technology has evolved as a potent tool for identifying and tracking persons, making it an integral feature of contemporary surveillance programs.

By incorporating face recognition technology based on artificial intelligence into surveillance systems, it is not only possible to identify individuals, but also to recover lost objects and monitor suspicious behavior.

Developing an AI-based surveillance system using facial recognition technology can be a cost-effective and efficient means of increasing public space security and safety. By automating the monitoring and analysis of video feeds, this technology can free up important human resources and enable security staff to respond to possible threats with greater speed and efficiency. In addition, the system can provide useful insights into behavioral patterns, enabling enterprises to improve their security procedures and enhance public safety.

To solve these challenges, an idea sprung up in Coriolis Technologies to create a surveillance system that would use state-of-the-art facial recognition algorithms combined with high-end infrastructure that is capable of recognising and tracking people.

3.2 Uses

vidiQulus can be used for various purposes including:

3.2.1 Lost Item Retrieval

Losing belongings such as phones, wallets, keys, or equipment may be a stressful experience, resulting in time-consuming searches and, on occasion, expensive replacements.

vidiQulus aims to provide a solution to this issue and has the potential to save valuable employers time and effort while searching for misplaced items.

vidiQulus combines object detection and face recognition to rapidly locate your missing item.

For instance-

If Ishita loses her laptop in the office around 2:00 PM, we may utilise the vidiQulus interface to search for both "Ishita" and "Laptop" between 1:30 PM and 3:30 PM. This will help us determine where Ishita was last seen with her laptop, thereby expediting our search for Ishita's laptop.

Individuals and businesses alike can use vidiQulus's simple, customisable user interface for all of these.

.

3.2.2 Enhanced emergency response

To improve workplace safety, vidiQulus can also be integrated with emergency response systems. Every second matters in an emergency, such as a fire or a stroke victim collapsing.

By integrating real-time monitoring and automatic alerting, vidiQulus enables security professionals and emergency services to respond to emergency situations more quickly and efficiently.

vidiQulus is capable of monitoring live video feeds from CCTV cameras and sensors in order to detect possible emergencies in real time. Using machine learning techniques, vidiQulus can learn to spot trends in surveillance data that may signify an emergency. When it detects an emergency, it can automatically notify the emergency services.

By implementing vidiQulus, offices can drastically shorten response times and boost the likelihood of a successful conclusion.

3.2.3 Enhanced Biometric systems

As office security becomes a primary priority for businesses, many are resorting to cutting-edge technologies such as surveillance AI systems with facial biometric capabilities to safeguard their assets and employees. vidiQulus can also be combined with a facial biometric system to enhance office security.

vidiQulus can monitor live video feeds from the facial biometric scanning camera, and when it recognises an office employee, it can immediately unlock the door and let the employee in.

vidiQulus can save organisations the cost of employing security guards or providing access cards to every employee by using an automated system to perform these tasks.

Chapter 4

Facial recognition Algorithm

Person recognition is one of the most significant features of our surveillance software. We can accomplish this in two ways:

- Recognising by Face(Facial recognition)
- Recognition by whole body(Person Re-identification)

We have tried to implement both features in our software.

The results of our facial recognition algorithm are discussed in this section.

We also tried recognition by entire body but weren't able to achieve significant results with the Person Re-identification algorithms. The results and algorithms are discussed in Appendix A.

4.1 Challenges faced with small faces

We noticed that the faces that are present in our CCTV images are very small, which creates problems for many types of facial recognition due to the following reasons:

- **Poor Image Quality** : One of the most difficult aspects of recognizing small faces is that the photos captured by cameras are frequently of poor quality, with low resolution and poor illumination. This can make it challenging for facial recognition algorithms to detect and identify faces with precision.
- **Limited Facial Feature** : Small faces contain fewer facial characteristics, making it difficult for face recognition systems to differentiate between persons. With fewer features to work with, the algorithm may have difficulty distinguishing between individuals, resulting in a decrease in accuracy.

- **Limited Training Data** : Most of the face recognition algorithms are trained on large datasets of facial images; however, these datasets may not contain sufficient examples of small faces. This can result in decreased accuracy while attempting to identify little faces.
- **Occlusion** : Accessories such as hats, spectacles, and masks are more likely to partially conceal small faces. This can hinder the algorithm's ability to detect and recognize the face effectively.
- **Pose Variations** : Small faces are more likely to display fluctuations in position and orientation, making it challenging for the algorithm to precisely match the face with the reference image. [18]

4.2 How Insightface solves these problems

We tried various algorithms before settling on Insightface as our preferred choice of facial recognition algorithm. Insightface solves the problem of small faces recognition due to the following reasons:

- **Advanced Facial Feature Extraction** : Insightface employs various subnetworks for extraction of facial traits, including both global and local characteristics. This improves the accuracy by capturing a greater variety of facial features, even in small faces.
- **Robust Image Enhancement and Normalization** : Insightface uses advanced image enhancement and normalisation techniques, such as histogram equalization and adaptive contrast enhancement, to improve the quality of photos, especially those with poor lighting circumstances.
- **Large-Scale Datasets** : Insightface is trained using massive datasets comprising a wide variety of faces, including several small faces. This ensures that the system can effectively recognize faces in a variety of sizes and variants.
- **Occlusion handling** : Insightface integrates strategies for addressing occlusion, including the use of attention mechanisms to focus on the viewable regions of the face and the use of robust feature representations that are less influenced by occlusion.
- **Pose-Invariant Features** : Insightface employs posture-invariant characteristics, which are less impacted by fluctuations in position and orientation, to increase accuracy in the recognition of small faces that may display pose variations.

Insightface has been optimized for high-performance computing, making it possible to process large volumes of images quickly and efficiently, even on resource-constrained

systems. This enabled us to create a highly scalable facial recognition algorithm that also achieves high performance, capable of processing 15 frames per second that is capable of detecting faces that are as small as in the range of 30 x 30 cm to 18 x 18 cm. [17]

4.3 Pipeline

This section describes the pipeline that we used for our facial recognition algorithm.

Figure 4.1: Face recognition pipeline

1. **Input image**: First, we take the input image for which we want to identify the person in that image.

2. **Embeddings generation**: We take the input image and pass it to the Buffalo-l model of Insightface. If it detects a person, we take embeddings of the face of the person recognised in that image. Let's call it $\vec{I}$

3. **Similarity search in Opensearch**: We have a database of images stored inside Opensearch. Each of these images has the embeddings of different people tagged to them with the name of that person. Let's call one of the mebddings as $\vec{O}$.

4. **Getting the image with the best score**: We calculate the similarity score between $\vec{I}$ and $\vec{O}$ for each of the $\vec{O}$ vectors present in the database. We obtain the image in the database that has the highest similarity score.

5. **Recognising image**: We take the name tag of the image that has the highest similarity score, let's say X and say that the recognised face in the input image is X.

4.4 Preparing the Database of images

1. We scanned through multiple images from the CCTV cameras and identified the "best" images from those images for each person. By "best", we mean images in which a person is clearly visible and is identifiable by eyes.

2. We collected various such images for many of the Coriolis employees and prepared a database of images that can be used for face recognition.

3. We stored the embeddings of all those images along with the name of the person in Opensearch.

4.5 Using the database to recognise faces

Suppose we have an image that has a face in it. We use Insightface to generate embeddings for that face, let's call it $\vec{I}$.

We also have the embeddings of each image in the database, let's call them $\vec{D_i}$, where i represents each image in the database.

We then calculate the cosine similarity between those 2 vectors using the formula:

$$\cos(\theta) = \frac{\mathbf{I} \cdot \mathbf{D}_i}{\|\mathbf{I}\|\|\mathbf{D}_i\|} \tag{4.1}$$

Then we transform it using the following equation to get the distance function value, let's call X:

$$X = \frac{1}{1 + \cos(\theta)} \tag{4.2}$$

We calculate X_i for each of images in the database and display the results in descending order of values of X.

We take the highest value of X_i, let's say it is X_m.

We then look for the name that was associated with D_m, suppose that the name associated with D_m is Ishita.

So, we would conclude that the face in the input image is that of Ishita. We also call X_m as the "confidence" with which the person is recognised in the image.

The algorithm described earlier is dependent on the number of people in the database of images. If it detects a new face that is not the database, it should display "Unknown".

To achieve that, we have to put a threshold, let's say K. If X_m ¡ K, we would say that the person recognised in the image is "Unknown".

4.6 Testing the model

To test the accuracy of our model, we generated a test data. We also used the test data to determine the threshold score below which we will say that the recognised person is "Unknown".

We curated a list of 181 images to test our algorithm. In all those images, we manually identified each person in all those photos. We will call all these names "correct names".

Then we used Insightface and the algorithm that was described earlier to identify the person in that image. We call these "recognised name". We also calculate the "confidence" score of detection.

Let's define 2 terms that will be useful for our analysis further **percent of detection** and **percent of true positives**.

Number of recognised images(D)

All the images that had a confidence score greater than the threshold are labeled as recognised image.

Number of true positives(T)

Among all the recognised images, the images that satisfied the condition "correct name"="recognised name", are labeled as True positive image.

We define **percent of detection(P_D)** and **percent of true positives(P_T)** as:

$$P_D = \frac{D}{X} * 100 \tag{4.3}$$

$$P_T = \frac{T}{D} * 100 \tag{4.4}$$

where X stands for the total number of images.

We calculated **percent of detection** and **percent of true positives** for different values of threshold and obtained the results.

4.6.1 Experiments with different distance functions and KNN algorithms

Our first experiment involved trying out different distance functions and different KNN algorithms to find the best combination of distance function and KNN algorithm, and also the best threshold value.

