Ain Shams University
Faculty of Computer & Information Sciences
Scientific Computing Department

Content-Based Video Search Engine

July 2021

Ain Shams University
Faculty of Computer & Information Sciences
Scientific Computing Department

Content-Based Video Search Engine

This documentation is submitted as required for the degree of bachelors in Computer and Information Sciences.

By

Andrew Awni Nasri	Scientific Computing Department
Ganna Nayef Kamel	Scientific Computing Department
Ahmed Mohammed Abdulazim	Scientific Computing Department
Nada Saeed Mohamed	Scientific Computing Department
Ahmed Mahmoud Korani	Scientific Computing Department

Under Supervision of

Dr. Dina Khattab	Scientific Computing Department, Faculty of Computer and Information Sciences, Ain Shams University
T.A. Youmna Ahmed	Computer Science Department, Faculty of Computer and Information Sciences, Ain Shams University
T.A. Hager Ismail	Scientific Computing Department, Faculty of Computer and Information Sciences, Ain Shams University

July 2021

Acknowledgments

We would love to seize this opportunity to express our thanks and appreciation to everyone who provided us with support, guidance, and invaluable constructive criticism that helped us shape this project into what it is.

We offer our sincerest gratitude to our supervisors, Dr. Dina Khattab, T.A. Youmna Ahmed, and T.A. Hager Ismail for their tremendous help and support throughout the whole course of this project.

Abstract

Media files retrieval and searching is a complex process due to the variety in content and queries. All available video search engines enable searching for video by title, description, and metadata written by the publisher.

In general, video content is richer than a single image and platforms don't have all the info about the videos from just the publishers, in addition to that, most of the internet traffic is unlabeled videos that we need to understand its content to analyze it for personalized ads, security and many other use cases, so automatic indexing and retrieval are considered in multimedia research.

Achieving a general CBVS engine is impossible due to the limitation of building a model that can recognize and understand everything in the world.

In our project, we provide a modular system with interfaces to allow developers and scientists to integrate with it and feed the system with any models (statistical-based, machine learning, or any sub-program) so the system can use any number and combination of them in automatic distributive indexing environment, where end-user can just enter their query and get the results.

HOG-based facial recognition and COCO with YOLO 80 classes are built as two independent models to test and measure system modularity and architecture.

The architecture is designed to enable high scalability by splitting the video processing phases into 6 main phases: segmentation, keyframes extraction, feature extraction, processing engine, indexing, and query/retrieval.
Each phase can be deployed independently on separate nodes to represent a layer of servers that receive input from the previous layer and send the output to the next layer. That design enabled us to run the whole process on the same dataset with a different number of processes to start with 29 mins to index using 1 process and end with 7 mins indexing time on 5 processes.

Table of Contents

List of Figures

List of Abbreviations

API	Application Programming Interface
CBVR	Content-Based Video Retrieval
CBVS	Content-Based Video Search
CNN	Convolutional Neural Network
COCO	Common Objects in Context Dataset
Conv	Convolutional
DDNNN	Divide-and-Conquer Dual-Architecture Convolutional Neural Network
FC	Fully-Connected
FPN	Feature Pyramid Network
HOG	Histogram of Oriented Gradients
IoU	Intersection over Union
LDA	Linear Discriminant Analysis
NN	Neural Network
OCR	Optical Character Recognition
ORM	Object-Relational Mapping
PCA	Principal Component Analysis
SIFT	Scale Invariant Feature Transformation
SQL	Structured Query Language
SVM	Support Vector Machine
YOLO	You-Only-Look-Once Algorithm

Chapter 1: Introduction

1.1 Problem Definition
1.2 Motivation
1.3 Objectives
1.4 Time Plan
1.5 Documentation Outline

Content-based search means searching for videos by keywords or an image referring to the meta-data of the desired video to be retrieved from a large database which is videos after being analyzed and indexed.

Although the term search engines are often used indiscriminately to describe crawler-based search engines, human-powered directories, and everything in between, they are not all the same. Each type of search engine gathers and ranks listings in radically different ways.

Crawler-based search engines such as Google, compile their listings automatically. They crawl or spider the web, and people search through their listings. These listings are the structure of the search engine's index or catalog. One can think of the index as a massive electronic filing cabinet containing a copy of every web page the spider finds. Because spiders scour the web on a regular basis, any changes made to a web site may affect search engine ranking. It is also important to remember that it may take a while for a spidered page to be added to the index. Until that happens, it is not available to those searching with the search engine.

1.1 Problem Definition

Content-based means that the search analyzes the content rather than the meta-data such as keywords, tags, or descriptions.

The title of the video does not often describe the real content of the video. The meta-data of the video describes the content of the video as it consists of speech, words, events and actions in the video.

The internet hosts an enormous amount of random, unlabeled videos, many with little to no information about their content and most of video hosting platforms often offer keyword-based search engines referring to the video titles, not content based. Many videos indexing and labeling solutions still require hours of manual human labor and all the available content based search engines are domain specific and not reusable.

Cisco made a research that by 2022 more than 82% of the internet traffic will be online videos.

1.2 Motivation

Traditional search engines only depend on texts, titles, description and metadata in the filtration process, it doesn't understand nor index based on the content of media files. It's also impossible to develop a generic system that has the ability to understand all possible actions, objects and other components of videos, so all content-based video search systems are domain specific and are applied in many use cases like:

- Quick browsing of video folders
- Analysis of visual electronic commerce
- News event analysis
- Intelligent management systems

1.3 Objectives

- A smart search engine that do the following procedures based on the models provided to the system:
    - Analyzes videos
    - Filters videos
    - Retrieves videos
- Any number of models can be plugged into the system for different analysis and data (videos) collection based on the desired use case and requirements.
- Other systems (like surveillance systems) can integrate with the system and connect its video repository with the system to parse and index them based on the given models.

1.4 Time Plan

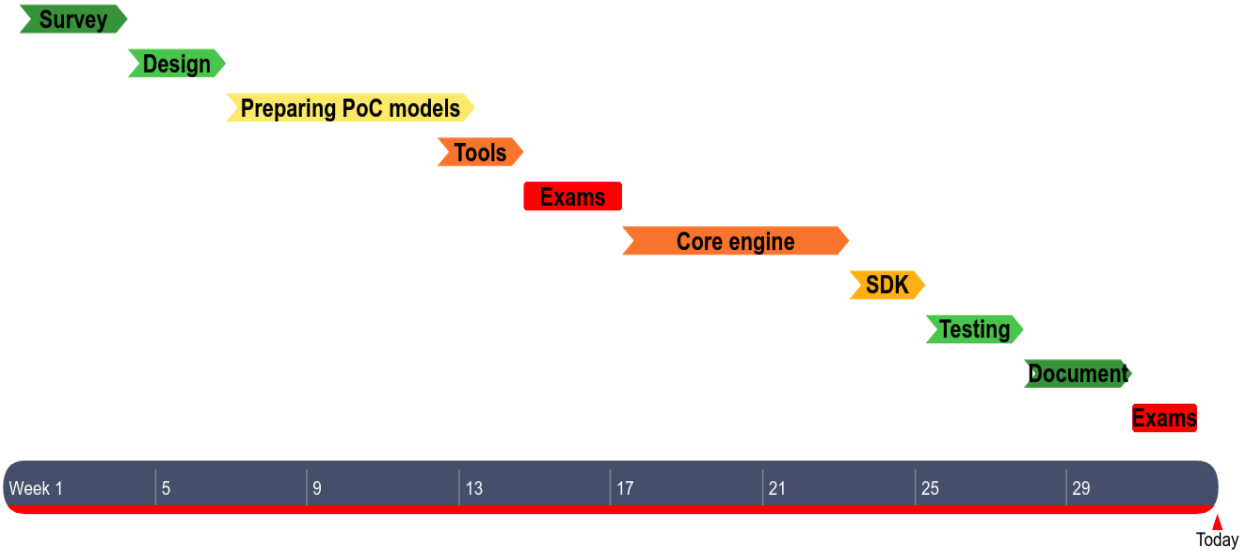


Fig. 1.1: Time Plan

Task	Duration
survey and related previous works search	2 weeks
System microservices architecture	2 weeks
Preparing (selecting the appropriate dataset and algorithms) and training at least two models for the proof of concept	1 month
Tools research	2 weeks
Building the video indexer (core engine)	3 week
Developing the models' wrapper interface (SDK)	2 weeks
Testing and verification	2 weeks
Documentation	2 weeks

1.5 Documentation Outline

Chapter 2: Background
This chapter includes background about the project, basic concepts, and the related work according to our research.

Chapter 3: System Architecture and Design
This chapter includes an overview of the whole system along with the intended user, system architecture, analysis and design.

Chapter 4: System Implementation
This chapter describes the system functions with details about the implementation of the project's modules.

Chapter 5: Experiments
This chapter shows our experiments, tells the numbers and performance we get from our system and explains design decisions taken based on these trials.

Chapter 6: System Testing
This chapter talks about the needed packages and libraries that must be installed before using the application, and also shows the user how to use it.

Chapter 7: Conclusion & Future Work
This chapter includes the conclusion and results of our work and the future work that may be done based on this project.

Chapter 2: Background

2.1 Content-Based Video Retrieval

An approach for facilitating the searching and browsing of large image collections over the World Wide Web or Large Datasets. In this approach, video analysis is conducted on low level visual properties extracted from video frames. It has been increasingly used to describe the process of retrieving desired videos from a large collection on the basis of features that are extracted from the videos. The extracted features are used to index, classify and retrieve desired and relevant videos while filtering out undesired ones.
In the following sections we will highlight all the steps taken by the video content analysis system in order to carry out its mission.

2.1.1 Video Segmentation

Video segmentation is the process of partitioning a video sequence into disjoint sets of consecutive frames that are homogeneous according to some defined criteria. In the most common types of segmentation, video is partitioned into shots, camera-takes, or scenes. Multiple techniques can be applied to segment videos like pixel differences, statistical differences, histograms, and machine learning for semantic segmentation.

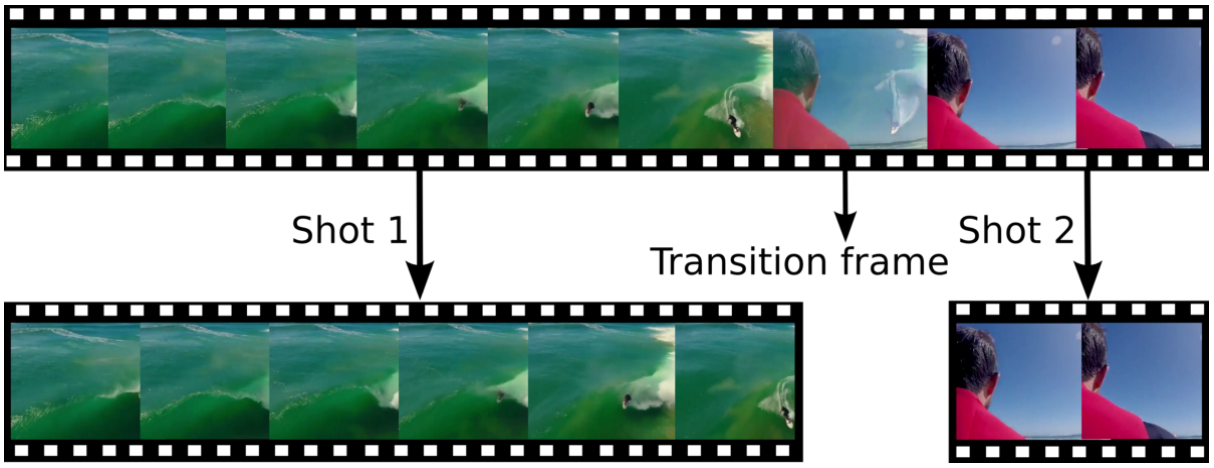


Fig. 2.1: Shot Boundary Detection

2.1.1.1 Pixel Differences / Thresholding

The easiest way to detect if two frames are significantlydifferent is to count the number of pixels that change in value more than some threshold.

$$D(k,k+1) = \sum_{i,j} |I_k(i,j) - I_{k+1}(i,j)|,$$

This total is compared against a second threshold to determine if a shot boundary has been found.

2.1.1.2 Statistical Differences

Statistical methods expand on the idea of pixel differences by breaking the images into regions and comparing statistical measures of the pixels in those regions. For example, compute a measure based on the mean and standard deviation of the gray levels in regions of theimages. This method is reasonably tolerant of noise, but is slow due the complexity of the statistical formulas. It also generates many false positives i.e. changes not caused by the shot boundary.

2.1.1.3 Machine Learning / Deep Learning

CNN can be used in shot boundary detection, one of the proposed CNN:

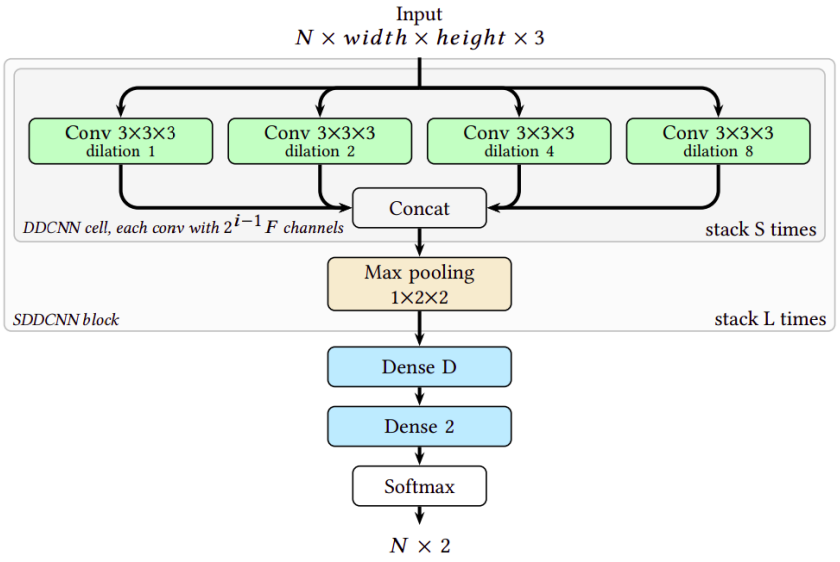


Fig. 2.2: CNN Architecture

The main building block of the model (Dilated DCNN cell) is designed as four 3D 3×3×3 convolutional operations. The convolutions employ different dilation rates for the time dimension and their outputs are concatenated in the channel dimension. This approach significantly reduces the number of trainable parameters compared to standard 3D convolutions with the same field of view. Multiple DDCNN cells on top of each other followed by spatial max pooling form a Stacked DDCNN block.

2.1.1.4 Histograms

Histograms are the most common method used to detect shot boundaries. The simplest histogram method computes gray level or color histograms of the two images. If the bin-wise difference between the two histograms is above threshold, a shot boundary is assumed. The color histogram changes rate to find shot boundaries.

The threshold calculation which will determine the shot changes or even in the case of a pre-processing. In most cases the similarity measure is calculated according to the equation.

$$HD_k = \frac{1}{N} \sum_{r=0}^{2^B-1} \sum_{g=0}^{2^B-1} \sum_{b=0}^{2^B-1} |p_k(r,g,b) - p_{k-1}(r,g,b)$$

where pk(r,g,b) is the number of pixel colors (r, g, b) in the frame k of N pixels. If this distance is greater than a predefined threshold, a cut is detected.

