GUN DETECTION FOR SECURITY SURVEILLANCE

Submitted in partial fulfilment of the
Requirements for the award of the degree

of
Bachelor of Technology
in
Computer Science & Engineering

by:

Anurag Bali (005/CSE3/2017)
Gurpreet Singh (015/CSE3/2017)
Gurpreet Singh Sahni (016/CSE3/2017)
Simrat Kaur (043/CSE3/2017)

Under the guidance of

Dr. Aashish Bhardwaj

Department of Computer Science & Engineering
Guru Tegh Bahadur Institute of Technology

Guru Gobind Singh Indraprastha University
Dwarka, New Delhi
Year 2017-2021

DECLARATION

We hereby declare that all the work presented in the dissertation entitled **"Gun Detection for Security Surveillance"** in the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in **Computer Science & Engineering,** Guru Tegh Bahadur Institute of Technology, affiliated to Guru Gobind Singh Indraprastha University Delhi is an authentic record of our own work carried out under the guidance of **Dr. Aashish Bhardwaj**

Date:

Anurag Bali (005/CSE3/2017)
Gurpreet Singh (015/CSE3/2017)
Gurpreet Singh Sahni (016/CSE3/2017)
Simrat Kaur (043/CSE3/2017)

CERTIFICATE

This is to certify that dissertation entitled **"Gun Detection for Security Surveillance",** which is submitted by- Anurag Bali, Gurpreet Singh, Gurpreet Singh Sahni and Simrat Kaur in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in **Computer Science & Engineering**, Guru Tegh Bahadur Institute of Technology, New Delhi is an authentic record of the candidate's own work carried out by them under our guidance. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

Dr. Aashish Bhardwaj

(Project Guide & H.O.D.)

ACKNOWLEDGEMENT

We would like to express our great gratitude towards our supervisor, **Dr. Aashish Bhardwaj**, who has given us support and suggestions. Without her help we could not have presented this dissertation up to the present standard. We also take this opportunity to give thanks to all others who gave us support for the project or in other aspects of our study at our college- Guru Tegh Bahadur Institute of Technology.

Date:

Anurag Bali (005/CSE3/2017)
Gurpreet Singh (015/CSE3/2017)
Gurpreet Singh Sahni (016/CSE3/2017)
Simrat Kaur (043/CSE3/2017)

ABSTRACT

Gun violence is an urgent, complex, and multifaceted problem. It requires evidence-based, multifaceted solutions. Gun violence should be considered a public health issue, not a political one—an epidemic that needs to be addressed with research and evidence-based strategies that can reduce morbidity and mortality. Gun violence affects people of all ages and races.

On average, police response time is around 18 minutes. When a shooting occur, sometimes calling police can't be done right away, instead, a person should be in a safe place for him to be able to call for help. and that might take so much more time. So, the 18 minutes police response time does not include the time it takes to call the police.

Gun Detector is a computer-aided process of detecting gun from image or video. The task of detecting gun based on Convolutional Neural Networks which is trained to detect this type of object. The weapon detection task can be performed by different approaches of combining a region proposal technique with a classifier, or integrating both into one model. However, any deep learning model requires to learn a quality image dataset and an annotation according to the classification or detection tasks.

We are proposing a way to reduce the police time greatly by using real-time weapon detection that could be implemented on any cctv camera. The gun detection model is made so it could find a weapon in a picture, video.

Once a weapon is detected in a video, police could be alerted. They can see the live feed and verify that someone has a gun before dispatching to the incident location. This method does not require anyone to actually call the police, moreover, this will cut the response time and may prevent a shooting from happening. This project aimed to minimize this response time by implementing a weapon detection. This project can be generalized to work on many other industries and locations. For example, public parks, gas stations, banks, etc.

LIST OF FIGURES

INDEX

Topic Page No.