We tried with different distance functions, namely:

1. **cosine similarity**.
2. **L2 distance function**
3. **Minkowski distance function with p=3**

The formula for each of the loss function are given below:
Cosine similarity distance function

$$\text{cosine similarity}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{|\mathbf{u}||\mathbf{v}|}$$

where $\mathbf{u}$ and $\mathbf{v}$ are two vectors, $\cdot$ denotes the dot product between them, and $|\cdot|$ denotes the Euclidean norm of a vector.

L_2 distance function

$$\text{L2 distance} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

where x_i and y_i are the i-th elements of vectors $\mathbf{x}$ and $\mathbf{y}$, respectively, and n is the dimensionality of the vectors.

Minkowski distance function

$$D_{\mathrm{M}}(\mathbf{p}, \mathbf{q}) = \left(\sum_{i=1}^{n} |p_i - q_i|^r \right)^{1/r}.$$

where $\mathbf{u}$ and $\mathbf{v}$ are two vectors and r is a parameter. We have used r=3 for our analysis.

We also tried to compare if the approximate KNN search that we are using in Opensearch (HNSW) any different from the exact KNN search, so we also performed exact KNN search using sklearn python library. Both of the KNN methods are described in Appendix B.

All the cases that we have tried is mentioned in the following table:

Case number	Loss function	KNN algorithm used
Case 1	cosine similarity	Approximate(Opensearch)
Case 2	cosine similarity	Exact(SKlearn)
Case 3	L2 similarity	Approximate(Opensearch)
Case 4	L2 similarity	Exact(SKlearn)
Case 5	minkowski p=3	Approximate(Opensearch)

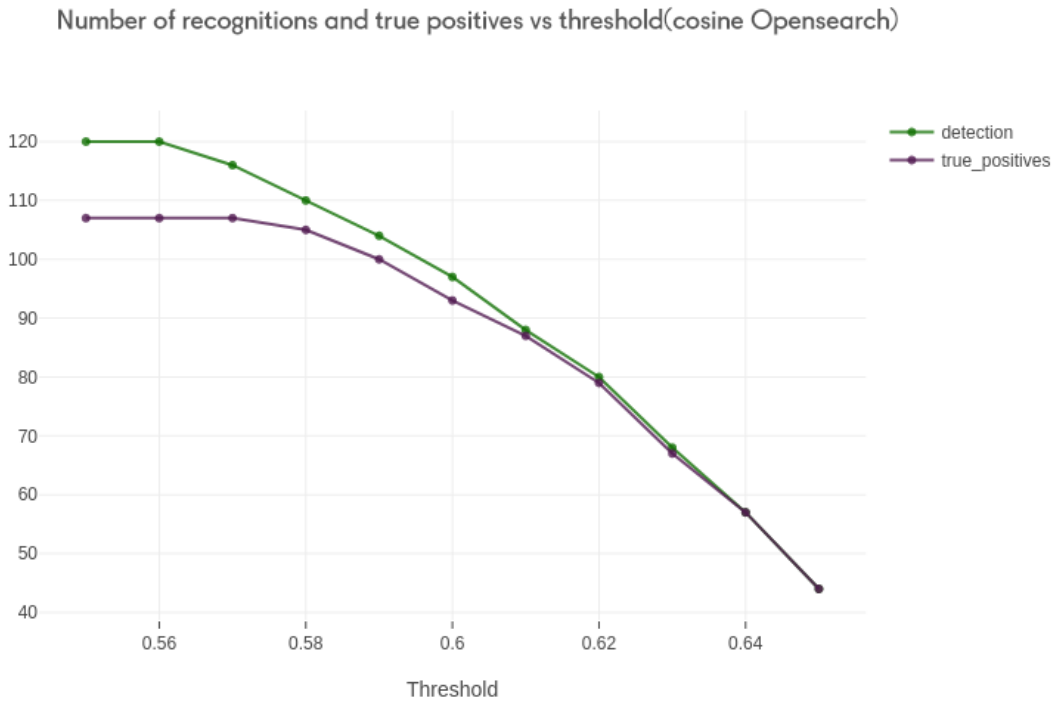
Table 4.1: All possible configurations tried with different loss functions and KNN algorithms

Below I present the results of all the cases.

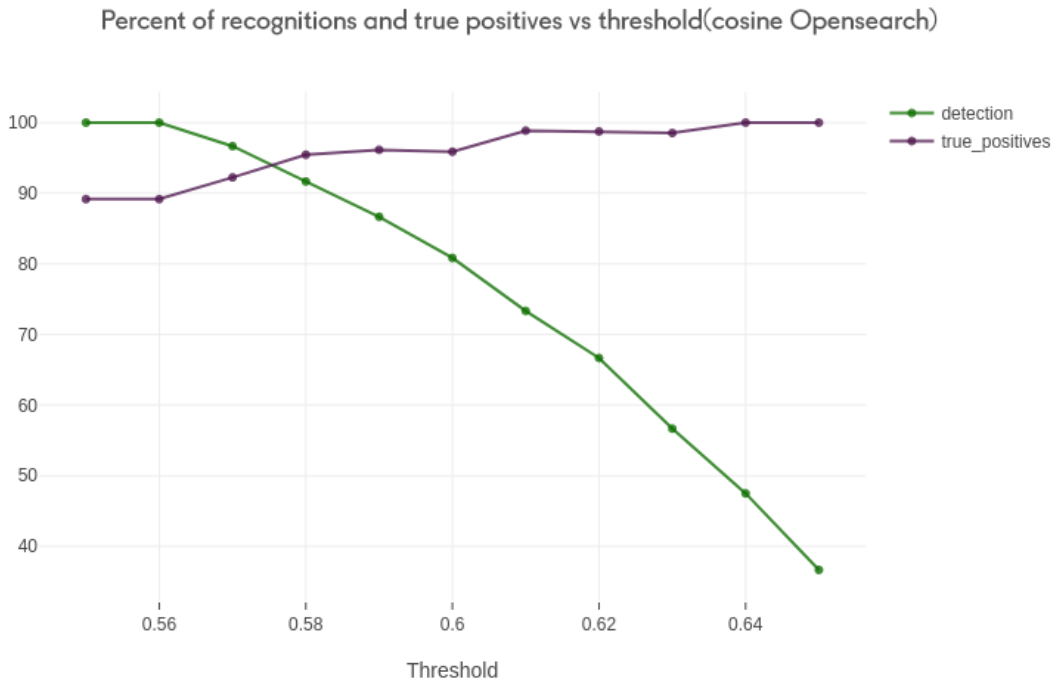
4.6.1.1 Case 1

Threshold	Number of Recognitions	Number of True positives
0.55	118	95
0.56	118	95
0.57	113	93
0.58	106	88
0.59	94	83
0.6	85	78
0.61	76	71
0.62	66	62
0.63	60	58
0.64	47	45
0.65	35	34

Table 4.2: Number of images recognised and number of true positives recognised for various thresholds.



(a) Number of images recognised and number of true positives recognised for various thresholds.



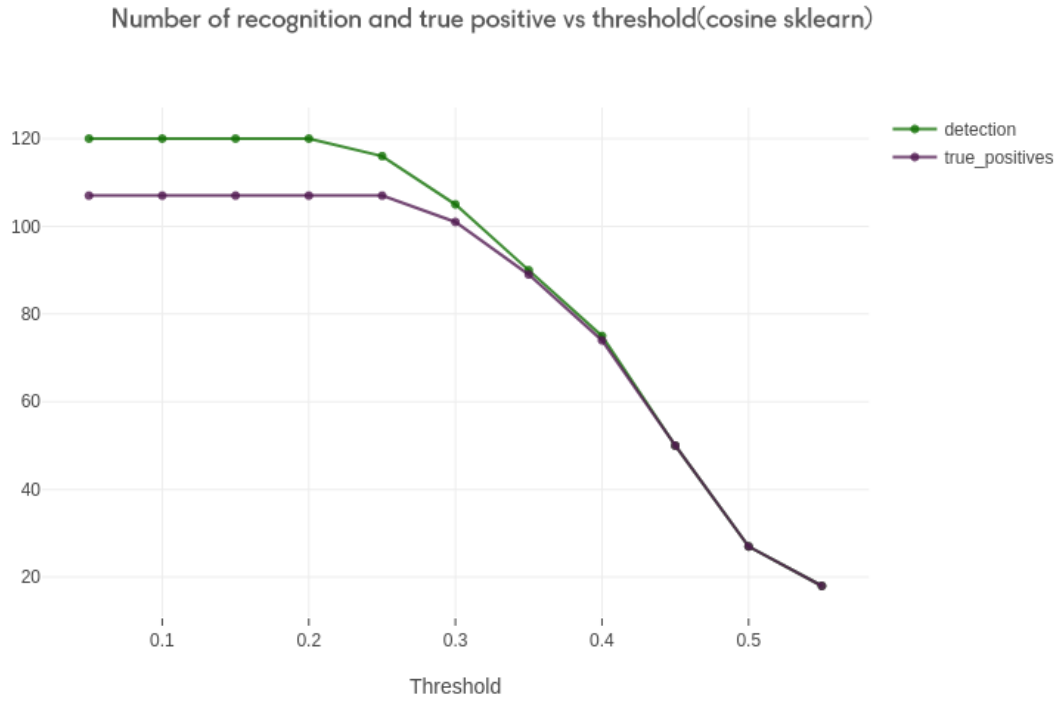
(b) Percent of images recognised and percent of true positives recognised for various thresholds.