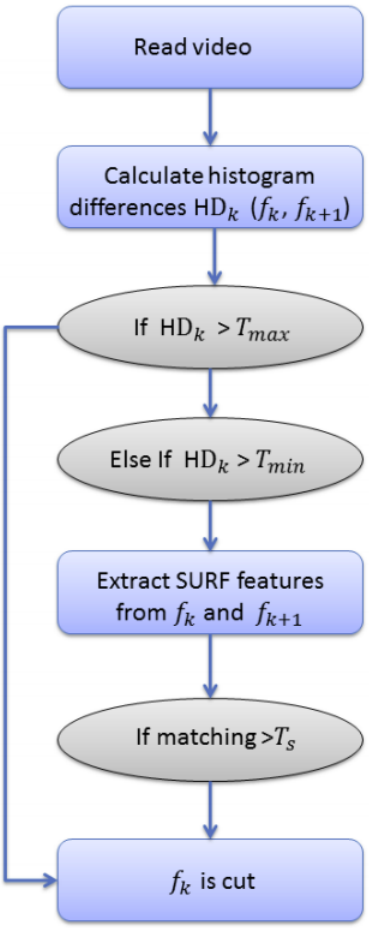


Fig. 2.3: Histograms method flow chart

2.1.2 Feature Extraction and Classification

This task extracts features that are typically used as indexes for CBVR systems. Three abstraction levels are usually considered to categorize video features: raw data, descriptors and concepts. Descriptors and concepts are also known as low-level and high-level features, respectively. If different levels are taken into account, a more complete characterization of video can be obtained. However, using a high amount of features may cause the curse of dimensionality, demanding dimensionality reduction approaches.
Descriptors can be applied to characterize several video elements, such as keyframes, movement and objects, i.e., relevant components within a specific domain, such as caption text or human face. Some descriptors are also used for image processing in general. Usual descriptors include a bag of visual words.
Concepts or semantic indexes are assigned to videos by differ- ent approaches, such as manual or automatic annotation. The idea is to associate segments, objects or events, complex activities that can be directly noticed and occur in specific locales and time with pre-defined semantic categories. Automatic annotation in particular is often supported by ML algorithms.
In summary, these intelligent techniques are able to learn patterns (models) from video features (usually descriptors). As a result, each input video is annotated with one or more concepts (classes) associated with the time interval describing when that class occurs.

2.1.2.1 Dimensionality Reduction

In real-world machine learning problems, there are often too many factors (features) on the basis of which the final prediction is done. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, many of these features are correlated or redundant. This is where dimensionality reduction algorithms come into play.

Dimensionality reduction is the process of reducing the number of random features under consideration, by obtaining a set of principal or important features.

Dimensionality reduction can be done in 2 ways:
A. Feature Selection: By only keeping the most relevant variables from the original dataset.
i. Correlation
ii. Forward Selection
iii. Backward Elimination
iv. Select K Best
v. Missing value Ratio

B. Feature Extraction: By finding a smaller set of new variables, each being a combination of the input variables, containing basically the same information as the input variables.
i. PCA
ii. LDA
PCA is a method of obtaining important variables (in the form of components) from a large set of variables available in a data set. It tends to find the direction of maximum variation (spread) in data. PCA is more useful when dealing with 3 or higher-dimensional data.
PCA can be used for anomaly detection and outlier detection because they will not be part of the data as it would be considered noise by PCA.

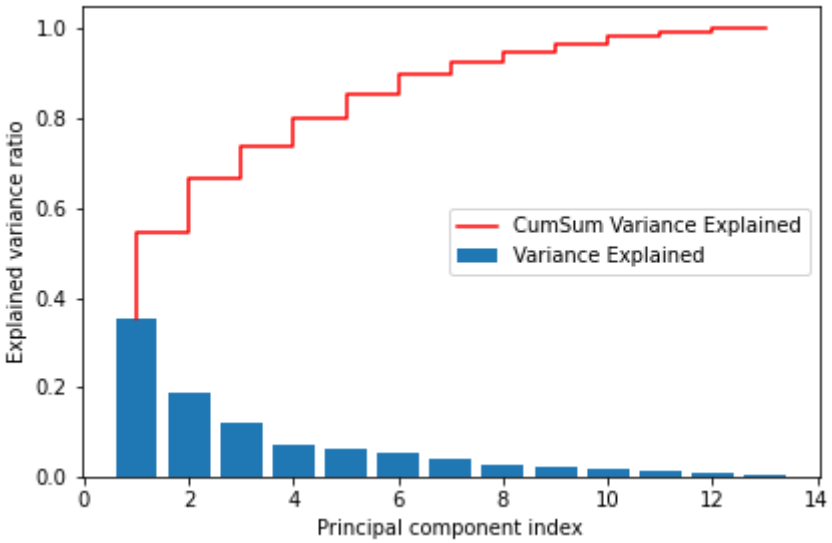


Fig. 2.4: Number of principal components vs captured data percentage

We can infer from the above Fig. that from the first 6 Principal Components we are able to capture 80% of the data.
This shows us the Power of PCA that with only using 6 features we are able to capture most of the data.

2.1.3 Results Indexing

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

2.2 Microservice Architecture

Microservices are an architectural style in which the process of software development is done by using autonomous components that isolate fine-grained business functionalities and communicate one with other through standardized interfaces. Due to an extensive use in web and cloud-based applications, we can observe a migration of some companies from the monolith architecture to microservices since the latter brings many benefits such as self-manageable (decentralized governance) and lightweight components.

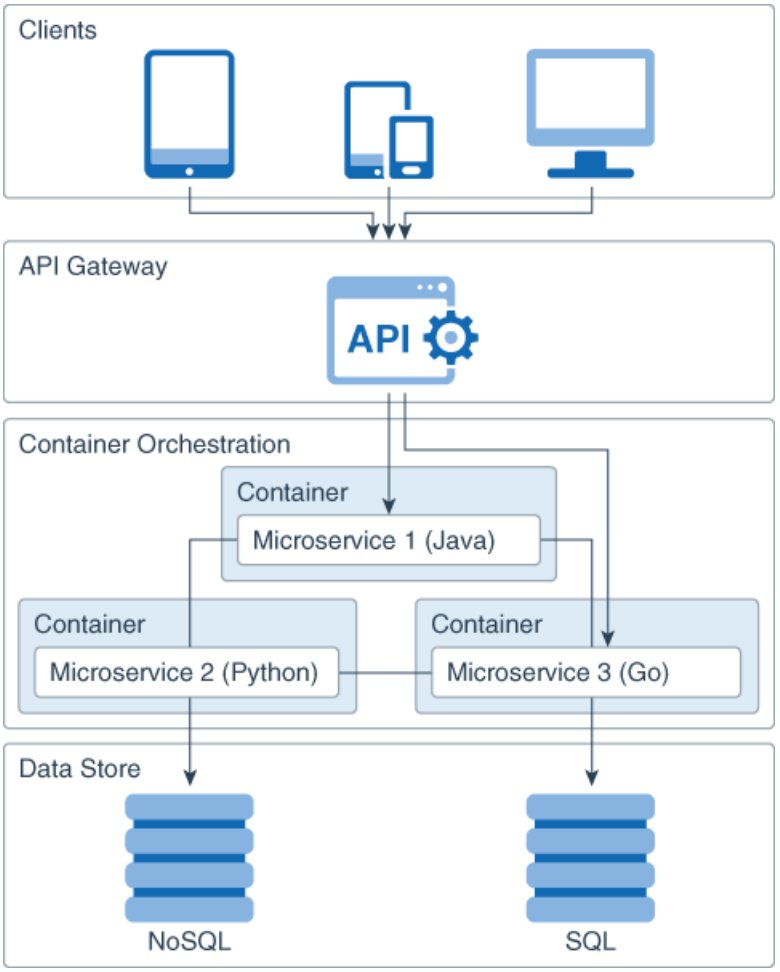


Fig. 2.5 : Microservice generic architecture

The purpose of microservices is to use autonomous units that are isolated one from another and coordinate them into a distributed infrastructure by a lightweight container technology, such as Docker. Usually, the adoption of this architectural model implies also in adopting agile practice, such as DevOps, which reduces the time between implementing a change in the system and transferring this change to the production environment.

2.3 YOLO

We take the help of our Eyes to see everything, it captures the information in the frame and sends it to our brain to decode and draw meaningful inferences from it. Well, it sounds pretty simple. We just look and we get the understanding of what are all the objects we're looking at, how they're placed, and tons of other information about them. But the processing our brain does for it is just beyond comparison. This is the approach that YOLO tries to achieve as the object predictions in the entire image is done in a single run. YOLO is an algorithm that detects and recognizes various objects in a picture. Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected objects in image. The objects' prediction in the entire image is done in a single algorithm run using the CNN to predict various class probabilities and bounding boxes simultaneously.

2.3.1 Residual blocks

First, the image is divided into various grids. Each grid has a dimension of S x S.

Fig. 2.6: An input image after dividing into S x S cells

2.3.2 Bounding Box Regression

After locating the exact position of the object in the image and detecting which class it belongs to it is used to highlight this object using these attributes:

- Width (bw)
- Height (bh)
- Class (c)
- Bounding box center (bx, by)
- Bounding box center (bx,by)

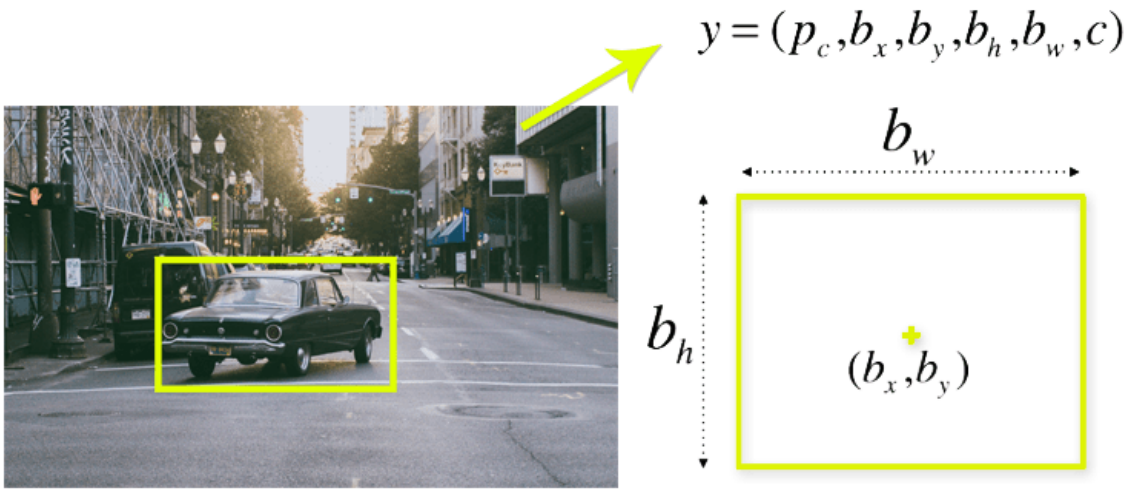


Fig. 2.7: Yolo proposed bounding box showing the 5 attributes

2.3.3 Intersection over union (IOU)

An evaluation metric used to measure the accuracy of the object detector. In which, each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box.

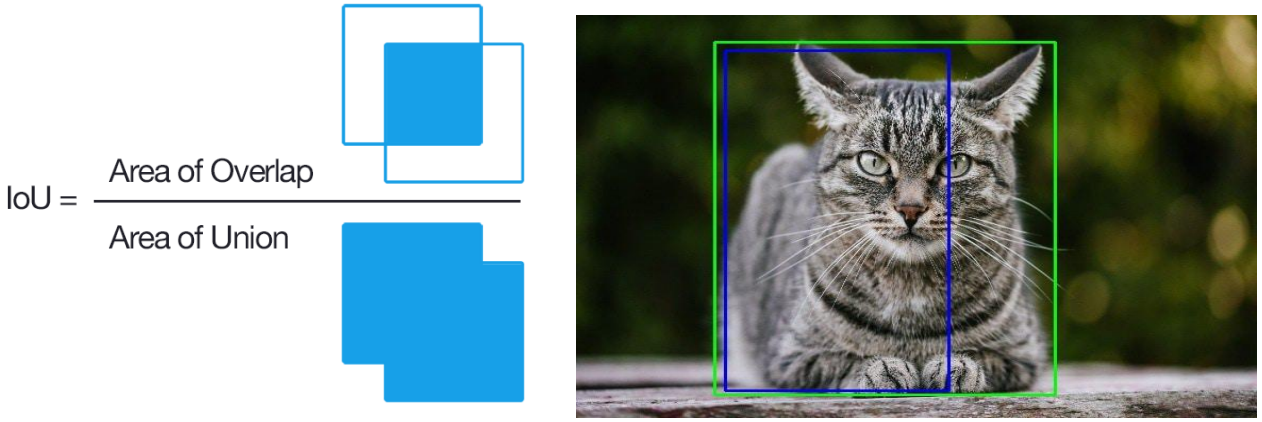


Fig. 2.8 : Intersection over union visualized example

The Overall Workflow:

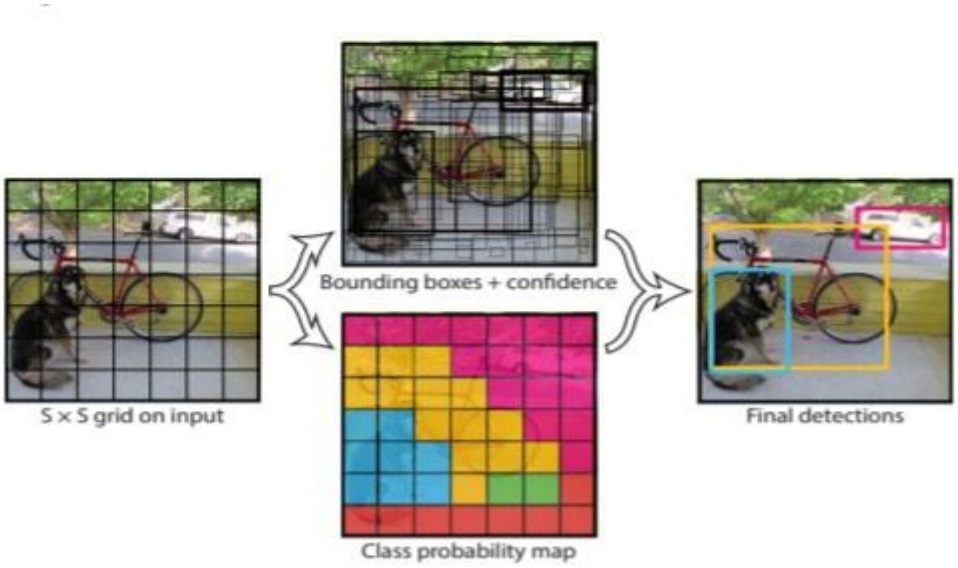


Fig. 2.9: Summary of yolo steps to detect and localize objects

2.3.4 Architecture

YOLO-V2 Architecture:

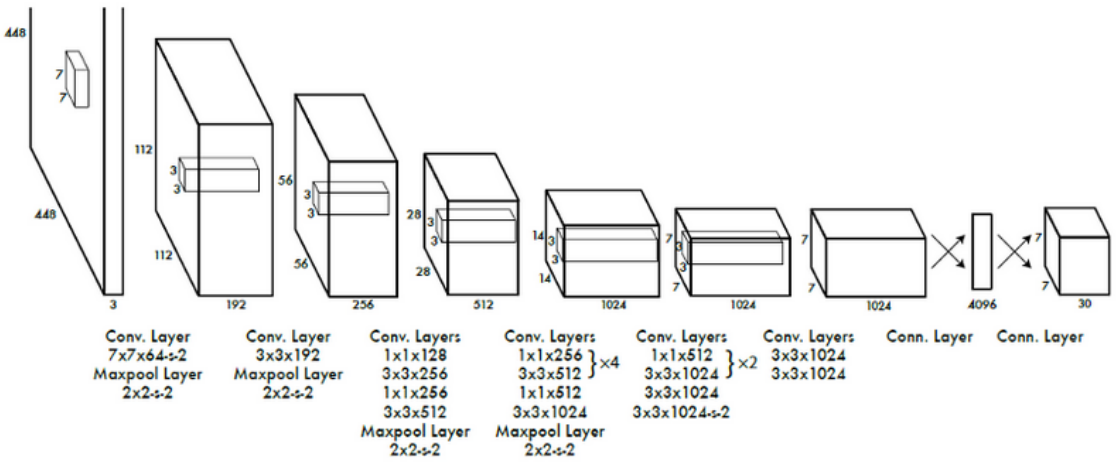


Fig. 2.10: YOLO-V2 architecture (source: towardsdatascience yolo-v2 explained)

YOLO-V3 Architecture:

Inspired by *ResNet* and *FPN* architectures, *YOLO-V3 feature extractor*, called *Darknet-53* (it has 52 convolutions) contains skip connections (like ResNet) and 3 prediction heads (like FPN) — each processing the image at a different spatial compression.

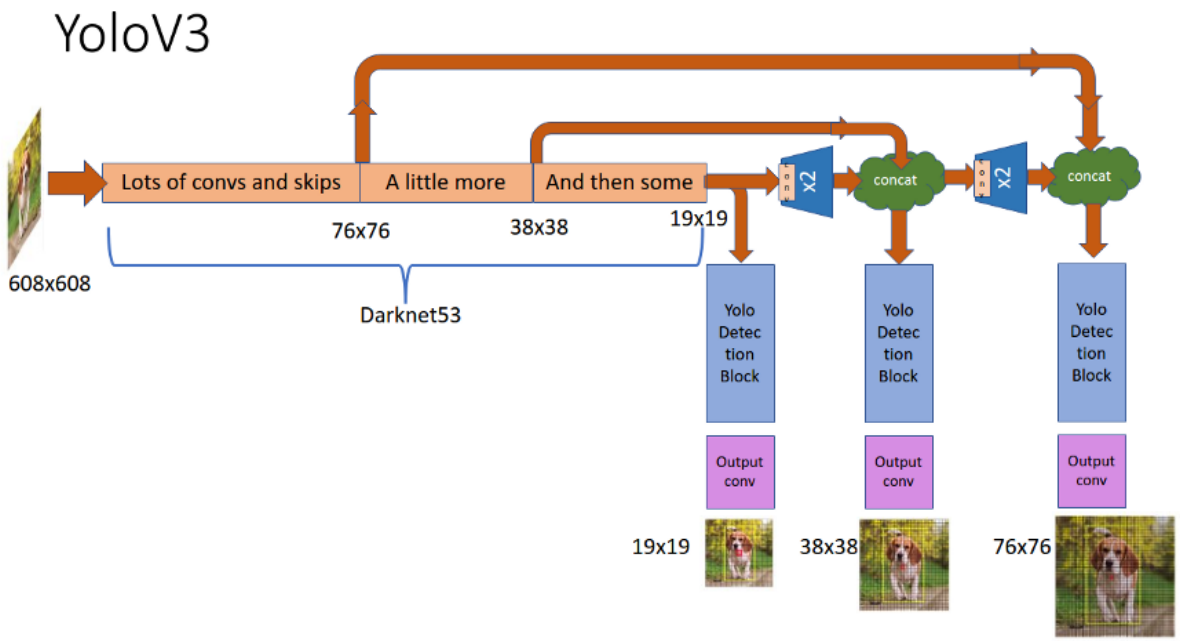


Fig. 2.11: YOLO-V3 architecture (source: towardsdatascience yolo-v3 explained)

FPN:

A Feature-Pyramid is a topology developed in 2017, in which the feature map gradually decreases in spatial dimension, but later the feature map expands again and is concatenated with previous feature maps with corresponding sizes. This procedure is repeated, and each concatenated feature map is fed to a separate detection head.

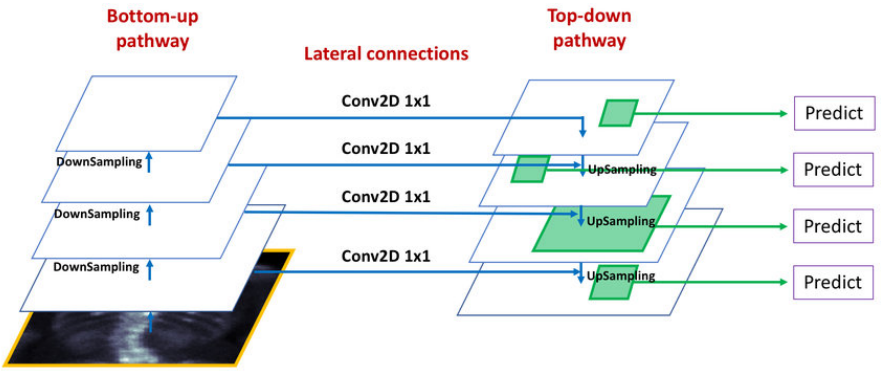


Fig. 2.12: Feature-pyramid network architecture

Darknet-53:

YOLO v2 used a custom deep architecture darknet-19, an originally 19-layer network supplemented with 11 more layers for object detection. With a 30-layer architecture, YOLO v2 often struggled with small object detections. This was attributed to loss of fine-grained features as the layers downsampled the input. To remedy this, YOLO v2 used an identity mapping, concatenating feature maps from a previous layer to capture low level features.

However, YOLO v2's architecture was still lacking some of the most important elements that are now staple in most of the state-of-the art algorithms. No residual blocks, no skip connections and no upsampling. YOLO v3 incorporates all of these.

First, YOLO v3 uses a variant of Darknet, which originally has a 53 layer network trained on Imagenet. For the task of detection, 53 more layers are stacked onto it, giving us a 106 layer fully convolutional underlying architecture for YOLO v3. This is the reason behind the slowness of YOLO v3 compared to YOLO v2.

2.4 Facial Recognition

Facial recognition is based on face descriptors. The system calculates the similarity between the input face descriptor and all face descriptors previously stored in a gallery. The goal is to find the face(s) from the gallery that are most similar to the input face.
The recognition of a face in a video sequence is split into three primary tasks: Face Detection, Face Prediction, and Face Tracking. The tasks performed in the Face Capture program are performed during face recognition as well. To recognize the face obtained, a vector of HOG features of the face is extracted. This vector is then used in the SVM model to determine a matching score for the input vector with each of the labels. The SVM returns the label with the maximum score, which represents the confidence to the closest match within the trained face data.

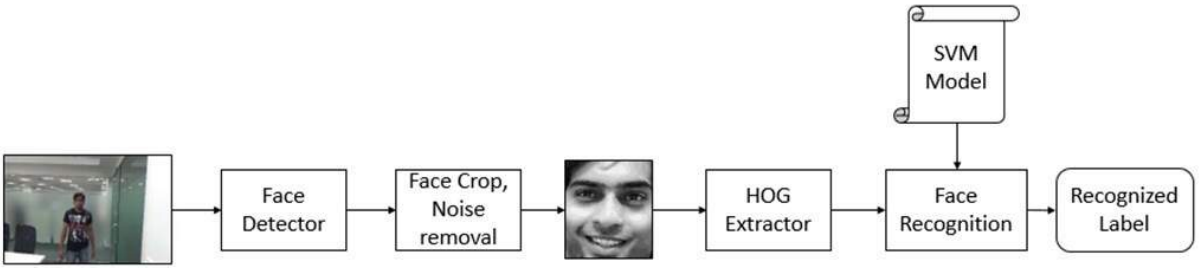


Fig. 2.13: Block diagram of the face recognition process (Source: eInfochips)

The task of calculating matching scores is exceptionally heavy to compute. Hence, once detected and identified, the labeled face in an image needs to be tracked to reduce the computation in future frames until the face eventually disappears from the video. Of all the available trackers, the Camshift tracking algorithm is used since it produces the best results with faces.

2.4.1 HOG

A HOG is a feature descriptor generally used for object detection. HOGs are widely known for their use in pedestrian detection. A HOG relies on the property of objects within an image to possess the distribution of intensity gradients or edge directions. Gradients are calculated within an image per block. A block is considered as a pixel grid in which gradients are constituted from the magnitude and direction of change in the intensities of the pixel within the block.

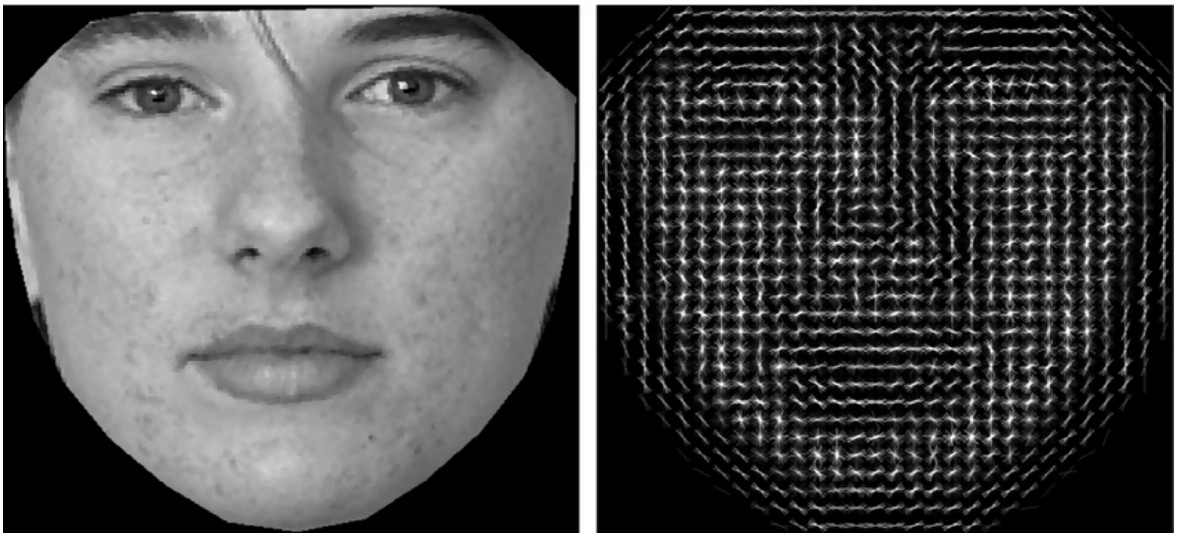


Fig. 2.14: HOG input-output visualization

2.4.1.1 HOG Algorithm

- HOG works on images of size 64x128
- The image is divided into 8×8 pixel size cells and the gradient magnitude and orientation is calculated for each pixel in all cells.
- A 9-bin histogram is created which corresponds to angles 0 – 180 where it calculates the “unsigned” gradients as a gradient and its negatives are represented by the same bin.
- A bin is selected based on the direction, and the value that goes into the bin is selected based on the magnitude.
- The image is divided to a bigger sized blocks of 16x16 pixels (2x2 cells), where each block has 4 histograms that are concatenated into a 1D vector of length 36
- The window is shifted each time by 8 pixels (1 cell) and the 36 length vector corresponding to each block is normalized to be illumination invariant.
- The window shift creates a size of 7 horizontal and 15 vertical blocks with a total of 105 blocks.
- All the 36 length normalized vector of the 105 blocks are concatenated in one giant vector of size 36x105=3780 dimensional vector which represents the feature vector of the HOG descriptor

2.5 Related Works

In recent years the need for content-based retrieval of image and video information from internet archives has created the attention of almost all researchers. Research efforts have caused the development of methods that provide access to image and video data. The methods are used to determine the similar things in the visual information content extracted from low level features. These features are clustered for creation of databases. All video searching algorithms reported in contemporary research are accessing the whole video in a stretch while the stakeholder is interested in a small portion of the video. However, the proposed methodology enables content wise retrieval of the video frames of interest only, considerably reducing the bandwidth. All of these systems are domain specific, they are shipped with pre-trained models that are totally coupled with the other system components, no separation and no flexibility to adapt these components based on the use case so any scientist or developer don't have to re-write all components to be able to develop their own CBVS system.
In the following sections we will discuss some of these domain specific papers, how they are built, what is the architecture, and which models are fully integrated with them.

2.5.1 Education domain

Cameras are used for recording a lecture therefore they apply State of Art Lecture recording systems such as TeleTeaching Anywhere Solution Kit. Lecture is conducted in two main parts. The first part is the main scene of the lecture is recorded by using a video camera and the second part is to capture the speech by desktop of the speaker's computer. The main drawback is that the video analysis may introduce errors due to synchronization between audio and video frames. Video segmentation method used is the SIFT feature and the required calculated threshold value. In previous work a SIFT feature was applied to measure slides with similar content. An adaptive threshold selection algorithm is used in their work to detect slide transitions. In their evaluation, this approach obtained promising results for processing one scene lecture video.
The synchronization problem is solved by proposing a solution which segments lecture video by analyzing its supplementary synchronized slides. The slides content arises automatically from OCR. Then partition the slides into different subtopics by examining their logical relevance. As the slides are synchronized with the lecture video, the subtopics of the slides indicate the segments of the video, then OCR results are the inputs for the whole procedure.

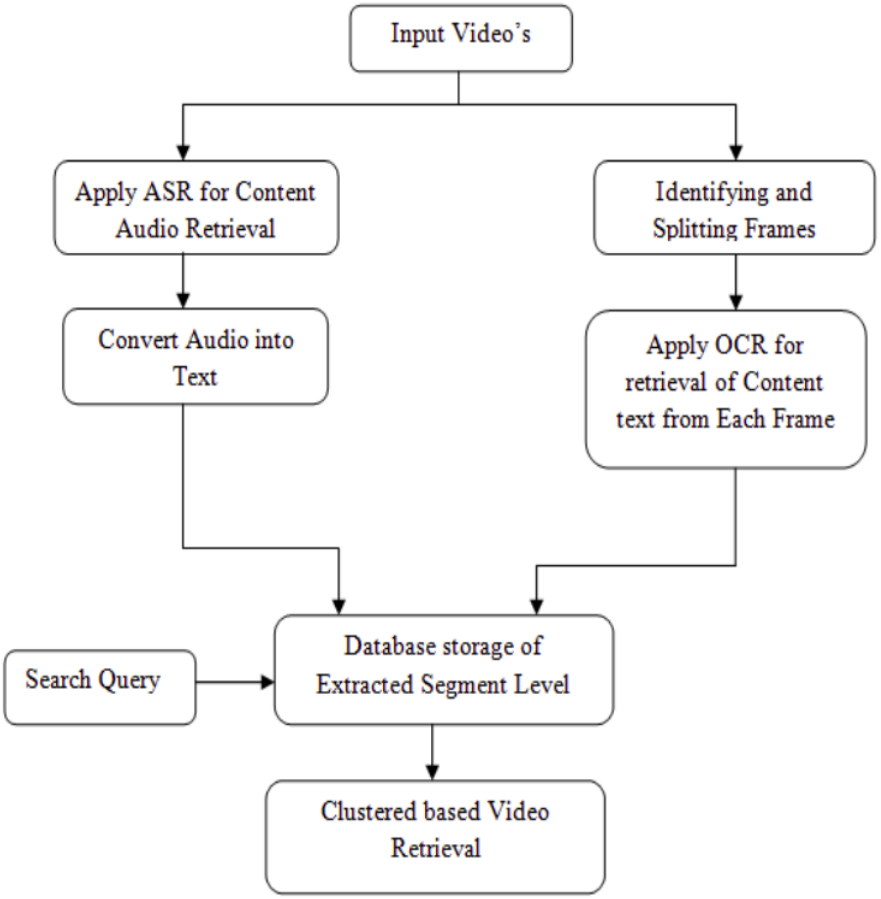


Fig. 2.15: Education system full architecture (source: IJARCCE, Content Based Video Retrieval Wagh K. D. et al.)