Figure 4.2: Results for Case 1

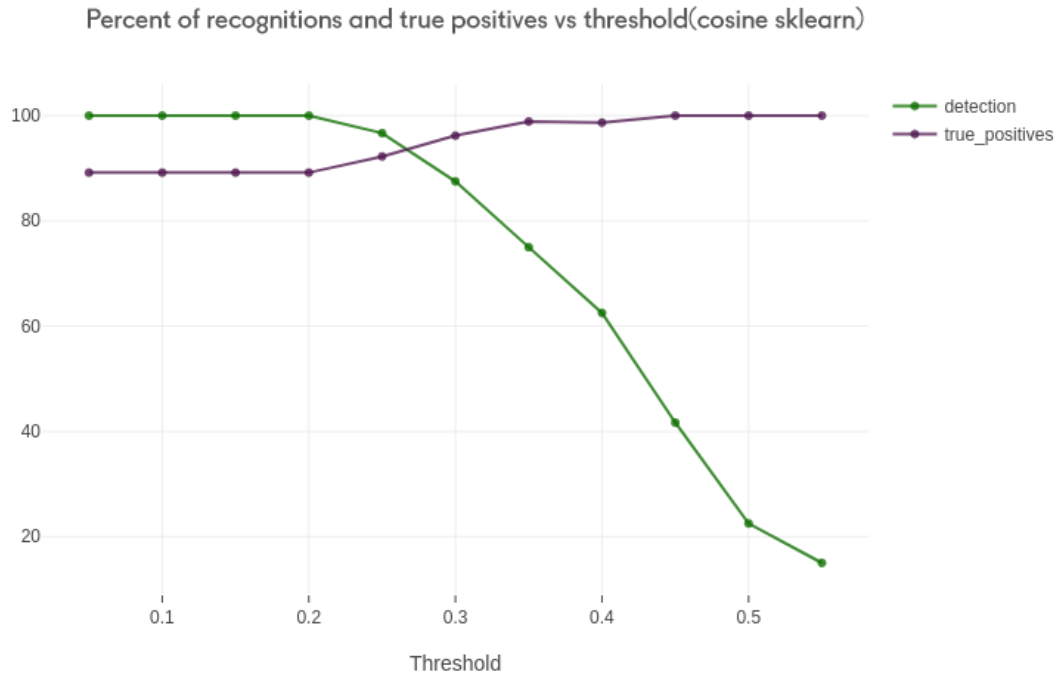
4.6.1.2 Case 2

Threshold	Number of Recognitions	Number of True positives
0.05	120	107
0.1	120	107
0.15	120	107
0.2	120	107
0.25	116	107
0.3	105	101
0.35	90	89
0.4	75	74
0.45	50	50
0.5	27	27
0.55	18	18

Table 4.3: Number of images recognised and number of true positives recognised for various thresholds.



(a) Number of images recognised and number of true positives recognised for various thresholds.



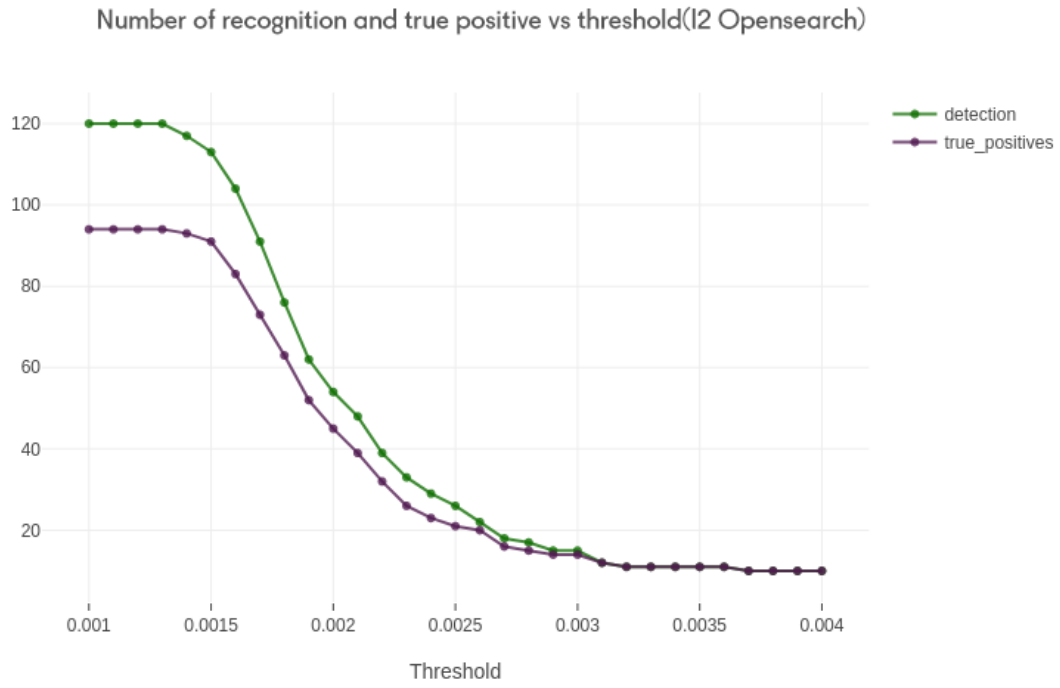
(b) Percent of images recognised and percent of true positives recognised for various thresholds.

Figure 4.3: Results for Case 2

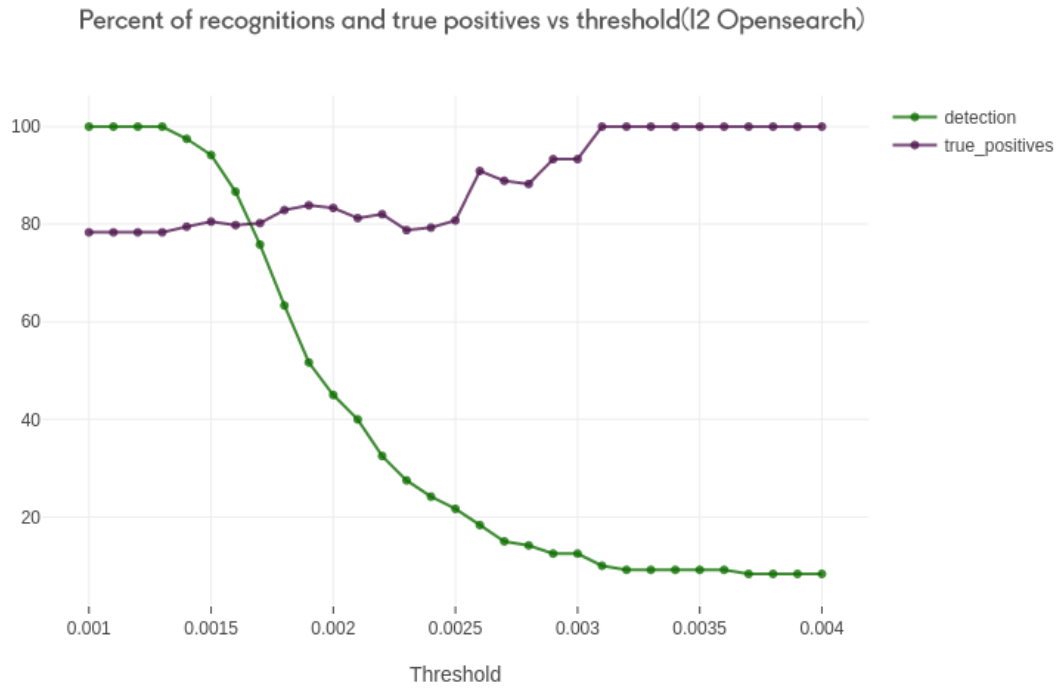
4.6.1.3 Case 3

Threshold	Number of Recognitions	Number of True positives
0.001	120	94
0.0011	120	94
0.0012	120	94
0.0013	120	94
0.0014	117	93
0.0015	113	91
0.0016	104	83
0.0017	91	73
0.0018	76	63
0.0019	62	52
0.002	54	45
0.0021	48	39
0.0022	39	32
0.0023	33	26
0.0024	29	23
0.0025	26	21
0.0026	22	20
0.0027	18	16
0.0028	17	15
0.0029	15	14
0.003	15	14
0.0031	12	12
0.0032	11	11
0.0033	11	11
0.0034	11	11
0.0035	11	11
0.0036	11	11
0.0037	10	10
0.0038	10	10
0.0039	10	10
0.004	10	10

Table 4.4: Number of images recognised and number of true positives recognised for various thresholds.



(a) Number of images recognised and number of true positives recognised for various thresholds.



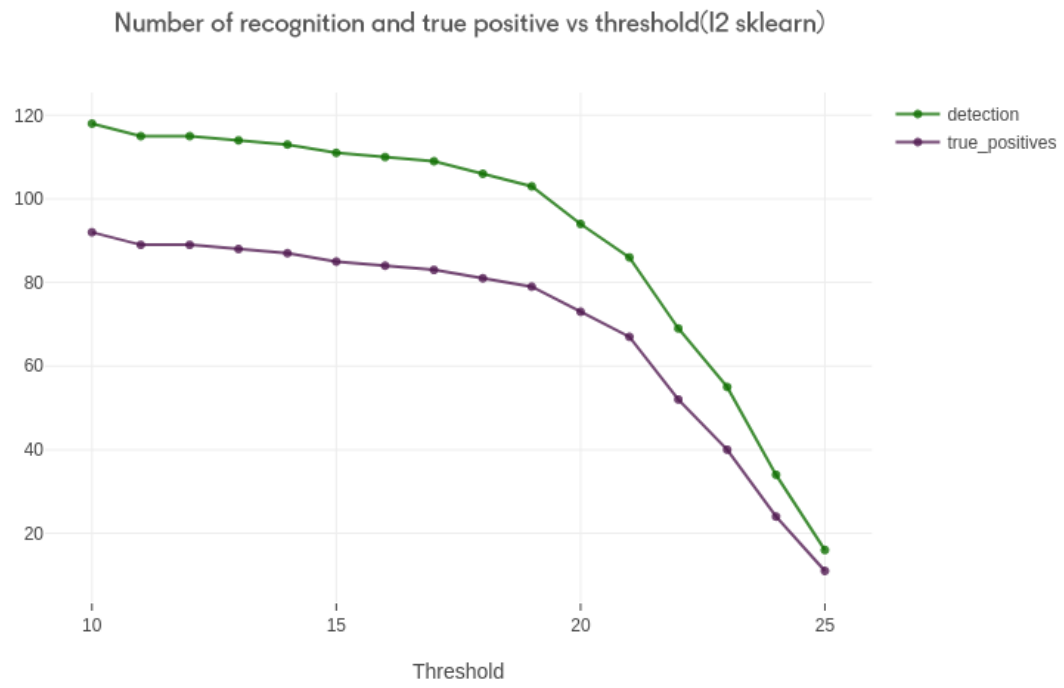
(b) Percent of images recognised and percent of true positives recognised for various thresholds.