2.5.2 Agriculture Domain

In Vietnam, a group of scientists built a system for Vietnamese agricultural multimedia information retrieval. For each video crawled from the online sources, they demultiplex it into audio and visual channels, which are later segmented into a sequence of frames. The audio part gets manually transcribed to serve as a training corpus for building the ASR module. This in turn, performs a force-alignment procedure on all video files, making them annotated with timestamps and keywords. Now, they define a concept shot Fk as follows: Fk(t, d)~ derived frames clamped by keyword K begin at timestamp t and last for duration d.

With the pre-built agricultural ontologies O, they then proceed to extract the concept shots Fk-i defined by all keywords K-i existed in the ontologies, positioned by the timestamps generated from the ASR module. With this way, the video database is now chopped down into segments – a set of concept-shots. They also keep track of their contextual information by padding them with adjacent frames for a short leap Δt. Fk is then refined as:

$$F_{k\text{-}i}(t_i - \Delta t,\ d + 2\Delta t), \quad i \in [1 \ldots |O|], \quad k_i \in O$$

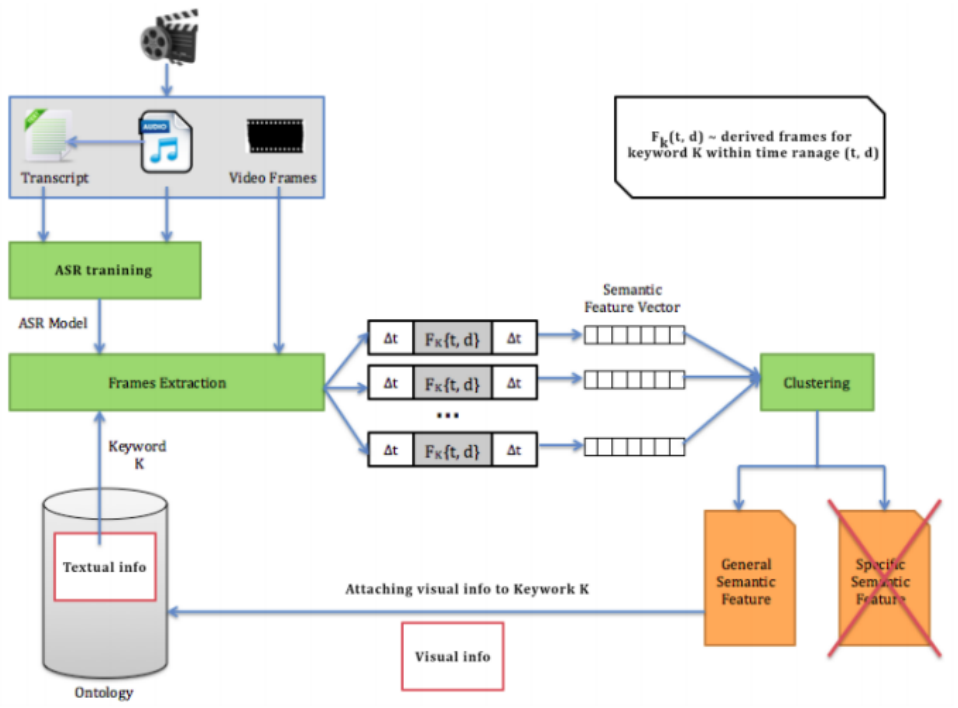


Fig. 2.16: Agricultural multimedia architecture (source: hybrid operations for content-based Vietnamese agricultural multimedia information retrieval)

Any future unseen media collected from the online sources will be auditorily transcribed and visually clustered into one of the available classes of ontology (i.e., keywords or concept-shots). The classification of concept-shots would definitely compensate for word-error-rates of the transcriptions, and ultimately tracking out missing info potentially available in the media.

2.5.3 Medicine

2.5.3.1 Monitoring of surgeries

In this context, an approach which aimed to develop a real time search tool for video sequences similar to a video recorded during an ophthalmic surgical procedure was presented. The researchers investigated the specific cases of epiretinal membrane and cataract surgeries. The technique involves identifying the surgical task being performed in the processed video sequence. According to the researchers, this approach can warn the surgeon as well as aid the professional to make a decision when an atypical or risk situation occurs during the procedure. The semantic gap was also a concern. The researchers used a feature weighting approach to improve the correlation between low-level features and semantic concepts related to surgical tasks. An approach aiming to automatic annotation of videos from monitoring of surgeries focused on medical education also was presented. This is a mixed approach of CBVR and keyword-based retrieval, in which the description of the video contents through extractors is used in assigning labels to them. The user uses keywords that refer to the labels assigned to find videos of interest.

2.5.3.2 Echocardiography

An approach to retrieve videos from echocardiography exams was presented. According to the researchers, this kind of video is an important source of information to aid cardiac diagnosis, being able to show the shape and movements of the heart from different angles. However, the feature extraction technique presented by the researchers is quite different from others mentioned in this paper. Features are extracted from the interpretation of texts featured throughout the video segment. The texts indicate measurements performed during the exam. An Optical Character Recognition (OCR) engine is used. The measures identified from the texts are used in the composition of feature vectors, which in turn allow measuring the similarity between videos.

Chapter 3: System Architecture and Design

3.1 System Overview

In this section, we introduce the entire architecture of the system.

The system consists of five separate subsystems that work in parallel completely independent of each other.

To achieve the modularity requirement, we introduced the concept of plugins, where a plugin is simply a class implementation of a base interface that accepts a resource as input and returns its detections/tags.

Each plugin operates independently of the system and is free to bundle its own dependencies, and can communicate with the engine through configuration files.

To demonstrate this concept, we bundled the system with two example plugins

1. Object Detection: based on YOLO
2. Face Recognition: based on dlib, custom trained to recognize some characters from The Office TV series

3.1.1 Architecture

We will explore the system modules one by one.

3.1.1.1 Data Feed

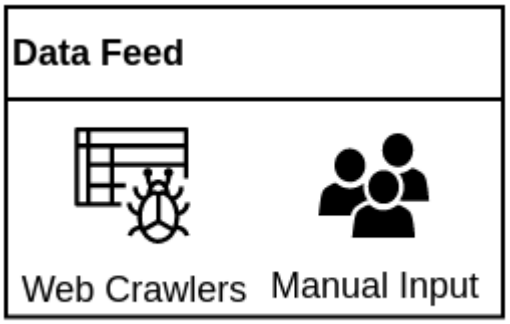


Fig. 3.1: Date feed independent component

The system that watches over the registered data sources, including local/remote directories in a file system, URLs that contain downloadable video content, or be a dump for some running web crawler processes.

It then converts them into structured units ready to be processed by the rest of the components.

3.1.1.2 Data Store

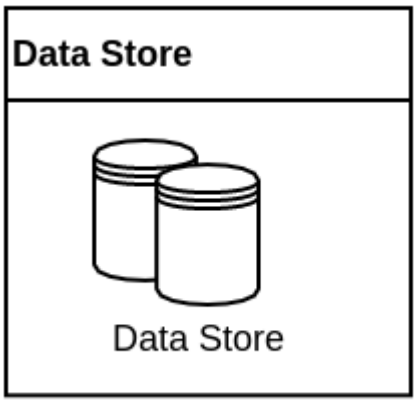


Fig. 3.2: Date store independent component

The storage layer that keeps the video resources and any useful information about them.

This could be a local file system, or a high performance distributed file system, or even a cloud object storage like AWS S3.

3.1.1.3 Processing Engine

The brains of the system.
This engine periodically pulls the resources waiting to be processed from the data store and feeds them into the processing cycle:

1.1. (optionally) perform the common structure analysis operations; video segmentation and keyframe extraction.
1.2. Invoke all the plugins on the currently processed resource
1.3. Aggregate the output of all the plugins into a standard format, passing it to the index server

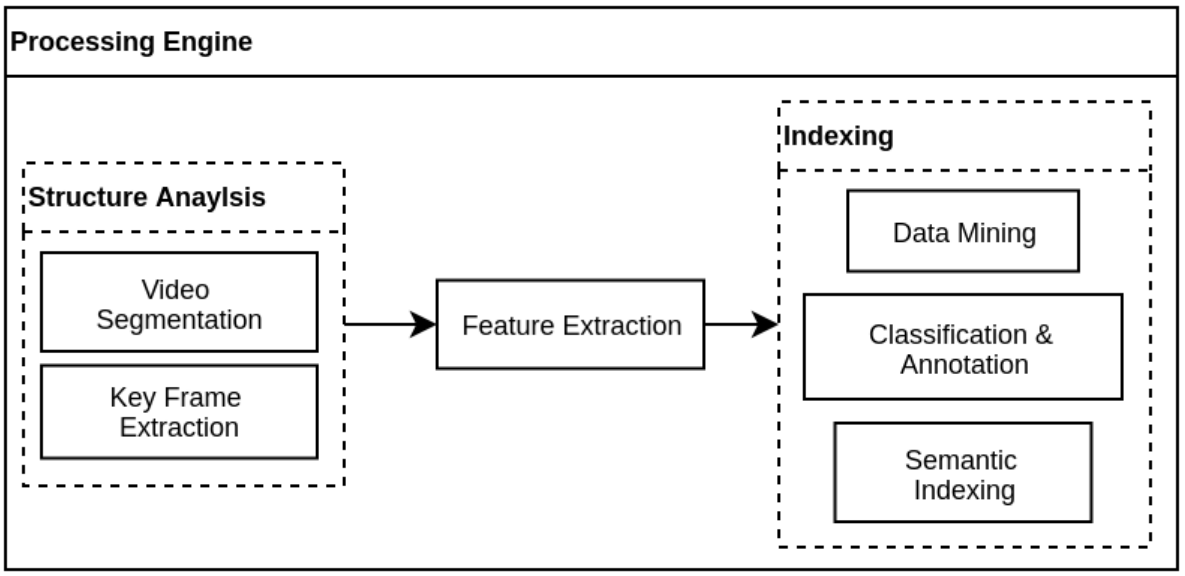


Fig. 3.3: Processing engine independent component

3.1.1.4 Integration Layer

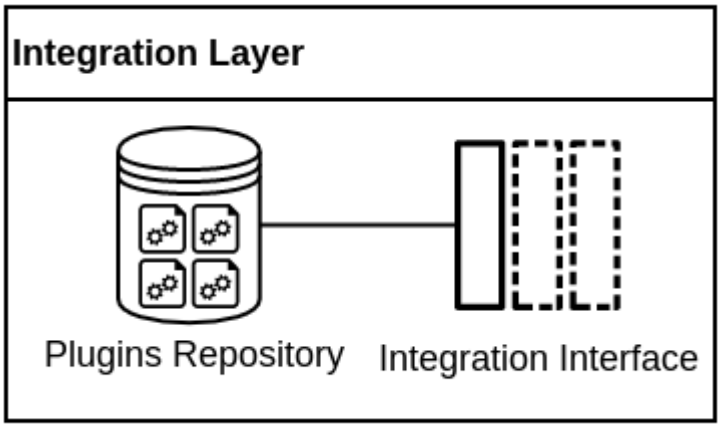


Fig. 3.4: Integration layer and plugins repository

This system is responsible for modifying the plugins, enabling/disabling them, and controlling their interactions with other parts of the system.

3.1.1.5 SDK

A utility that helps advanced users build and integrate their plugins.

Fig. 3.5: SDK independent component

3.1.1.6 Search Engine

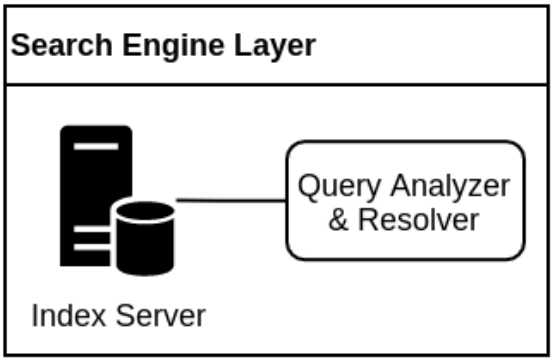


Fig. 3.6: Search engine independent component

A service that receives indexed data from the database and persists them into a high performance index server (e.g: elasticsearch).

It can also be bundled with a query analyzer to get more context from the queries requested by the application layer.

3.1.1.7 Application Layer

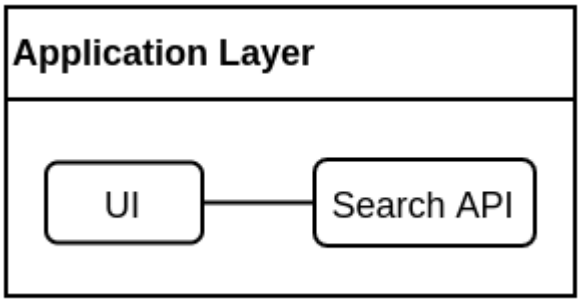


Fig. 3.7: End-user interface for performing search query

The user-facing part of the system consists of the UI and the API serving said UI.

3.1.1.8 The Big Picture

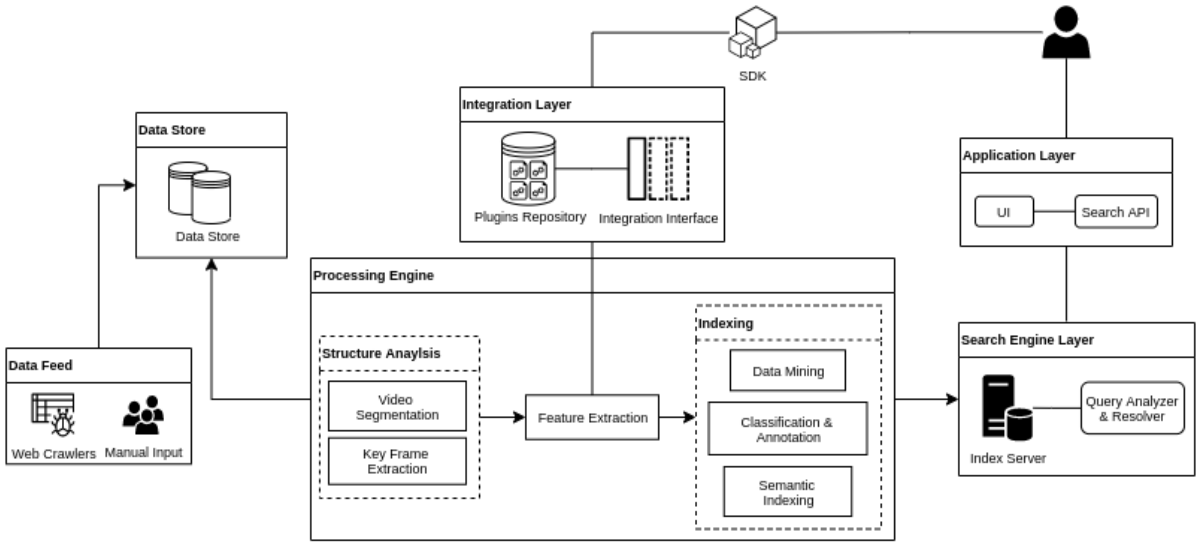


Fig. 3.8: All independent components relationships

Here we see how all the subsystems work together -yet independently- to provide the full functionality of a CBVSE.