Figure 4.4: Results for Case 3

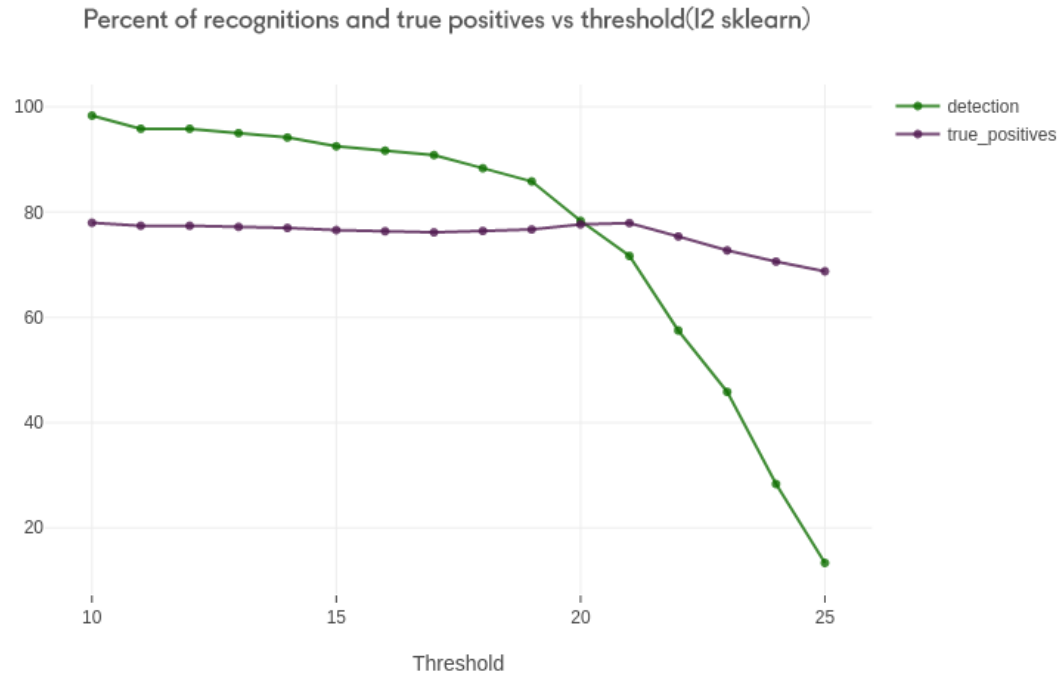
4.6.1.4 Case 4

Threshold	Number of Recognitions	Number of True positives
10	118	92
11	115	89
12	115	89
13	114	88
14	113	87
15	111	85
16	110	84
17	109	83
18	106	81
19	103	79
20	94	73
21	86	67
22	69	52
23	55	40
24	34	24
25	16	11

Table 4.5: Number of images recognised and number of true positives recognised for various thresholds.



(a) Number of images recognised and number of true positives recognised for various thresholds.



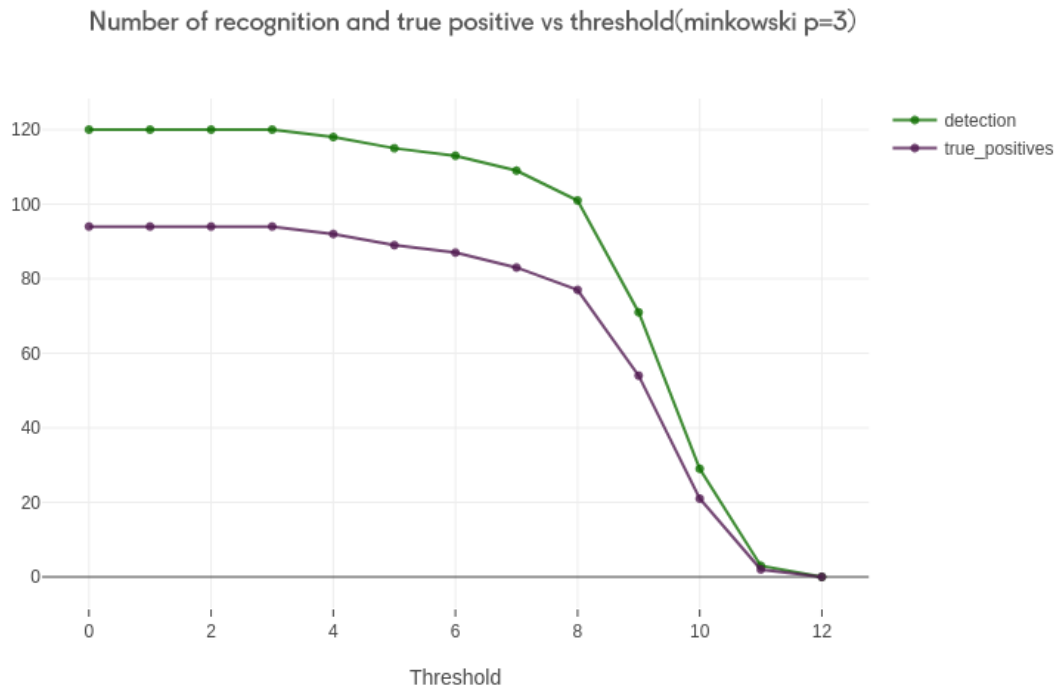
(b) Percent of images recognised and percent of true positives recognised for various thresholds.

Figure 4.5: Results for Case 4

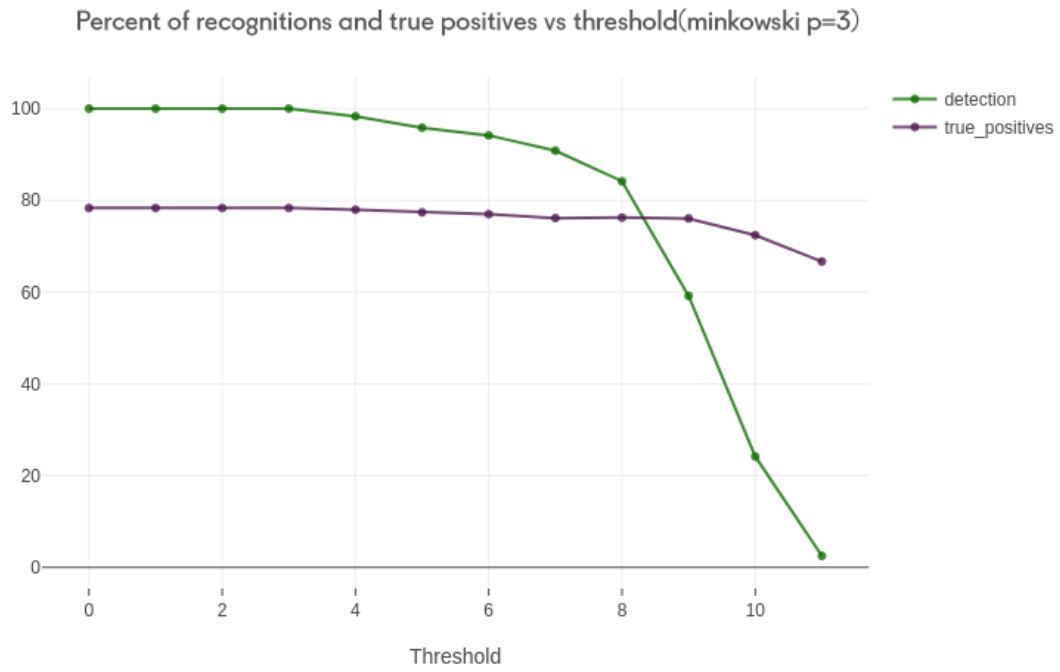
4.6.1.5 Case 5

Threshold	Number of Recognitions	Number of True positives
0	120	94
1	120	94
2	120	94
3	120	94
4	118	92
5	115	89
6	113	87
7	109	83
8	101	77
9	71	54
10	29	21
11	3	2

Table 4.6: Number of images recognised and number of true positives recognised for various thresholds.



(a) Number of images recognised and number of true positives recognised for various thresholds.



(b) Percent of images recognised and percent of true positives recognised for various thresholds.

Figure 4.6: Results for Case 5

4.6.2 Discussions

In all these cases, we can see that as the threshold is increased, the number and percent of detection keep decreasing as with a higher threshold, the number of images that will be recognised with a confidence greater than the threshold will be decreased.

We also noticed that the number of true positives decreased with increasing threshold, in line with its definition. However, the percent of true positives kept on increasing with an increasing threshold as the number of low confidence recognitions kept on decreasing and the number of true positives kept on decreasing.

From these experiments, we concluded that the best performing combination was when the distance function was cosine similarity paired with the approx KNN method by Opensearch.(Case 1)

We rejected cases 2 and 4 because they gave almost the same results but took much longer to process (approx 10 times more). We rejected case 3 and case 5 as the percent of true positives was less than in case 1.

We also decided on a threshold of 0.57 for our algorithm, as it wasn't leaving a lot of images unrecognised but was also making a reasonable amount of correct predictions.

4.6.3 Varation of K

In a K-nearest neighbors (KNN) search, K refers to the number of nearest neighbors that will be used to make a prediction for a new data point. For example, if K=3, then the algorithm will consider the 3 closest points to the new data point and make a prediction based on the labels of those 3 points.

We wanted to see if the variation of K had any effect on our result or not. To check that, we varied the values of K keeping all other variables the same with threshold=0.57.

Here are the results:

Threshold	Number of Recognitions	Number of True positives
1	116	107
5	116	107
10	116	107
15	116	107
20	116	107
25	116	107
30	116	107
1000	116	107

Table 4.7: Number of images recognised and number of true positives recognised for various values of k.

From the above table, we can clearly see that the number of images recognised and the number of true positives doesn't change with changing values of k. So, from our experiments, we can conclude that the hyperparameter k in our model doesn't affect the results.

4.6.4 Variation of size

Size in our model refers to the number of images that OPensearch pulls from its KNN search. More is the size, more is the number of images that are pulled. Out of all those images, we take the one with the highest score and call that the recognised image.

We wanted to see if the variation of size had any effect on our result or not. To check that, we varied the values of size while keeping all other variables the same with threshold=0.57.

Here are the results:

Threshold	Number of Recognitions	Number of True positives
1	116	107
5	116	107
10	116	107
15	116	107
20	116	107
25	116	107
30	116	107
1000	116	107

Table 4.8: Number of images recognised and number of true positives recognised for various values of size.

From the above table, we can clearly see that the number of images recognised and the number of true positives don't change with changing values of size. So, from our experiments, we can conclude that the hyperparameter size in our model doesn't affect the results.

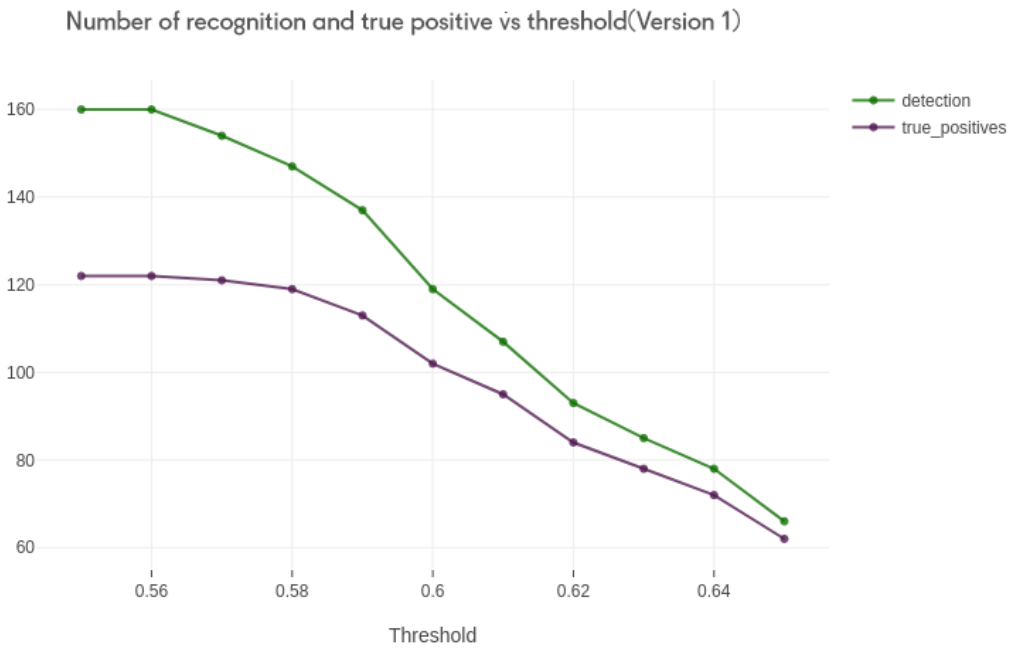
4.6.5 Increasing the number of images in the database

One of the experiments also involved increasing the number of images in the database from 168 to 417.

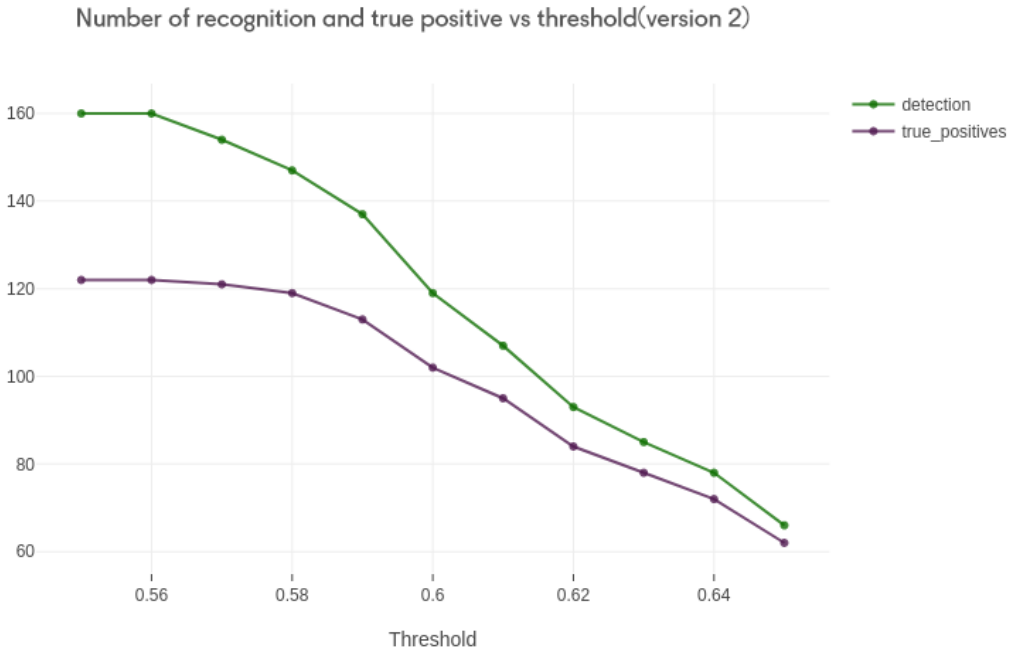
The hypothesis that we wanted to check was whether increasing the number of images improves the accuracy of the face recognition model.

We tested both databases on the same test dataset, which consisted of 161 images that were curated by me and tagged with names. Here are the results:

4.6.5.1 Comparison of both versions of dataset

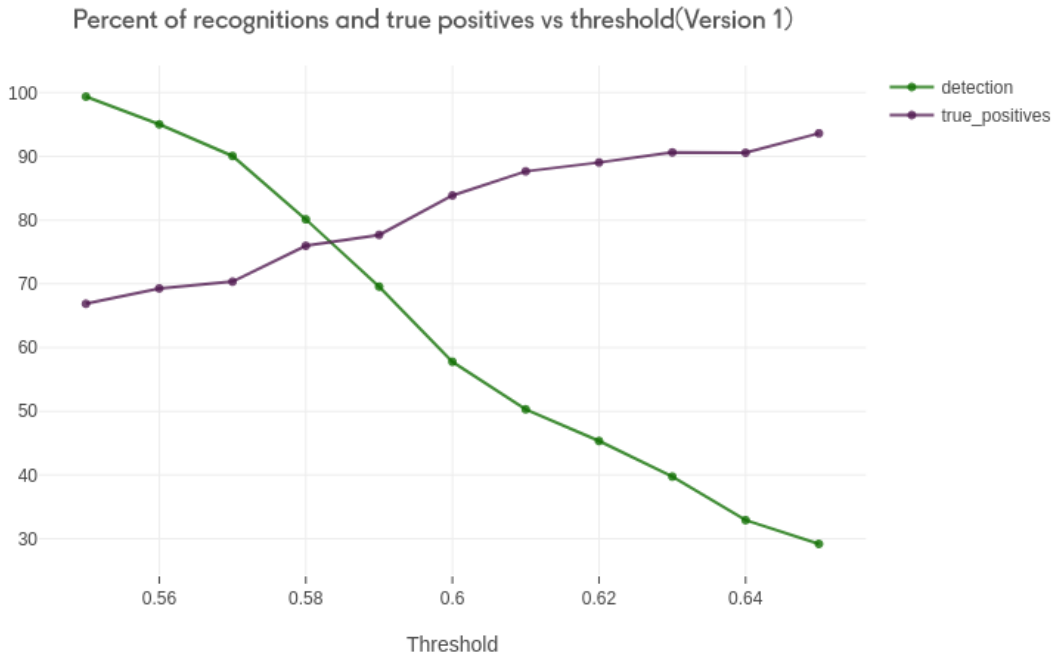


(a) Version 1 of database

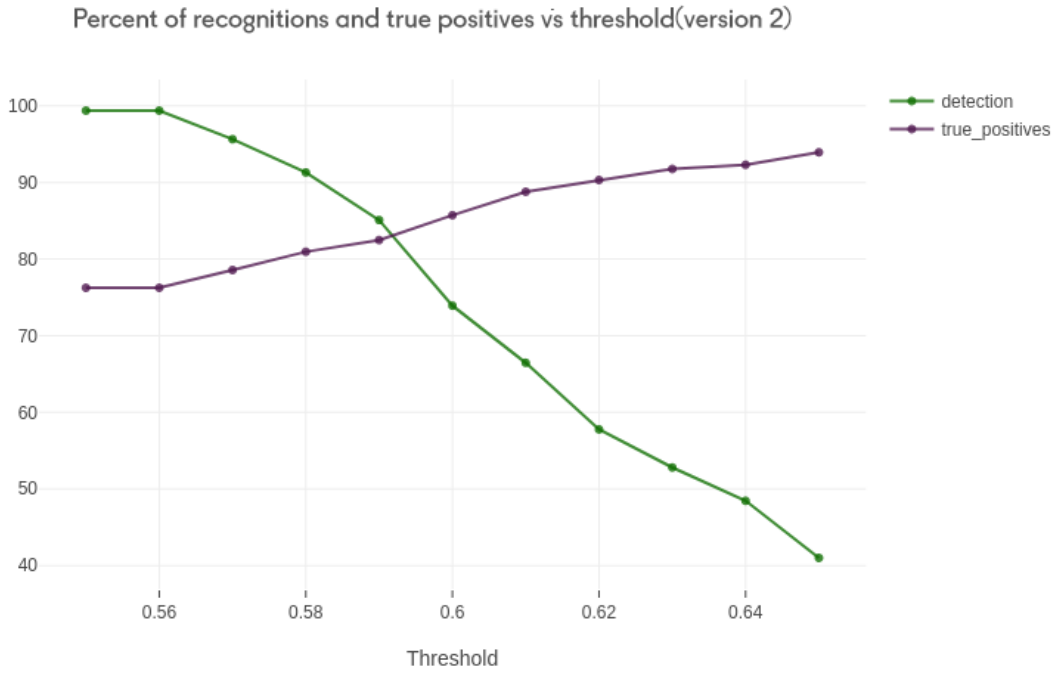


(b) Version 2 of database.