- The data feed layer keeps running, seeding new discovered data into the storage layer
- The processing engine keeps pulling new data from the storage layer and performs the knowledge extraction cycle.
- The application layer is always presented to the user, ready to receive any queries and hit the search engine layer with them.
- The search engine layer is always watching over newly indexed data and updating its data structures.
- The integration layer is isolated and owns its resources, ready for any communication with the processing engine.
- The SDK is an on-demand service that can be invoked by the system administrator to integrato new plugins or modify existing ones.

3.1.3 System Users

1. **System Administrator**
   The user that owns the system and tailors it to their organization's use cases. They can create new plugins, enable/disable existing plugins, and modify their configuration settings.
2. **End-User**
   The user that benefits from the search functionality of the system.

3.2 System Analysis

3.2.1 Use Case Diagram

1. Scientist/developer use case diagram

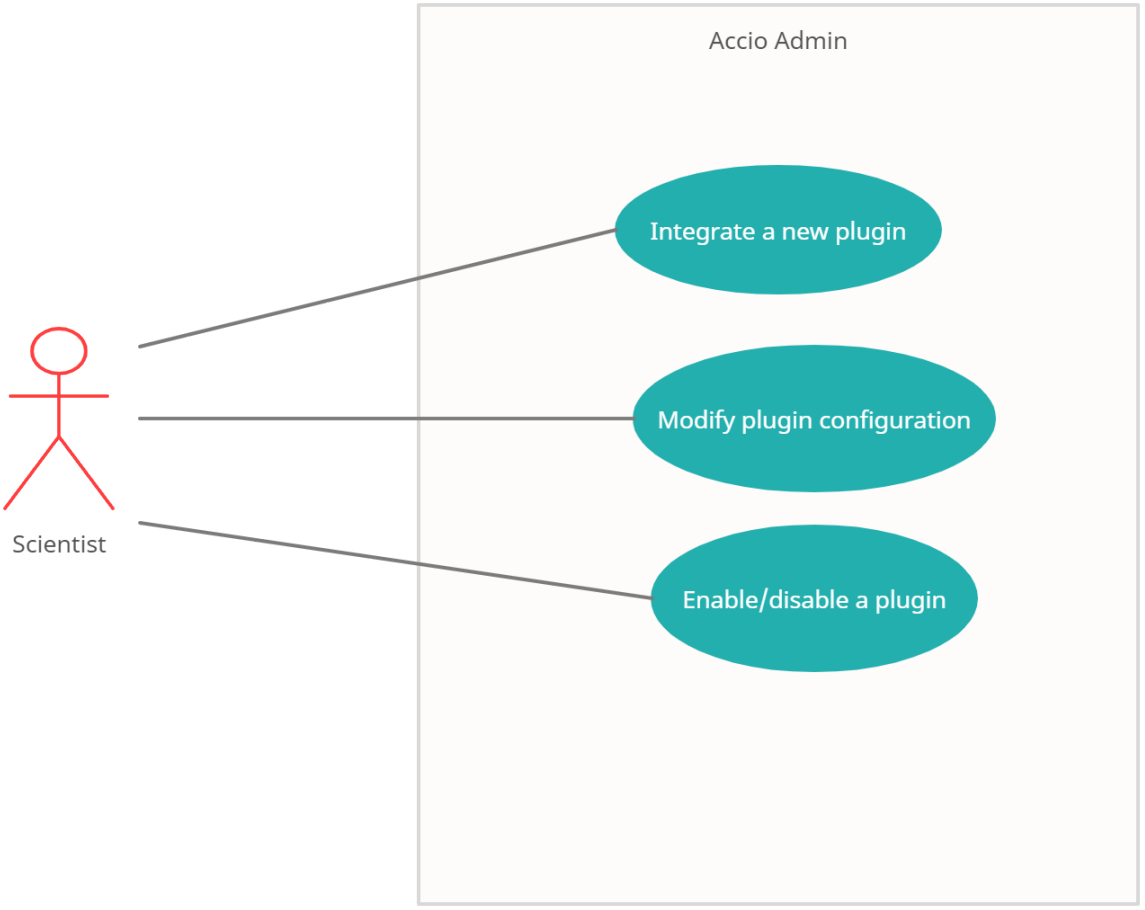


Fig. 3.9

2. End user use case diagram

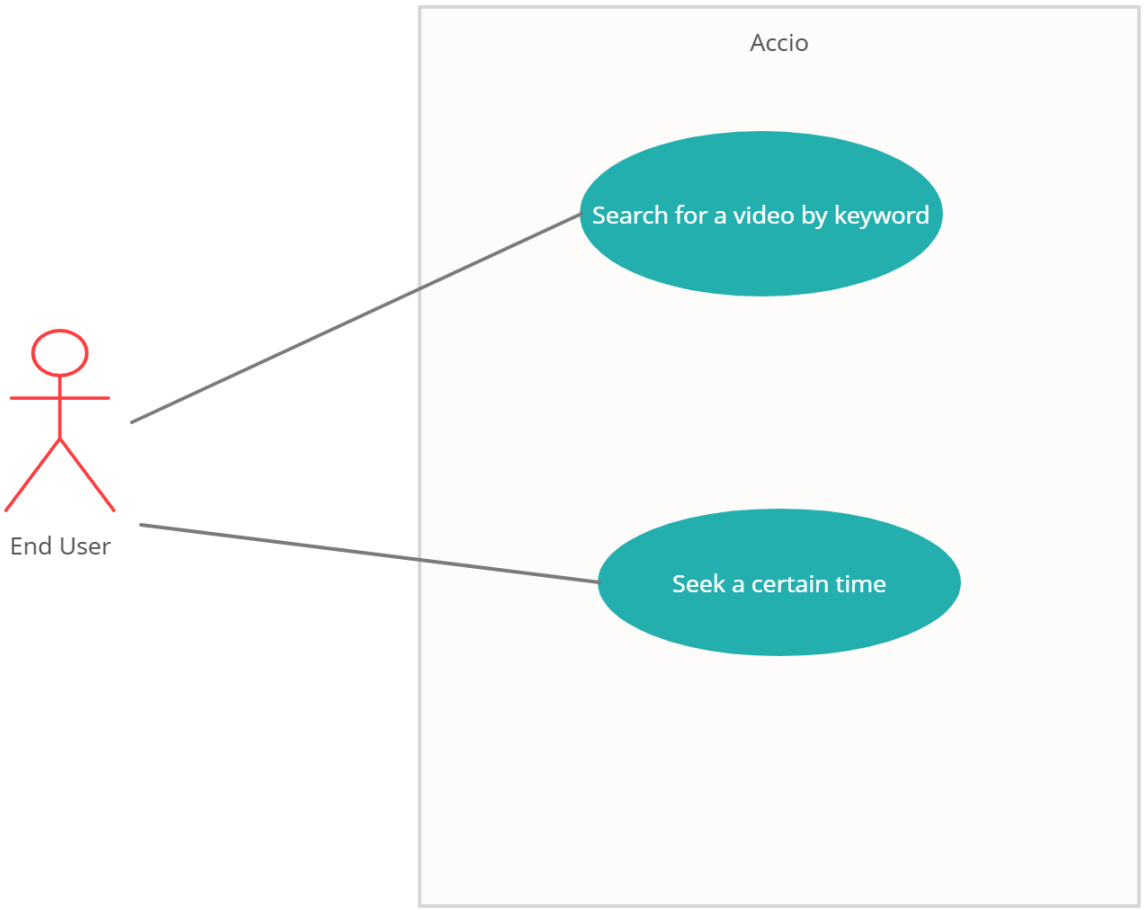


Fig. 3.10

3.2.3 Sequence Diagram

1. Sequence diagram for Enable/disable function.

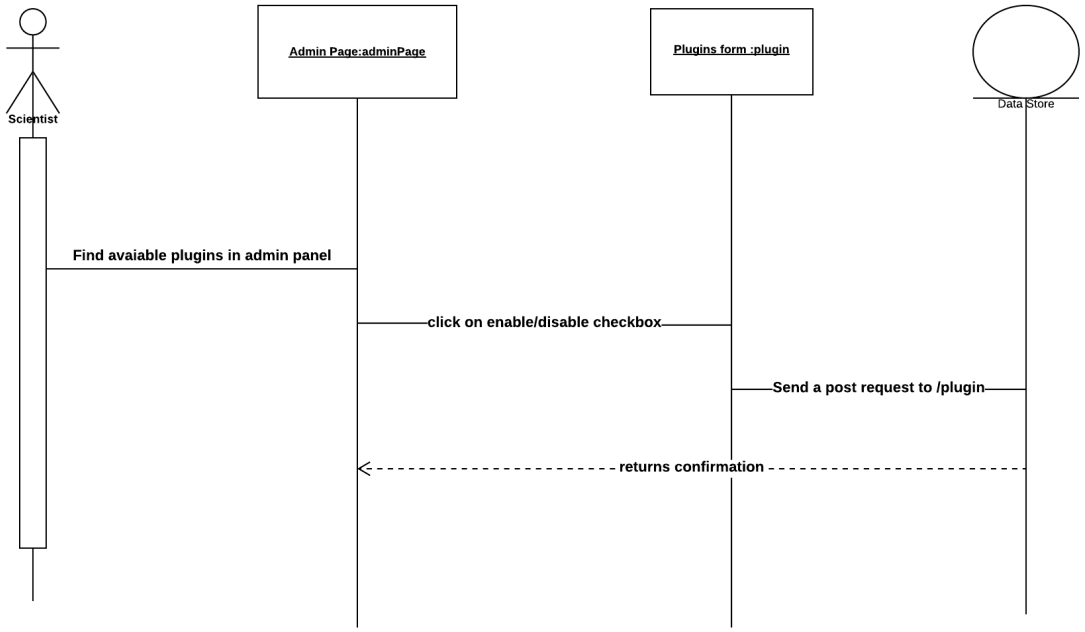


Fig. 3.11

2. Sequence diagram for Add New Plugin functionality.

Fig. 3.12

3. Sequence diagram for modifying a plugin configuration functionality.

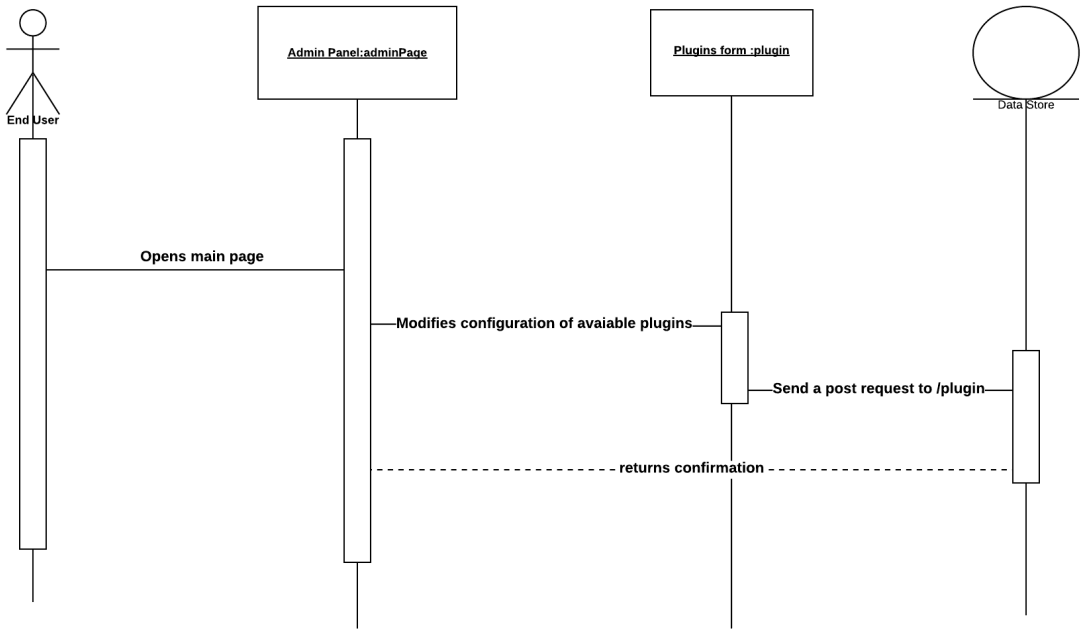


Fig. 3.13

4. Sequence diagram for search video functionality.

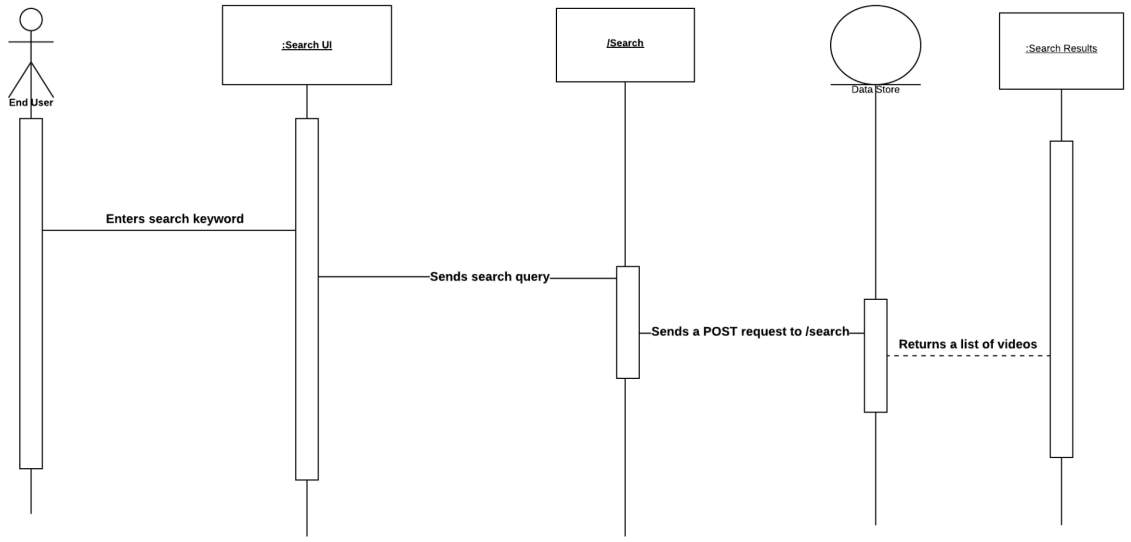


Fig. 3.14

Chapter 4: System Implementation

The system implementation consists of two main subsystems: the frontend and the backend.

The frontend is a web application built using the React framework, React is one of the most powerful web frameworks available and is an industry standard.

The backend is a set of containerized server-side applications backed by a PostgreSQL Database. The applications were built using the Python programming language, due to its ease of use and its wide selection of packages and third-party libraries, and also being the go-to choice for many research papers implementation in the machine learning and computer vision domain.

The main components will be broken down in the next sections.

4.1 Frontend

The frontend is built using React javascript library.

React is an open source javascript library, maintained by Facebook and a community of individual developers and companies.

4.1.1 Pages/routes

1. /
   The main page: a search box is displayed, where the user types his query. It makes a POST request to the /search endpoint.

2. /SearchVideos
   A page that displays the user's search results, where the user can select a single video and will be redirected to the next route.
   It makes a GET request from /GET endpoint.

3. /singlevideo
   A page that displays the selected video in the previous route /SearchVideos

4. /myadmin
   The admin panel page where the scientist/developer can integrate new plugins or enable/disable an existing one.

5. /addnewplugin
   A page for the scientist/developer that contains a form to add a new plugin.
   4 and 5 pages make a POST request to the /plugins endpoint.

4.2 Virtualization

We wanted to adopt the microservices architecture early on during the implementation phase, to be able to experiment with different components in different environments, and to make the system easier to scale later on to adapt to the massive scale of data it needs to handle and the distributed nature of the proposed architecture.