Figure 4.7: Number of images recognised and number of true positives recognised for various thresholds for version 1 vs version 2 of database



(a) Version 1 of database



(b) Version 2 of database.

Figure 4.8: Percent of images recognised and percent of true positives recognised for various thresholds for version 1 vs version 2 of database

4.6.6 Discussions

From the above 2 experiments, it was clear that increasing the number of images in the database does indeed lead to a significant increase in the accuracy of recognitions. So, it supports our initial hypothesis that increasing the number of images in the database will lead to a better result in face recognition.

4.7 Deploying the model

After rigorously testing the model, we determined that it was ready to be deployed in our production cluster. So we had to translate our models to be capable of running on Spark. We made those changes and deployed them on the production cluster. The code was successfully deployed, and the UI was successfully able to display the images with a bounding box on the persons faces with a name tag attached to it.

Chapter 5

Infra changes

5.1 Automating the creation of the Kubernetes cluster

As discussed in the architecture section, our entire cluster is built on the backbone of Kubernetes. Setting up a Kubernetes cluster is a very time-consuming and tricky process. So, to avoid any future troubles with this and to ensure that the Kubernetes cluster can be created without any trouble with just a few simple commands.

5.1.1 What is Ansible?

Ansible is an open-source automation tool for configuration management, application deployment, and job automation. Michael DeHaan built it in 2012, and Red Hat acquired it in 2015. Ansible defines a set of actions that should be run on distant machines using a simple YAML syntax called "playbooks." [19]

5.1.2 Ansible architecture

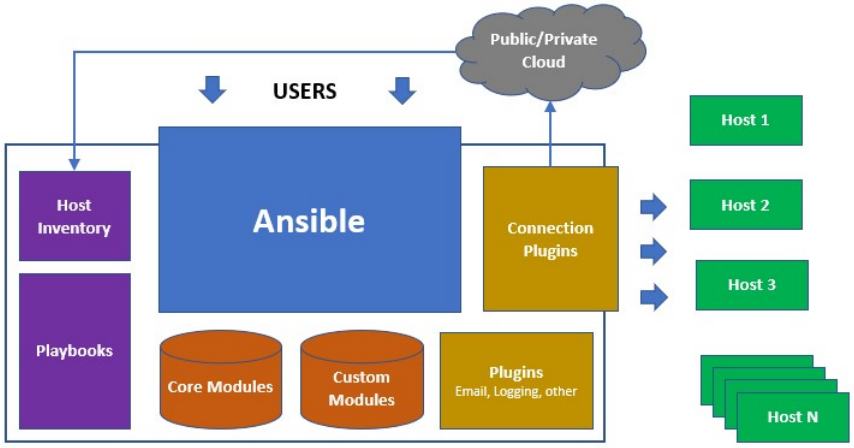


Figure 5.1: Ansible architecture taken from [20]

The Ansible orchestration engine interacts with a user who is writing the Ansible playbook to execute the Ansible orchestration and interact with the services of a private or public cloud and a configuration management database.

The Ansible architecture operates as follows:

- **Control Machine** : Ansible is installed on a control machine, which is used to manage the remote hosts. The control machine runs the Ansible playbooks and executes tasks on the remote hosts
- **Inventory**: The inventory is a configuration file containing a list of remote hosts that Ansible will manage. The control machine uses the inventory file to determine which hosts to connect to and which tasks to conduct.
- **Playbooks**: Playbooks are YAML files that define the tasks that Ansible will run on remote systems. Playbooks are comprised of one or more plays, each of which contains a list of actions to be carried out by a certain set of hosts.
- **SSH**: Ansible uses SSH to connect to and execute tasks on remote servers. Ansible may be configured to use a variety of SSH settings, including SSH keys, usernames, and passwords.
- **Execution**: Ansible conducts tasks in parallel on distant computers, making it quick and effective.

Ansible's design is based on a client-server model. Ansible does not require the installation of agents on client machines, which means it can communicate with other host machines, setup ansible engine in those host machines and execute the commands on those host machines. This makes Ansible portable and simple to use.

5.1.3 How Ansible saved time?

As said earlier, deploying a Kubernetes cluster is a repetitive and time-consuming process. So, to automate the process of this repetitive and time-consuming process, we have developed ansible playbooks, which will create a Kubernetes cluster by executing the ansible playbooks with a few simple commands.

Since Ansible is lightweight, it also does the job very fast and creates the kubernetes within 15 minutes, which earlier took around 3 hours to create.

5.2 Changing from Elasticsearch to opensearch

A large part of our infrastructure involves taking the embeddings of an input image and doing a similarity search. So, we needed a software that can perform the similarity search in a fast and accurate manner. This is where elasticsearch came into picture.

5.2.1 What is Elasticsearch?

Elasticsearch is a distributed search and analytics engine built to process massive volumes of data in near real-time. It is built on the Apache Lucene search library and offers a scalable, flexible, and reliable platform for searching, indexing, and analyzing data.

Elasticsearch employs a distributed design that enables the indexing and searching of data over numerous nodes in a cluster, enabling high availability, fault tolerance, and scalability. It can index and search data in a variety of formats, such as JSON, text, geographical data, and structured data.

Elasticsearch's robust search capabilities, which include full-text search, aggregations, filters, and sorting, are one of its defining characteristics. Furthermore, fuzzy matching, proximity search, and partial matching are supported.

Elasticsearch's RESTful API and support for many programming languages facilitate integration with a wide variety of applications and systems. In addition, it contains numerous built-in tools and plugins for data visualization, data ingestion, security, monitoring, and administration.

Overall, Elasticsearch is a highly powerful search and analytics engine that can assist enterprises in searching, indexing, and analyzing enormous volumes of data rapidly and efficiently. It is a popular option for a variety of applications and use cases, including e-commerce, log analysis, security analytics, and more, thanks to its distributed design, sophisticated search capabilities, and extensive feature set.

5.2.2 Problems with using Elasticsearch

Before, we used Elasticsearch as our software for choice for querying data for UI.

But we noticed 3 distinct problems during our course of our internship:

- **Persistent storage** : A major problem that we faced was that the data that elasticsearch stored, i.e., embeddings and other metadata of all the images, were deleted everytime elasticsearch stopped working.
- **Slow loading time of UI**: During the course of our project, we noticed that elasticsearch was loading data in a slow way taking around 2-3 minutes to load a full day's data.

So, to solve these 2 crucial issues we moved to Opensearch. Also, combined with the fact that the similarity search feature in Elasticsearch was paid and in Opensearch was free prompted us to move to Opensearch.

5.2.3 What is opensearch?

OpenSearch is an open-source, distributed search and analytics engine meant to provide a platform for indexing, searching, and analyzing data that is quick, scalable, and customizable. AWS just forked Elasticsearch, and a group of developers and contributors are currently developing and maintaining it.

OpenSearch, like Elasticsearch, is based on the Apache Lucene search framework and supports a variety of data kinds and formats, such as JSON, text, geographical data, and structured data. It employs a distributed design that enables data to be indexed and searched across several nodes in a cluster, hence delivering high availability, fault tolerance, and scalability.

OpenSearch provides many of the same fundamental features and capabilities as Elasticsearch, including full-text search, aggregations, filters, and sorting. In addition, it contains extra functionality not found in Elasticsearch, including anomaly detection and index state management.

It also provides similarity search free of cost.

5.2.4 Changes observed

- **Persistent storage** : Introducing opensearch solved our problem of persistent storage. Opensearch has been stopped multiple times but it has always been reinstated.

- **Slow loading time of UI**: The loading time of all day's data was reduced from 2-3 mins to 1 min.

Chapter 6

Spark changes: Making processing images faster

6.1 What is Apache spark?

Apache Spark is an open-source, distributed computing system built for analysing massive amounts of data. It can quickly handle very large data sets and distribute data processing tasks across several computers, either alone or in conjunction with other distributed computing technologies.

Spark was developed by AMPLab at U.C. Berkeley in 2009. From its humble origins, it has become one of the key big data distributed processing frameworks in the world. [5]

It achieved this phenomenal growth on the backbone of its 4 key design principles:

- **Speed** : Spark's internal implementation is optimized to take advantage of the hardware industry's improvements in CPU and memory performance, and its physical execution engine, Tungsten, uses whole-stage code generation to generate compact code for execution. This makes Spark hugely fast with a huge performance boost.
- **Ease of Use** : Spark provides a simple programming model for building big data applications in familiar languages.
- **Modularity** : Spark offers unified libraries with well-documented APIs to create a unified processing engine for workloads.
- **Extensibility** : Spark is a fast, parallel computation engine that decouples storage and compute, allowing it to read data from multiple sources and process it in memory.

6.2 Spark architecture

A Spark system is comprised of a number of independent machines working cooperatively towards a shared objective. Spark's architecture is based on the master-slave model.

To make the entire cluster working, we need to have a machine which manages the cluster as a whole. That machine is called the **Driver Node**.

6.2.1 Driver Node

The Spark driver is used to orchestrate the whole Spark cluster. It manages how work is spread across the cluster and which computers are available for the duration of the cluster's existence and ensures the distribution and smooth running of all spark jobs.

The driver node is similar to any other computer in the cluster. It contains hardware like as a CPU, memory, DISKs, and a cache; however, these components are employed to host the Spark Program and control the cluster as a whole. The driver is the user's connection to the physical computing necessary to finish any job submitted to the cluster.

All the machines inside the spark cluster requires **JVM(Java Virtual Machine)** to be running inside the machines. JVM is responsible for managing memory, garbage collection, and other low-level system resources necessary for executing the Spark application.

This JVM is utilised by the Spark Program to build the SparkContext, serves as the entry point for all Spark applications and coordinates the execution of tasks throughout the cluster.

The driver contains various components which are responsible for translating user code into Spark jobs which are to be executed on the cluster.

6.2.2 Cluster Manager

The driver node conceals the cluster manager, which is responsible for procuring Spark cluster resourcess, such as CPU and memory and allocating them to Spark jobs. Additionally, the cluster manager is responsible for initiating and stopping executors as required and controlling the cluster's overall health.

6.2.3 Worker Node

A worker node in Apache Spark is a machine in a cluster that executes one or more Spark executors. The responsibility of worker nodes is to execute the tasks given to them by the driver and return the results to the driver. [21]

6.3 UDFs

Spark's UDFs allow developers to design their own functions that may be used in Spark SQL queries and DataFrame manipulations. A UDF is a function that accepts several inputs and returns one output.

When a Spark UDF is registered, it becomes accessible to all Spark sessions running on the same cluster. This means that UDFs can be shared across many apps and reused in various codebase components.

Spark implements UDFs internally as distributed functions that are executed throughout the cluster. Spark serialises the called UDF and transmits it to the worker nodes responsible for data processing when a UDF is invoked. Thereafter, the UDF is executed on the worker nodes and the results are delivered to the driver node.

In short, Spark's UDFs are a powerful tool for extending Spark's functionality and constructing unique data processing pipelines that can handle a broad variety of use cases.

6.3.1 Pandas UDFs

The class of UDFs that were used before I joined in Coriolis was Pandas UDFs.

Pandas UDFs, also known as Vectorized UDFs, are a method for applying Python code that use Pandas functions to Spark DataFrames. Pandas UDFs allow users to conduct complex data operations on huge datasets in a distributed manner using Spark.

Fundamentally, Pandas UDFs in Spark operate by partitioning input data and applying the UDF to each partition. This allows the UDF to be executed in parallel across the cluster, making it far faster than standard row-based UDFs.

Pandas UDFs work in the same manner as regular UDFs but because of using pandas under the hood technology of handling large datasets, it can perform computations in a much faster way.

My job was to move away from pandas UDF based approach to a RDD based approach.

6.4 RDD

In Apache Spark, RDD stands for Resilient Distributed Dataset. RDD is Spark's core data structure that is used to represent a cluster-wide distributed collection of data.

RDDs are durable, so if a node in the cluster fails, Spark can retrieve the missing data by recomputation from the original source. This makes RDDs perfect for fault-tolerant processing of large-scale data.

Additionally, RDDs are meant to be cacheable, meaning they may be cached in memory for quick access. This allows RDDs to be reused throughout several stages of a Spark application, enhancing speed by minimising the number of times identical data must be recomputed.

In Spark all transformations are lazy, meaning that they do not immediately calculate their outcomes. Spark instead retains the alterations made to a base dataset (e.g. a file). Only when an action requires a result to be returned to the driver program are the transformations computed.

By default, each modified RDD is recomputed whenever an operation is performed on it. As can be seen from the above figure. RDDs can be retained in memory using the persist (or cache) technique and Spark will keep the elements on the cluster for considerably faster access on the next query. This creates extremely fast computations. [22]

6.5 Algorithm used

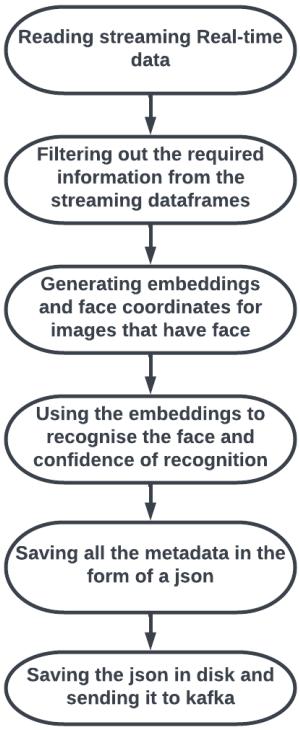


Figure 6.1: Algorithm used for implementing RDD method

The first step of the algorithms involve generating a sparkContext in the driver node.

Once the sparkContext is generated, we read the data from Kafka topic which stores all the data from the cameras in the form a streaming dataset.

We filter out the necessary information from the streaming dataframe.

We then pass this Dataframe into a transform function. The transform function takes each row of the dataframe. It extracts all the key columns of each of these rows.

One of those columns include the path of the image. We use the image to apply the insightface model on that image. If it detects any face we generate the embeddings and the coordinates of the faces in that image.

We use the embeddinsg and the face recognition algorithm that we develped earlier to recognise the persons in that image and the confidence with which the persons are detected.

We then put all the data into a json format and save that json in a disk and push the data to a Kafka topic to be used for further processing.

6.6 Benefits of using the RDD based approach

The pandas UDF based approach was using a for loop to loop over all the columns in the row. The for loop is a very time consuming algorithm and the introduction of the RDD based approach completely eliminated the need of the for loop making it a much faster process.

The RDD is also a much faster was to apply transformations as described in the earlier section.

All these lead to a tremendous boost in the performance of the algorithm. The previous UDF based approach was processing at an speed of 15 frames per second. Our RDD based approach increased the speed of the computation to 24 frames per second.

Chapter 7

Conclusions

In conclusion, we have made a prototype of an AI surveillance system that can serve the following purposes:

- Detect people by their whole body and face and display them in a comprehensive UI.
- Recognise people's faces and display the name above the bounding box.
- Search by people's face embeddings.
- Search by people's names and show them in descending order of time.

We have used Insightface models to generate the embeddings of the detected faces.

We also developed a face recognition algorithm using the embeddings given by the insightface model. The algorithm detects faces with an accuracy of 75% accuracy on a carefully curated test dataset.

We have also transformed all the codes for use in the Spark environment and deployed the codes in the production environment.

The UI has been made more fast and responsive thanks to transferring from Elasticsearch to Opensearch.

We have also made the entire system more scalable, which is capable of handling $\tilde{6}0$ million images per day.

By installing Opensearch, we have ensured that the entire system is fault tolerant and in the event of failures, there is no loss of data.

We have also increased the processing speed of the spark codes by changing from

a UDF based approach to a RDD based approach, and it increased the processing speed from 15 frames per second to 24 frames per second.

Chapter 8

Further work

- **Less accuracy of face recognition**: The accuracy of the face recognition algorithm that have been developed, and currently has an accuracy of 75% on a carefully curated dataset, which is probably less when we use it on real-time images. So, we have found ways to improve the algorithm, some possible changes that can be implemented include increasing the number of images in the dataset, changing the models that generate the embeddings, improving the KNN search algorithm in opensearch.
- **Implementing the improved Spark RDD code**: We have implemented some modifications to the Spark code that does all the processing in our cluster. We have changed from a UDF based approach to a RDD based approach. Currently, the code is saving all the RDD dataframes on our disk, so after a certain point in time the disk space in the machine is overloaded, resulting in the machine going down. So we have to find a way to delete the data frames so that code keeps on running, and we can implement that in the production system.
- **Recognition by entire body**: We wanted to implement recognition by the entire body, but we haven't found an algorithm that generates embedding that is able to recognise a person by those embeddings. So we have to find a model that is capable of recognising a person by its body embeddings accurately. We can also try to use a training dataset to train our models instead of using pre-trained models.
- **Make the UI more fast and responsive**: Currently, the UI is lagging and the responses take a lot of time to appear. So we have to make the UI much more fast and responsive.