This was mainly done by utilizing virtualization technologies, specifically Docker. Docker is a platform that uses OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries, and configuration files; they can communicate with each other through well-defined channels.

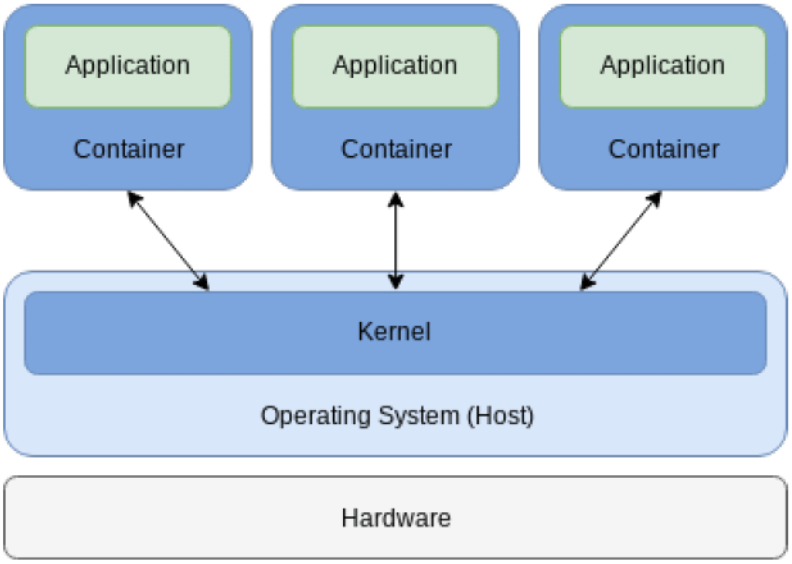


Fig. 4.1: Containerization/virtualization architecture

Utilizing Docker allowed us to horizontally scale the system and perform benchmarks with different configurations to come up with the most suitable one for our local development environment.

4.3 Backend

The backend consists of four docker services running independently.

4.3.1 API

An HTTP server that serves requests coming from the frontend by providing the following endpoints:

1. GET /
   The index of the server, mainly used for health checks.

2. POST /search
   The search endpoint receives search queries and returns a list of matching videos, sorted by relevance.

3. POST /plugins
   An endpoint that allows for modification of the currently integrated plugins.

The API was built using the Flask web framework, and the SQLAlchemy ORM to map SQL tables to traditional Python classes.

4.3.2 Database Server

A PostgreSQL instance running with the default configuration

Videos	
id	int
name	varchar
duration	int
url	varchar
results	JSON
is_processed	boolean

Plugins	
id	int
name	varchar
is_enabled	boolean
executable_path	varchar
system_configuration	JSON
plugin_configuration	JSON

Even though the JSON fields provide flexibility on the database engine level, we had to ensure a level of consistency across all records to be easily managed and used in different parts of the system.

```
[
  {
    "at": 12.266666666666667,
    "results": [
      {
        "content-type": "face_recognition",
        "content": "jim",
        "bb": [176 ,563, 265, 474],
        "confidence": 1.0
      }
    ]
  },
  {
    "at": 2.033333333333333,
    "results": [
      {
        "content-type": "yolo",
        "content": "person",
        "bb": [661, 19, 211, 658],
        "confidence": 0.997
      },
      {
        "content-type": "yolo",
        "content": "car",
        "bb": [836, 175, 219, 75],
        "confidence": 0.99
      },
      {
        "content-type": "yolo",
        "content": "fire hydrant",
        "bb": [332, 200, 45, 50],
        "confidence": 0.532
      }
    ]
  }
]
```

4.3.3 Data Feeder

The data feeder is a daemon service with the main responsibility of exploring new videos and injecting them to be later processed by the processing engine

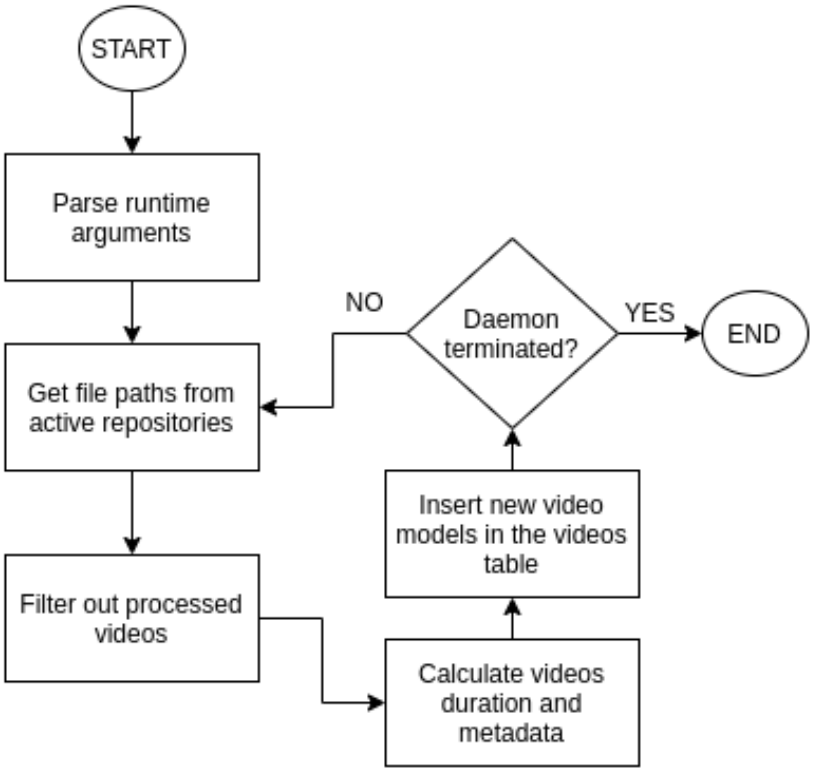


Fig. 4.2: Data Feeder Work Flow

This logic is wrapped in a Python CLI script that supports the following arguments

--repository	List of filesystem directories that feeder is going to monitor and discover new files from
--sleeptime	Wait time between each two consecutive iterations

Sample run:

Fig 4.3

```
(venv) ➜  accio git:(ahmed/apis) ✗ python engine/datafeeder
.py --sleeptime 60 --repository ./repository

[DATA-FEEDER] Watching repository at ./repository
[DATA-FEEDER] DB is already synced with videos repository.
[DATA-FEEDER] 46 seconds remaining.
```

Videos in the DB are same as videos in the repository

Once a video is added to the repository, the feeder detects that change and update the unprocessed datastore as following:

```
(venv) ➜  accio git:(ahmed/apis) ✗ python engine/datafeeder
.py --sleeptime 60 --repository ./repository

[DATA-FEEDER] Watching repository at ./repository
[DATA-FEEDER] DB is already synced with videos repository.
[DATA-FEEDER]  0 seconds remaining.
[DATA-FEEDER] Discovered new videos ['./repository/The Dodo
 - Zoo.mp4'].
[DATA-FEEDER] Added new videos ['./repository/The Dodo - Zo
o.mp4'].
[DATA-FEEDER] 59 seconds remaining.
```

[DATA-FEEDER] Detected new change

4.3.4 Engine

4.3.4.1 Engine Overview

The engine is also a daemon process that is responsible for carrying out the structure analysis, feature extraction, and indexing phases.

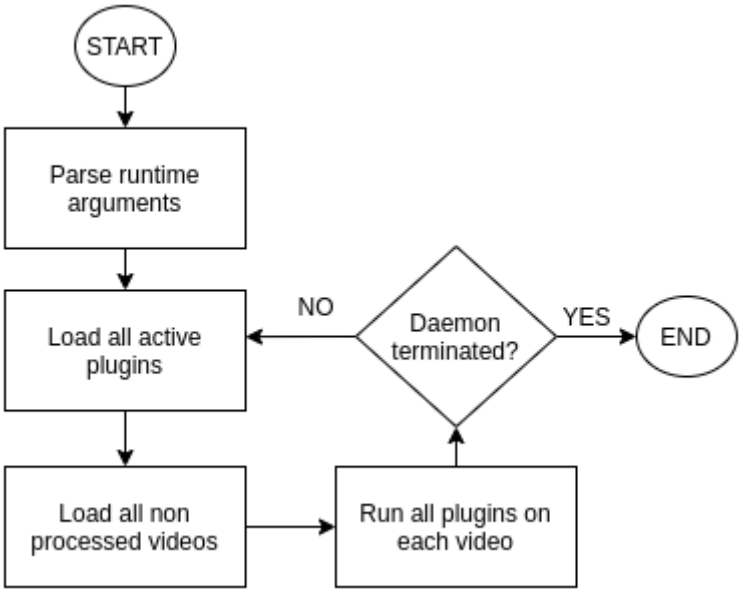


Fig. 4.4: Engine workflow

The main running function of the engine follows this algorithm

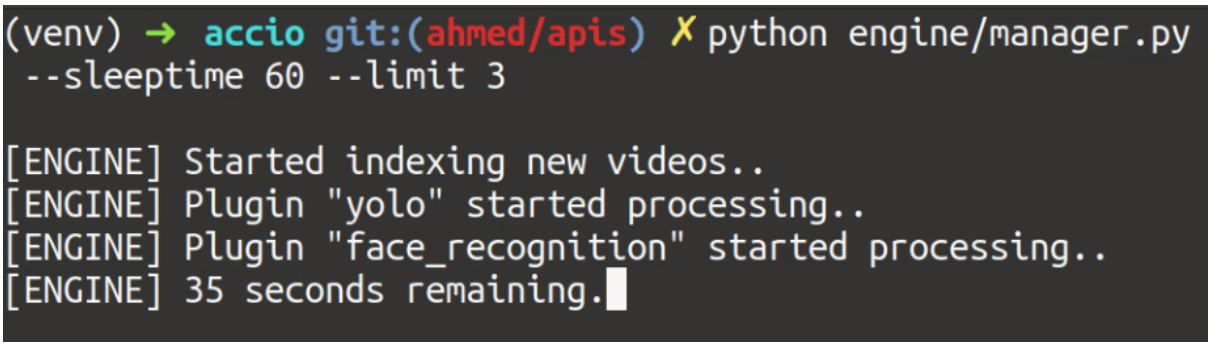
```
def run():
  videos := videos.where(status = NotProcessed)
  videos.each.set(status = Processing)
  plugins := plugins.where(is_enabled = True)
  video_results = {}
  for plugin in plugins:
    for video in videos:
      if plugin performs its own key frame extraction
        video_results[video] += plugin.run(video)
      else
        frames := key_frames_extractor(video)
        video_results[video] += plugin.run(frames)
      end if
    end for
  end for

  videos.each.set(status=Processed)
  videos.each.set(results=video_results[video])
end def
```

This logic is also wrapped in a Python CLI script that accepts the following arguments

--limit	The number of videos to be pulled in one iteration
--sleeptime	Wait time between each two consecutive iterations

Sample run:

Fig. 4.5



Engine is watching internal datastore for new videos

Once the new videos are reported by the Data Feeder, the Engine starts processing them.

```
(venv) ➜ accio git:(ahmed/apis) ✗ python engine/manager.py
 --sleeptime 60 --limit 3

[ENGINE] Started indexing new videos..
[ENGINE] Plugin "yolo" started processing..
[ENGINE] Plugin "face_recognition" started processing..
[ENGINE]  0 seconds remaining.
[ENGINE] Plugin "yolo" started processing..
[ENGINE] yolo: "The Dodo - Zoo.mp4" is being indexed..
[ENGINE] yolo: Extracting keyframes for "The Dodo - Zoo.mp4
"..
```

Running Yolo on the new reported video

4.3.4.2 Keyframe Extractor

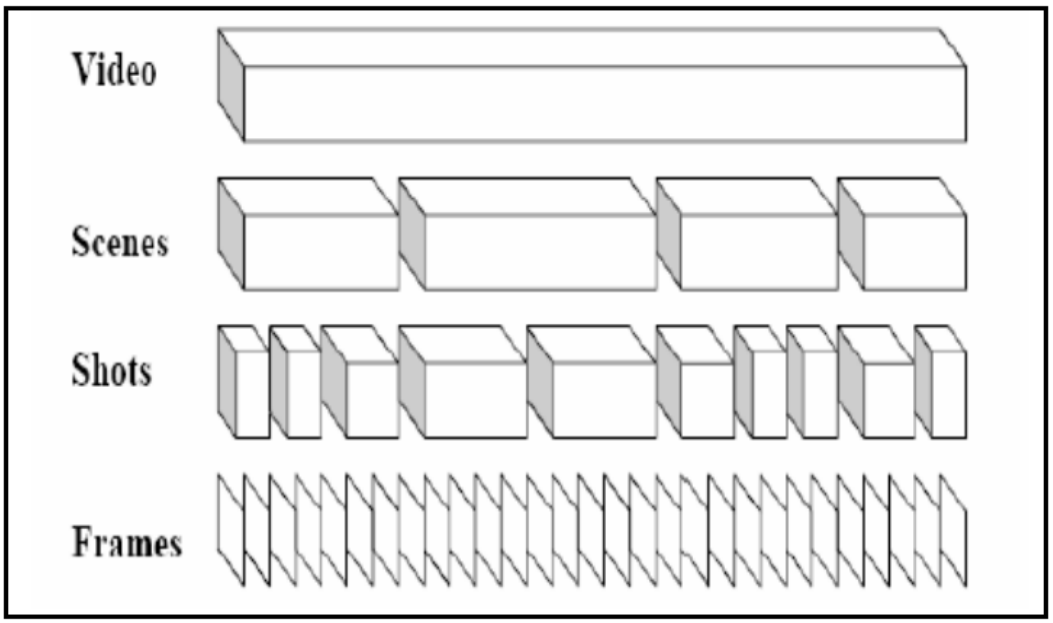


Fig. 4.6: Keyframe Extractor

Keyframes extractor is a submodule inside the processing engine that is responsible for returning the bottom level by splitting the video into scenes then marking the key frame in each scene to be processed.

We implemented a thresholding algorithm to provide flexibility to the advanced user to extract key frames on the rate they need based on their use case.

The procedure calculates

$$HD_k = \frac{1}{N} \sum_{r=0}^{2^B-1} \sum_{g=0}^{2^B-1} \sum_{b=0}^{2^B-1} |p_k(r,g,b) - p_{k-1}(r,g,b)$$

between frames k and k - 1, then compare that value with threshold T as following:

```
def extract_keyframes(video, T):
  keyframes := []
  HD := 0
  previous_frame := NULL
  for frame in video.frames:
    for pixel in frame:
      HD += 1/frame.size * RGB_difference(frame, previous_frame)
    end for
    if HD >= T:
      keyframes.append(frame)
    end if
    previous_frame := frame
  end for
  return keyframes
end def
```

4.4 Plugins & SDK

4.4.1 Plugins overview

One of the most important requirements in our system is modularity, and its adaptability to different use cases and domains.

As a result, the concept of plugins was introduced, where each plugin is a logical component of the system that contributes to the indexed results.

A plugin may represent a single model, or a suite of models and tools working in coherence, or even a fully fledged system in a remote environment.

Plugins are completely isolated from the system, meaning the system can still operate normally in case of absence or failure of any plugin.

All plugins must inherit from the base AbstractPlugin class

```
class AbstractPlugin():
  def __init__(self, plugin_config):
    self.plugin_config = plugin_config
    pass

  def run(self, input_path):
    pass
  # end def
# end class
```

Where each implementation provides the following:

1. **Overridden constructor:** this is where one-time actions need to happen, like loading binaries or configurations from the file system, do any required initializations.
2. **Overridden "Run" method:** this is the method being invoked by the processing engine, it should expect an input_path as a parameter (could either be a frame, shot, or the whole video) as specified in its plugin_configuration

Each plugin is expected to be represented as a record in the Plugins table. Within each record, the plugins need to define two kinds of configurations:

1. **Plugin Configuration:** this can be a place for hyperparameters, dynamic variables that can be changed at runtime and accessed by the "Run" method.
2. **System Configuration:** this can be used to override the default values of some variables defined by the processing engine, like setting a different threshold for the keyframes extraction, or even disabling it altogether.

4.4.2 SDK

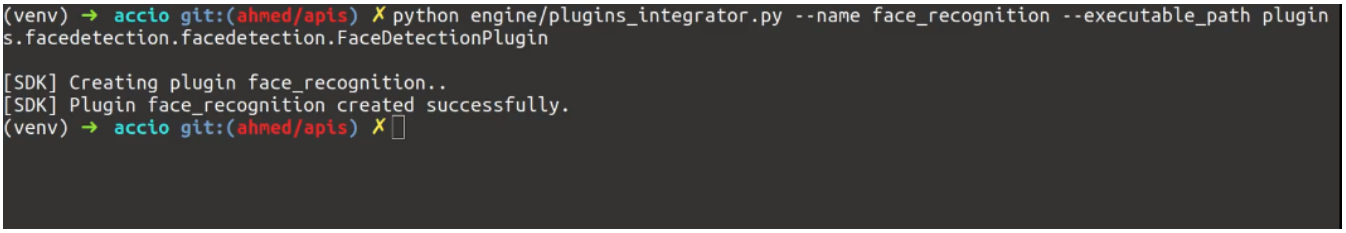
Writing a class implementation is a burden, let alone having to keep track of all the configuration and database operations.

Therefore, we saw the need to automate this process, which is what the SDK does. The SDK is another Python CLI application that eases the process of integrating plugins into the system.

It expects the following arguments

--name	Name of the plugin
--executable_path	The path of the plugin class implementation
--plugin_config	The plugin configuration
--system_config	The plugin's system configuration (uses the system's defaults if absent)

Sample Run:



4.5 Demonstrated Plugins

We implemented two plugins to showcase the system and gain some insights on what needs to be improved in the integration layer.

4.5.1 Object Detection Plugin

This is a plugin class implementation based on a YOLO-V3 model trained on 80 classes of the COCO dataset.

```
def init():
  load weights and yolo parameters from the plugin configuration
  net := initialize a darknet network object using the weights and
config
end def
```

As the weights are loaded during runtime, this would let us easily load different weights of custom trained classes in addition to the default 80 classes.

```
def run(input_path):
  frame = read(input_path)
  resize(frame)

  ln := output layer of the YOLO NN

  blob := generate_blob from frame
  perform a forward pass of the blob of the YOLO detector
  set net input = blob
  layer_outputs = get network outputs
  results := []
  for output in layer_outputs:
    for detection in output:
      // threshold on the value specified in the configuration
      if detection.confidence >= plugin_config['threshold']:
        results += detection
      end if
    end for
  end for
  // apply non-maxima suppression on overlapping bounding boxes
  NMS(results)
  // decode classes into labels
  results := decode_classes(results)
  return results
end def
```

4.5.2 Face Detection Plugin

This is a plugin class implementation of a HOG-based multi-face recognition model, pre-trained to detect 4 characters from “The Office” Series.

Fig. 4.7: The Office Characters

```
def init():
  encodings := load serialized face encodings
end def
```

Notice that the encodings are also loaded during runtime, this would let us train the model on more faces and update the configuration during runtime without any down time.

```
def run(input_path):

  frame = read(input_path)
  resize(frame)

  generate encodings using HOGDescriptor for the current frame
  results := []

  // calculate scores for each class
  for face in encodings:
    classes_scores = {}
    for class in encodings:
      Score := similarity score between the class and the face
      classes_scores[class] = score
    end for

    // get the class with the max confidence
    dominant := max(classes_scores)
    if dominant.score >= plugin.config['confidence_threshold']:
      results += dominant
    end if
  end for

  return results
end def
```

Chapter 5: Experiments

In this section, we demonstrate some trials and experiments we applied on the system in a local development environment.

5.1 Video Segmentation/Keyframes Threshold

Segmentation threshold is one of the tuning parameters that advanced users need to set based on their use cases. If the use case is to recognize actions, then the model expects a large number of frames in one second (high threshold).
If the use case is to recognize faces, it would be much better to set the threshold with smaller value to get the keyframes with faces only without duplication.

The following graph is a comparison between different values of threshold applied of the same video (a 2 mins scene from the office series, with 3600 frames)

Threshold value	Number of keyframes
T=0.3	1680 frames
T=0.4	757 frames
T=0.5	342 frames
T=0.55	102 frames
T=0.65	81 frames
T=0.8	45 frames
T=0.9	19 frames
T=1	2 frames

If all frames have exactly the same pixels in the same positions, the number of frames will be 1 regardless of the value of the threshold, which means the number of extracted frames is video dependent.

5.2 Scalability Test

In out initial design, the system was monolith application running sequentially, the old design was as following:

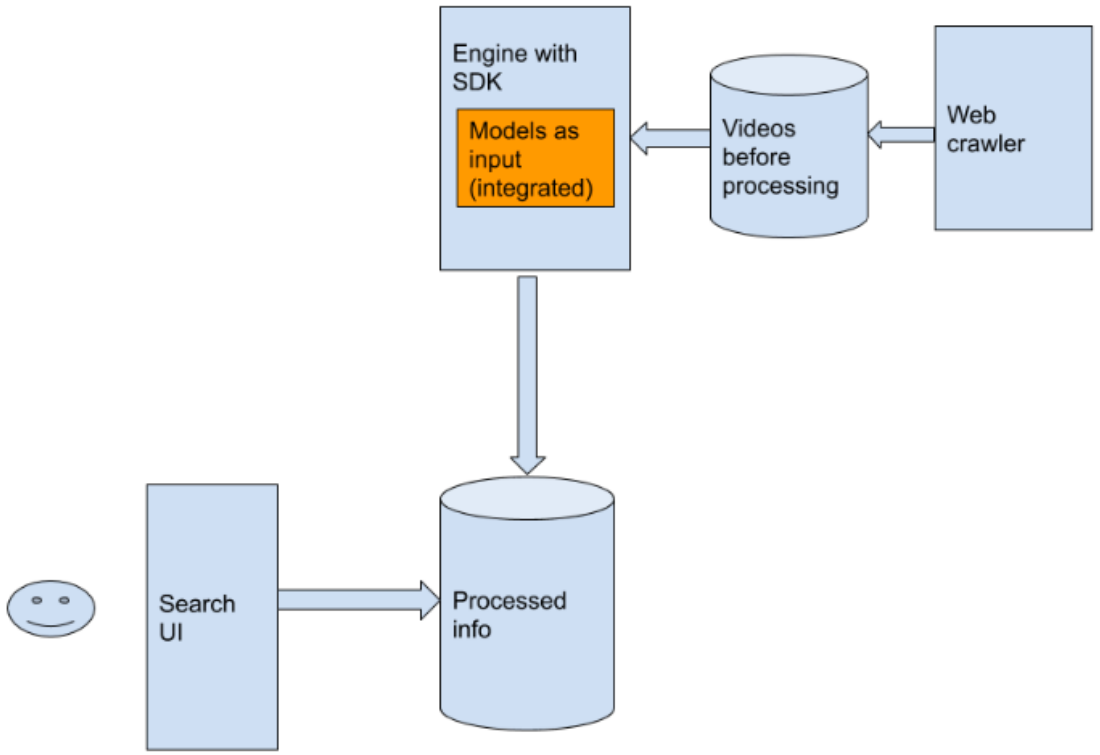


Fig. 5.1: Initial architecture

We redesigned the architecture, re-implemented the components to meet the current architecture as explained in chapter 3 and chapter 4.
The new design enables us to create multiple instances from each component based on the nature of the datasets. The following is a comparison between old and new design with different number of processes/instances in terms of number of minutes taken to process a small dataset of 10 videos of length 3000 seconds.

Number of instances	Time (rounded to nearest minute)
1 instance (old design)	29 minutes
2 instances (new design)	16 minutes
5 instances (new design)	7 minutes

Chapter 6: System Testing

6.1 System Installation

6.1.1 Backend (Accio)

This section illustrates the installation steps for the backend application.

6.1.1.1 System Dependencies (Python Packages)

```
alembic==1.6.2
backcall==0.2.0
click==8.0.0
decorator==5.0.9
dlib==19.22.0
face-recognition==1.3.0
face-recognition-models==0.3.0
Flask==2.0.0
Flask-Cors==3.0.10
Flask-Migrate==3.0.0
Flask-SQLAlchemy==2.5.1
greenlet==1.1.0
imutils==0.5.4
ipdb==0.13.7
ipython==7.23.1
ipython-genutils==0.2.0
itsdangerous==2.0.1
jedi==0.18.0
Jinja2==3.0.1
Mako==1.1.4
MarkupSafe==2.0.1
matplotlib-inline==0.1.2
numpy==1.20.3
opencv-contrib-python==4.5.2.52
parso==0.8.2
PeakUtils==1.3.3
pexpect==4.8.0
pickleshare==0.7.5
Pillow==8.2.0
prompt-toolkit==3.0.18
psycopg2-binary==2.8.6
ptyprocess==0.7.0
Pygments==2.9.0
python-dateutil==2.8.1
python-dotenv==0.17.1
python-editor==1.0.4
scipy==1.6.3
six==1.16.0
SQLAlchemy==1.4.15
toml==0.10.2
traitlets==5.0.5
wcwidth==0.2.5
Werkzeug==2.0.1
```

Fig. 6.1: Python Packages in the requirements file

6.1.1.2 Significant Packages

face-recognition

Recognize and manipulate faces from Python or from the command line with the world's simplest face recognition library. Built using **dlib's** state-of-the-art face recognition built with deep learning. The model has an accuracy of 99.38% on the Labeled Faces in the Wild benchmark. This also provides a simple face_recognition command line tool that lets you do face recognition on a folder of images from the command line.

Flask

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks.

Flask-SQLAlchemy

Flask-SQLAlchemy is an extension for Flask that adds support for SQLAlchemy to your application. It aims to simplify using SQLAlchemy with Flask by providing useful defaults and extra helpers that make it easier to accomplish common tasks.

SQLAlchemy

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

Pickle

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. "Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.

6.1.1.3 Install Packages

Use the command line interface to create new virtual environment and activate it

```
$ virtualenv venv
$ source venv/bin/activate
```

Install the requirements from requirements.txt

```
$ pip install -r requirements.txt
```

Export Flask configuration

```
$ source .env
# OR
$ export FLASK_APP=./api/main.py
$ export FLASK_ENV=development
```

Install Postgres or setup Docker:

- Postgres Installation

  ```
  $ sudo apt-get install postgresql-12

  # Setup admin user
  $ sudo -u postgres psql postgres
  $ CREATE DATABASE accio;
  $ ALTER USER postgres PASSWORD 'postgres';
  ```

- Docker Setup

```
# Install requirements
$ sudo apt-get install \
apt-transport-https \
ca-certificates \
curl \
gnupg \
lsb-release


# Add Docker’s official GPG key
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg

$ echo \ "deb [arch=amd64
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu \ $(lsb_release -cs)
stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null

# Install Docker Engine
$ sudo apt-get install docker-ce docker-ce-cli containerd.io

# Install Docker Compose
$ sudo curl -L
"https://github.com/docker/compose/releases/download/1.25.4/d
ocker-compose-$(uname -s)-$(uname -m)" -o
/usr/local/bin/docker-compose sudo

$ chmod +x /usr/local/bin/docker-compose

# Run docker file
$ docker-compose up -d

# Stabilize connection between Flask and Docker
$ docker-compose exec db bash

# Login to DB
$ psql accio -U postgres
```

Setup Flask databases

```
$ flask db migrate
$ flask db upgrade
```

After performing all previous steps, run Flask server

```
$ flask run
```

6.1.2 Frontend (Accio-frontend)

This section illustrates the installation steps for the frontend application.

6.1.2.1 System Dependencies (Package.json)

```
{
    "@material-ui/core": "^4.11.4",
    "@testing-library/jest-dom": "^5.11.4",
    "@testing-library/react": "^11.1.0",
    "@testing-library/user-event": "^12.1.10",
    "axios": "^0.21.1",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-player": "^2.9.0",
    "react-router-dom": "^5.2.0",
    "react-scripts": "4.0.3",
    "web-vitals": "^1.0.1"
}
```

6.1.1.2 Significant Packages

Axios

axios is a very popular JavaScript library you can use to perform HTTP requests, that works in both Browser and Node. js platforms. It supports all modern browsers, including support for IE8 and higher. It is promise-based, and this lets us write async/await code to perform XHR requests very easily.

Material-ui

Material-ui is a design language developed by Google in 2014. Expanding on the "cards" that debuted in Google Now, Material Design uses more grid-based layouts, responsive animations and transitions, padding, and depth effects such as lighting and shadows.

React-router

React Router is a standard library for routing in React. It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.

Reactor-roter-dom

A tool that allows you to handle routes in a web app, using dynamic routing. Dynamic routing takes place as the app is rendering on your machine, unlike the old routing architecture where the routing is handled in a configuration outside of a running app.

6.1.2.3 Install Packages

Download and install Node from https://nodejs.org/en/download/

Use the command line to install the dependencies by typing the following command:

```
$ npm install -g yarn
$ yarn install
```

You can also use npm instead of yarn

```
$ npm install -g npm
$ npm install
```

6.1.2.3 Run Scripts

```
{
   "start": "react-scripts start",
   "build": "react-scripts build",
   "test": "react-scripts test",
   "eject": "react-scripts eject"
}
```

To start the development server, run the first script.

```
$ npm start
# OR
$ yarn start
```

The server works on localhost:3000

6.2 User Manual

The system has two types of users; the **end user** and the **scientist/developer.**