References

1. Turnbull, J. *The Docker Book* (2016).
2. Poulton, N. *The Kubernetes Book* (2017). http://leanpub.com/thekubernetesbook.
3. Narkhede, N., Shapira, G. & Palino, T. *The Defi nitive Guide REALLTIME DATA AND STREAM PROCESSING AT SCALE* ().
4. *Spark Architecture* https://www.javatpoint.com/apache-spark-architecture.
5. Chambers, B. (A. & Zaharia, M. *Spark : the definitive guide : big data processing made simple* 576. ISBN: 9781491912218 ().
6. Gormley, C. & Tong, Z. *Elasticsearch the definitive guide : a distributed real-time search and analytics engine* ISBN: 9781449358549 ().
7. Et. al, J. G. *InsightFace: 2D and 3D Face Analysis Project* https://github.com/deepinsight/insightface. 2021.
8. Guo, J., Deng, J., Lattas, A. & Zafeiriou, S. Sample and Computation Redistribution for Efficient Face Detection. *arXiv preprint arXiv:2105.04714* (2021).
9. An, X. *et al. Partial FC: Training 10 Million Identities on a Single Machine* in *Arxiv 2010.05222* (2020).
10. Deng, J., Guo, J., Liu, T., Gong, M. & Zafeiriou, S. *Sub-center ArcFace: Boosting Face Recognition by Large-scale Noisy Web Faces* in *Proceedings of the IEEE Conference on European Conference on Computer Vision* (2020).
11. Deng, J., Guo, J., Ververas, E., Kotsia, I. & Zafeiriou, S. *RetinaFace: Single-Shot Multi-Level Face Localisation in the Wild* in *CVPR* (2020).
12. Guo, J., Deng, J., Xue, N. & Zafeiriou, S. *Stacked Dense U-Nets with Dual Transformers for Robust Face Alignment* in *BMVC* (2018).
13. Deng, J. *et al.* The Menpo benchmark for multi-pose 2D and 3D facial landmark localisation and tracking. *IJCV* (2018).
14. Deng, J., Guo, J., Niannan, X. & Zafeiriou, S. *ArcFace: Additive Angular Margin Loss for Deep Face Recognition* in *CVPR* (2019).
15. Howard, A. *et al.* Searching for MobileNetV3.
16. Deng, J. *et al.* ArcFace: Additive Angular Margin Loss for Deep Face Recognition.
17. Miao, Y., Lattas, A., Deng, J., Han, J. & Zafeiriou, S. Physically-Based Face Rendering for NIR-VIS Face Recognition. https://github.com/deepinsight/insightface/tree/master/recognition.

18. Solomon, M. M., Meena, M. S., Kaur, J., Student, M. T. & Professor, A. CHALLENGES IN FACE RECOGNITION SYSTEMS. **6.** http://ijrar.com/.

19. Geerling, J. *Ansible for DevOps Server and configuration management for humans* ISBN: 978-0-9863934-0-2. http://leanpub.com/ansible-for-devops (2014).

20. *Spark Architecture* https://www.ecanarys.com/Blogs/ArticleID/401/The-Ansible-Architecture.

21. Zaharia, M. *et al.* This open source computing framework unifies streaming, batch, and interactive big data workloads to unlock new applications. *COMMUNICATIONS OF THE ACM* **59** (11 2016).

22. Zaharia, M. *et al.* Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.

23. Wang, C.-Y., Bochkovskiy, A. & Liao, H.-Y. M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.

24. *Yolov5 Github* https://github.com/ultralytics/yolov5.

25. *Efficientnet Github* https://github.com/qubvel/efficientnet.

26. Tan, M. & Le, Q. V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *36th International Conference on Machine Learning, ICML 2019* **2019-June,** 10691–10700. https://arxiv.org/abs/1905.11946v5 (May 2019).

27. *OSnet Github* https://github.com/KaiyangZhou/deep-person-reid.

28. Zhou, K., Yang, Y., Cavallaro, A. & Xiang, T. Omni-Scale Feature Learning for Person Re-Identification. *Proceedings of the IEEE International Conference on Computer Vision* **2019-October,** 3701–3711. ISSN: 15505499. https://arxiv.org/abs/1905.00953v6 (May 2019).

29. Guo, G., Wang, H., Bell, D., Bi, Y. & Greer, K. KNN model-based approach in classification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **2888,** 986–996. ISSN: 16113349. https://link.springer.com/chapter/10.1007/978-3-540-39964-3_62 (2003).

30. Malkov, Y. A. & Yashunin, D. A. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **42,** 824–836. ISSN: 19393539. https://arxiv.org/abs/1603.09320v4 (4 Mar. 2016).

Appendix A

Body similarity

In this section, we will discuss about the other method of Person recognition that is recognition by the entire body or Person Re-Identification.

We observed many images in which a face was not detected but it detects a person in that image. So, not to waste those images we thought of having a body recognition algorithm. The algorithm would be capable of recognising a person by their entire body.

The first step to check that the model was working or not was through body similarity. In this method, we will take an image, detect a person in that image(done through yolov5) [23] [24] and take the embeddings of that person and search for images which have embeddings similar to that.

Many models have been tried in Coriolis before me which didn't lead to any good result. I focused my efforts on 2 of the most promising models.

- **Efficientnet**: EfficientNet is a family of deep neural network models designed to deliver state-of-the-art accuracy with much fewer parameters than conventional networks. EfficientNet can aid in the re-identification of individuals by facilitating more efficient feature extraction from photos.

  In person re-identification, the objective is to identify individuals across many camera views, even if their appearance may alter owing to changes in lighting, pose, clothes, or other factors. To do this, deep learning models are trained to extract characteristics from photos that are robust to these fluctuations and can be used to compare and match individuals across several perspectives.

  EfficientNet models are intended to attain high levels of accuracy with fewer parameters, allowing for more efficient extraction of features from huge datasets. This is particularly beneficial in human re-identification, where big datasets with numerous variations in appearance are frequently used to train deep learning models. [25] [26]

- **Osnet**: OSNet (Omni-Scale Network) is a deep learning network built for human re-identification, which entails detecting individuals across numerous camera perspectives. OSNet aids in re-identification of individuals in multiple ways: [27] [28]

    1. **Feature Embedding**: OSNet is meant to extract highly discriminative characteristics from photos that are robust to differences in lighting, position, and other factors that can affect the appearance of individuals across many camera perspectives. This facilitates matching individuals across several perspectives.
    2. **Cross-Resolution Matching**: OSNet is designed to extract features from input photos at several scales or resolutions. This enables it to collect features at various degrees of detail, which is crucial for human re-identification in situations where individuals can appear at various scales in various camera views.
    3. **Metric Learning**: OSNet employs metric learning to optimize the embedding space, meaning it learns to map photographs of the same person to similar feature vectors and images of different persons to dissimilar feature vectors. This facilitates the comparison and matching of feature vectors across several camera viewpoints.

However, both these models failed to give any meaningful results. We tried various variations of both these models: **EfficientNet B0**, **EfficientNet B1**, **EfficientNet V2S**, **Osnet_x1_0**, **Osnet_ibn_x1_0**, **Osnet_ain_x1_0**.

The are major challenges to identifying a person by its body embeddings that we faced during our project was:

- **Viewpoint variation**: Different viewpoints of same person due to different cameras leads models to believe that they are different person.
- **Illumination changes**: Lighting conditions might vary between camera viewpoints, making it difficult to match a person's appearance across photographs.
- **Intra-class variation**: Even within the same individual, there can be considerable variations in look, such as changes in dress, accessories, or hairdo.
- **Inter-class similarity**: Different individuals may share similar physical characteristics, making it difficult to identify between them.
- **Occlusion**: Individuals can be partially or completely obscured in an image, making it difficult to compare their appearance to other photographs.
- **Scale variation**: Humans can appear at various distances from the camera, resulting in variations in their perceived size and scale.

One potential solution, which we thought of implementing was taking a bunch of images which will have different person's whole body in them, annotating them manually and then train the models with that annotated data and using those trained models instead of using pre-trained models. However, due to lack of time we couldn't do that.

So, we decided to focus our efforts instead on the face recognition algorithm and making a working prototype of our software.

Appendix B

Exact KNN vs HNSW KNN

B.1 Exact KNN

K-Nearest Neighbor (KNN) is a classification algorithm that falls under the category of instance-based or memory-based learning. However, for our project we used KNN algorithm as a similarity search algorithm.

Instance-based or memory-based learning is a sort of machine learning in which the model memorizes the complete training dataset rather than being explicitly trained on a set of labeled data. In this method, the model retains all labeled data in memory and classifies new data points based on their similarity to the training data.

In KNN, the model memorizes the full training dataset rather than learning any parameters. When a new data point must be classified, KNN analyzes the distance between the new data point and all of the points in the training dataset and selects the K-nearest neighbors. Next, KNN assigns the new point's class label based on the majority class of its K-nearest neighbors. [29]

Algorithm used in our case:

1. **Input**: The algorithm's input consists of a collection of labeled data points and a query point.
2. **Distance Metric**: The algorithm calculates the distance between the query point and each of the data points using a distance metric such as Euclidean distance, Manhattan distance, or Minkowski distance.
3. **K-nearest neighbors**: Based on the estimated distances, the algorithm finds the K-nearest neighbors of the query point.
4. **Similarity searcch**: Based on the scores generated on the above steps we find the points that are most similar to the query vector.

Although the output generated from the exact KNN search would be more accurate, however, it takes a lot of computational power to generate the results.

B.2 HNSW

The Hierarchical Navigable Small World (HNSW) technique is a variation of the K-Nearest Neighbor (KNN) algorithm designed for fast search of nearest neighbors in high-dimensional spaces. HNSW is a memory-based technique that stores data points in a hierarchical data structure and enables rapid and scalable KNN search.

1. **Build the graph**: The algorithm begins by constructing a network in which each node represents a data point and each edge reflects the similarity between two data points. The graph is generated with a variation of the approach for randomized partition trees.
2. **Build the hierarchy**:The graph is subdivided into numerous levels in order to produce a hierarchical structure. Each successive level comprises an increasing number of nodes, beginning with a tiny number of nodes at the top level.
3. **Assign connections**: Each node is assigned a predetermined number of connections to other nodes on the same level and the next level.
4. **Indexing**:The data points are indexed utilizing a hierarchical framework. Each point is assigned to a leaf node at the highest level of the hierarchy, and its connections are assigned to the nodes at the next level.
5. **Search**: To do a KNN search, the algorithm begins at the top of the hierarchy and navigates through the nodes to locate the query point's nearest neighbors. At each level, the method employs a heuristic to decide the following node to visit depending on its distance from the query location.

HNSW is a much faster algorithm compared to the exact KNN search but it is less accurate. [30]