1. The **end user** enters the keyword that they want to search for.

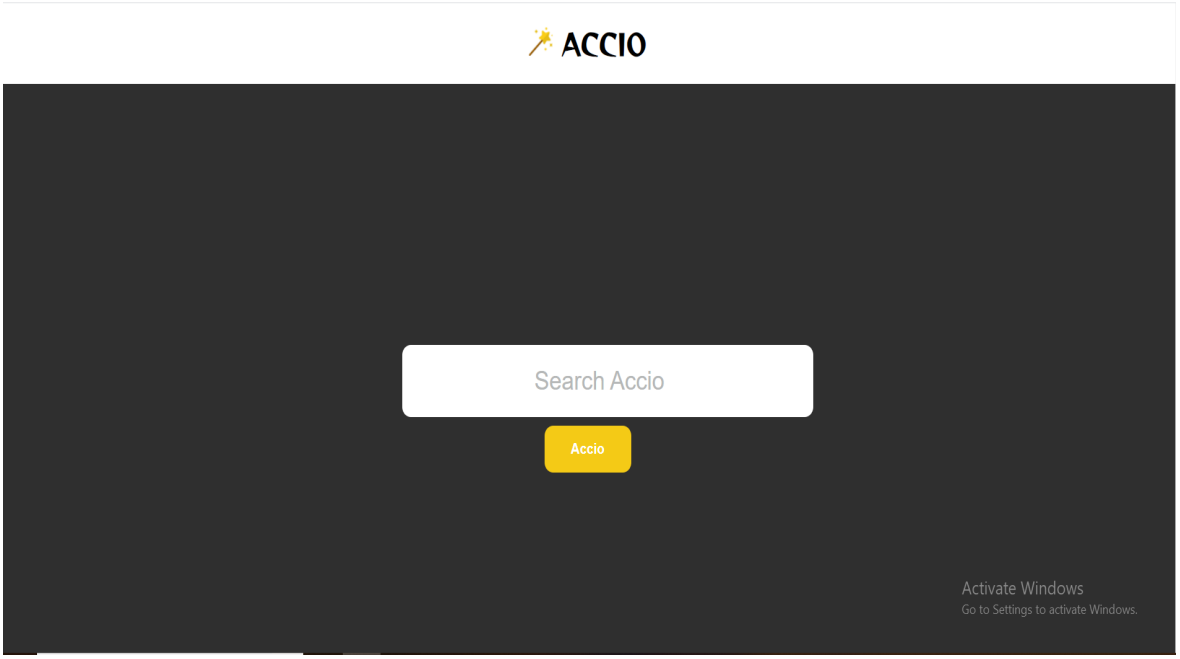


Fig. 6.2 Welcome page

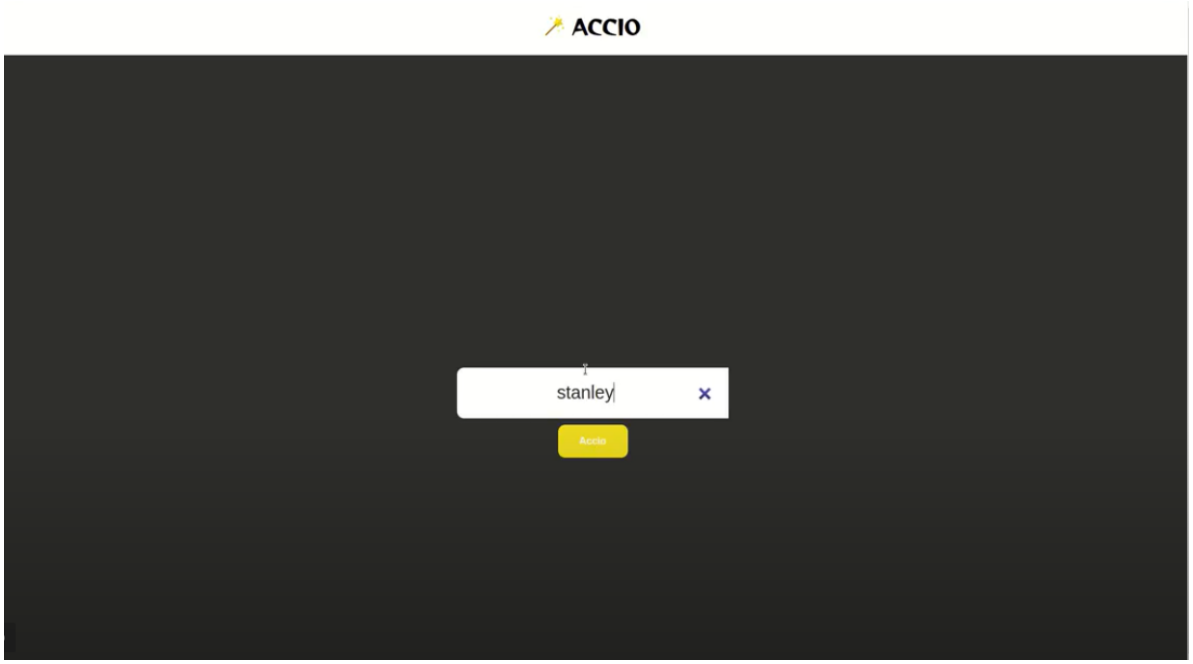


Fig. 6.3 User enters a search keyword

2. The user will be directed to another page where search results will be displayed.

Fig. 6.4 Search results
All videos that contain the search keyword.

3. The user can click on a certain video and they will be redirected to another page where all the seconds at which his search appears are displayed.

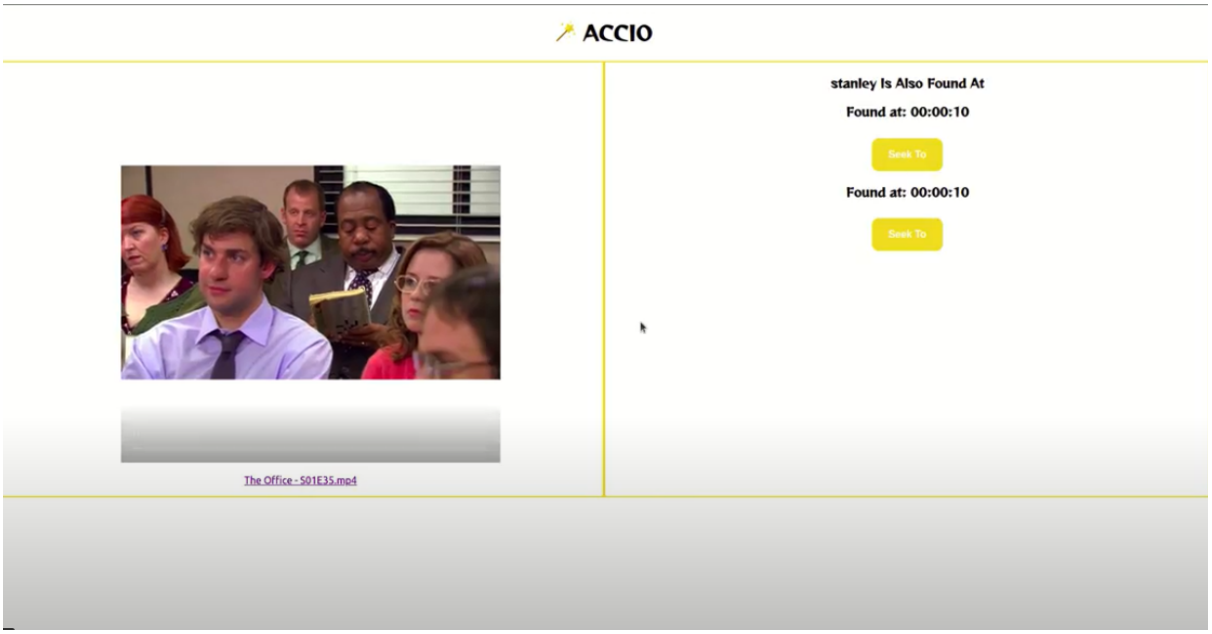


Fig. 6.5 Users can seek a certain time in the video.

4. The **scientist/developer** integrates the system with new plugins, models or programs.

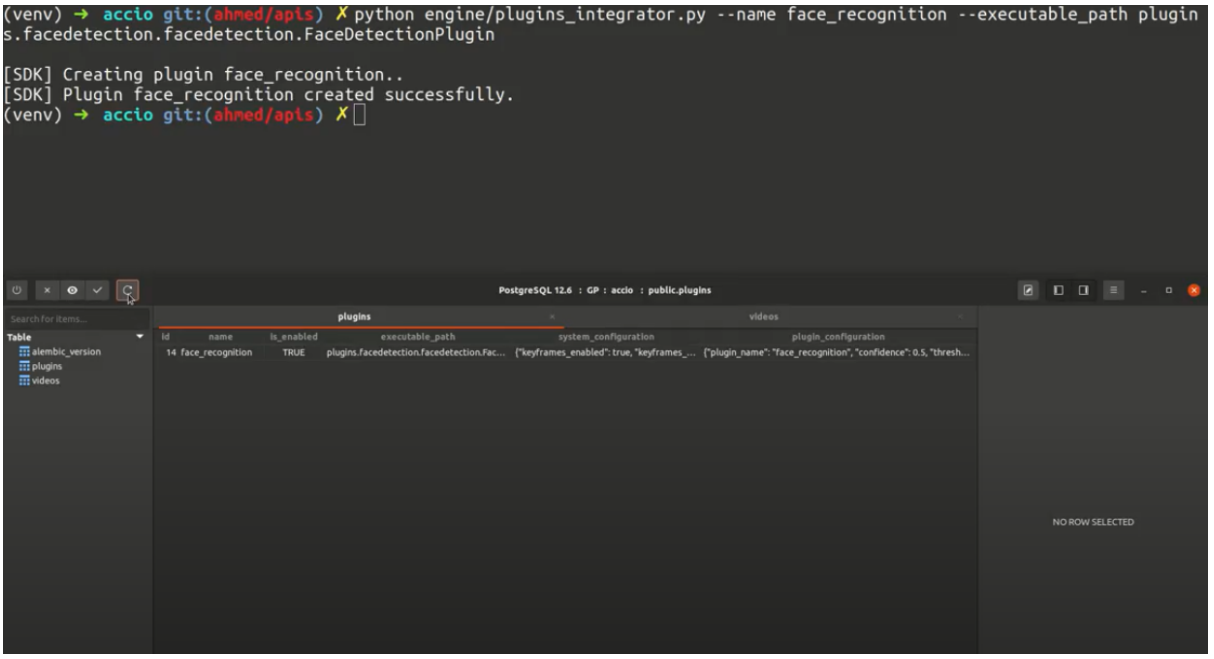


Fig. 6.6 Example of integrating facial recognition model with the system.

5. The scientist/developer has an admin panel where they can enter a new plugin or enable/disable an available one.

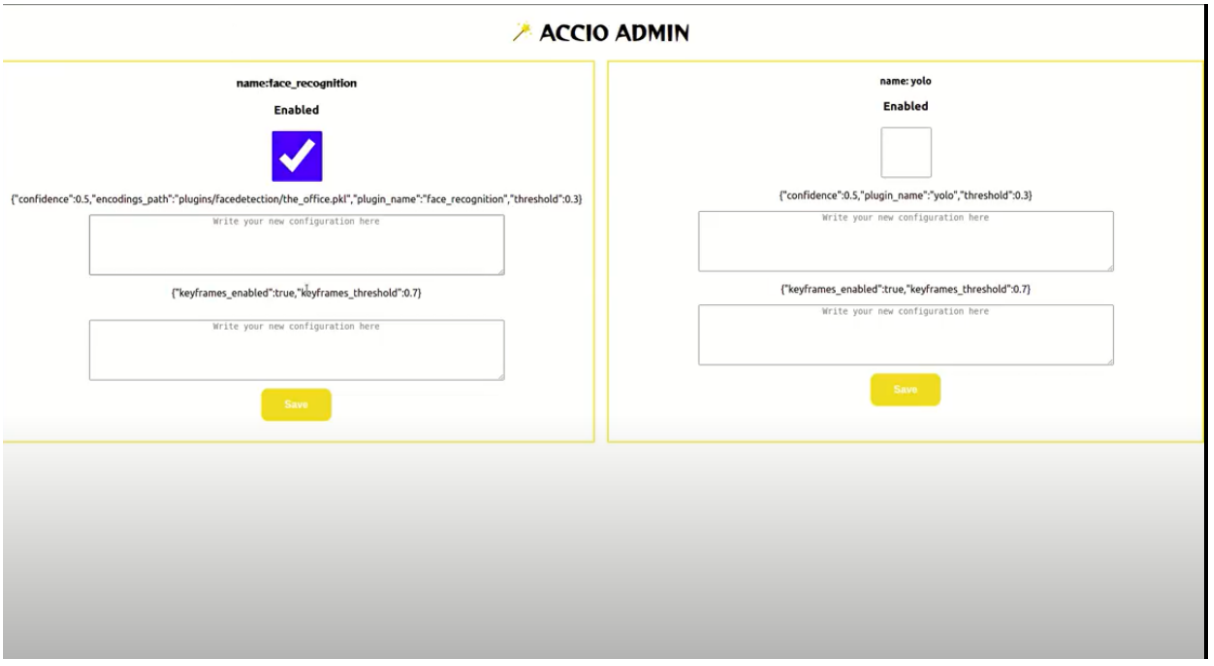


Fig. 6.7 Scientist enables face recognition plugin in admin panel.

Chapter 7: Conclusion and Future Work

7.1 Conclusion
7.2 Future work

7.1 Conclusion

The project aim was to prove the modularity concept we used in the project and the possibility to add different modules that work independently across different servers or instances in the server-side backend to better enhance performance and increase processing speed and efficiency which was proved using the proposed architecture in the project and further ease the optimization and modularity for each module on its own as a standalone module aside from the whole project.

The overall performance compared from using a monolithic architecture vs a microservices architecture was extremely different and demonstrated the power we leveraged from the microservices architecture.

The project also demonstrates the great power of distributed computing and how it can be useful for huge files computations

7.2 Future work

1. Add audio detection models to leverage the audio channels in the video files to make further more accurate predictions and make a wide range of search options added to the project current perspective and the model will be added as a plugin in a way similar to the current plugins.
2. Leverage cloud computing technologies to further improve the computation performance to reach an acceptable limit when used with huge video datasets under high load pressure.
3. Stress testing each module in the project hence every module can be operated separately to find and eliminate possible bottlenecks to further increase performance.
4. Use different models and provide them to the system as plugins to enhance the modularity options by updating and increasing (Developer/Scientist) control over the system modules to achieve a better modular structure.

References

[1] Spolaôr, N., Lee, H., Takaki, W., Ensina, L., Coy, C., & Wu, F. (2020, February 18). A systematic review on content-based video retrieval.

[2] Shivanand Sharanappa Gornale, Ashvini K Babaleshwar & K Babaleshwar (2019, March). Analysis and Detection of Content based Video Retrieval.

[3] Mühling, M., Meister, M., Korfhage, N., Wehling, J., Hörth, A., Ewerth, R., & Freisleben, B. (2016, September 05). Content-Based Video Retrieval in Historical Collections of the German Broadcasting Archive.

[4] Selvaganesan, J., & Kannan, N. (2016). Unsupervised feature based key-frame extraction towards face recognition. Int. Arab J. Inf. Technol., 13, 777-783.

[5] Luong, T., Pham, N., & Vu, Q. (2016). Vietnamese Multimedia Agricultural Information Retrieval System as an Info Service.

[6] Mohammed Aasif Ansari, & Muzammil H Mohammed. (2015, March). Content based Video Retrieval Systems -Methods, Techniques, Trends and Challenges. Researchgate.

[7] Souček, T., Moravec, J., & Lokoč, J. (2019). TransNet: A deep network for fast detection of common shot transitions. arXiv preprint arXiv:1906.03363.

[8] Mühling, M., Korfhage, N., Müller, E., Otto, C., Springstein, M., Langelage, T., ... & Freisleben, B. (2017). Deep learning for content-based video retrieval in film and television production. Multimedia Tools and Applications, 76(21), 22169-22194.

[9] Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2117-2125).

[10] Xu, J., Song, L., & Xie, R. (2016, November). Shot boundary detection using convolutional neural networks. In 2016 Visual Communications and Image Processing (VCIP) (pp. 1-4). IEEE.

[11] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[12] Patel, B. V., & Meshram, B. B. (2012). Content based video retrieval systems. arXiv preprint arXiv:1205.1641.

[13] Dalal, N., & Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05) (Vol. 1, pp. 886-893). Ieee.

[14] Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).

[15] Tong, W., Song, L., Yang, X., Qu, H., & Xie, R. (2015, June). CNN-based shot boundary detection and video annotation. In 2015 IEEE international symposium on broadband multimedia systems and broadcasting (pp. 1-5). IEEE.

[16] M. (2020, May 30). YOLO — You Only Look Once - Towards Data Science. Medium. https://towardsdatascience.com/yolo-you-only-look-once-3dbdbb608ec4

[17] Introduction to YOLO Algorithm for Object Detection. (2021, April 15). Engineering Education (EngEd) Program | Section. https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/

[18] Almog, U. (2020, October 13). YOLO V3 Explained - Towards Data Science. Medium. https://towardsdatascience.com/yolo-v3-explained-ff5b850390f