1. INTRODUCTION

1.1 STATEMENT ABOUT THE PROBLEM

Over the past eight years, the number of school mass shootings has increased drastically. Gun violence on school grounds resulted in many wounded or dead school kids which mirrors the gun violence problem in the United States. The failure to address the root cause of school gun violence is having lasting consequences for millions of American children. As we are waiting for our leaders to find a solution for gun violence as a whole and on school grounds specifically, we need to find an alternative way to minimize the number of incidents or prevent it from occurring in the first place. On average, police response time is around 18 minutes. When a shooting occurs, sometimes calling police can't be done right away, instead, a person should be in a safe place for him to be able to call for help. and that might take so much more time. So, the 18 minutes police response time does not include the time it takes to call police.

1.2 OBJECTIVES OF THE STUDY

In this project, we are proposing a way to reduce the police time greatly by using real-time weapon detection that could be implemented on any cctv camera. The object detection model is made so it could find a weapon in a picture, video or most importantly, a live video feed.Once a weapon is detected in a live video, police could be alerted. They can see the live feed and verify that someone has a gun before dispatching to the incident location. This method does not require anyone to actually call the police, moreover, this will cut the response time and may prevent a shooting from happening.

1.3 BENEFITS OF GUN DETECTION

Gun violence headlines tend to populate our search engines and social media. Constantly developing news stories involving unresolved violent crimes are deteriorating police-community relations. Gunshot Detection Technology (GDT) addresses this problem by helping authorities respond to the scene in a shorter time span, prevent crime and find strategies to solve problems in insecure neighbourhoods.

1.3.1 RAPID RESPONSE

A witness call might be a great way to encourage a faster dispatch and prevent crime, but some people might still be reluctant to call the officers or going through a lot of panic thus not providing the right demographics. Providing the police with accurate information during a gunfire situation is key to a rapid response.GDT was designed to identify, authenticate and notify officers automatically within seconds of a gunshot detection. The information provided includes specific times and position of gunfire. Several acoustic sensors collect information and triangulate it to pinpoint the source of the sound, date and time. The audio is then transferred to law enforcement or the vendor (depending on the specific GDT product) and from this point the data can be verified as a gunshot or discarded as a false alarm.Some vendors provide technicians specifically trained to provide more detailed information about the incident, like what type of firearm was used and whether it came from a moving vehicle. With this technology, the rate of response can be enhanced allowing officials to not only save lives but in other cases they are able to recover forensic evidence for investigation.

1.3.2 PROBLEM SOLVING

When integrated with video surveillance, GDT allows researchers to review video feeds of street block attributes, run onsite observations and conduct interviews. Gunshot detection reports provided by citizens, demographics and physical features of a neighbourhood can help identify hot spots. This information can then be used to review the effectiveness of an official's response to crime scenes and analyze how these problems can be acknowledged and resolved.

1.3.3 CRIME PREVENTION

GDT provide information that crime analysts use to spot trends, locate concentrations of gun violence and produce maps to help authorities make tactical decisions about where to deploy patrols and other resources. This analysis can also encourage social workers to recruit community members who want to participate in conducting a street outreach to address conflicts between social groups, crews and gangs before they escalate. Though it is critical for residents to continue reporting gun fire incidents, GDT is an effective tool to help solve firearm violence by generating a comprehensive report resulting in faster response and reduced crime rates.

1.4 FEATURES OF GUN DETECTION

In this study, the state-of-the-art YOLO V3 object detection model was implemented and trained over our collected dataset for weapon detection. We propose a model that provides a visionary sense to a machine or robot to identify the unsafe weapon and can also alert the human administrator when a gun or a firearm is obvious in the edge. The experimental results show that the trained YOLO V3 has better performance compared to the YOLO V2 model and is less expensive computationally. There is an immediate need to update the current surveillance capabilities with improved resources to support monitoring the effectiveness of human operators. Smart surveillance systems would fully replace current infrastructure with the growing availability of low-cost storage, video infrastructure, and better video processing technologies. Eventually, the digital monitoring systems in terms of robots would fully replace current surveillance systems with the growing availability of cheap computing, video infrastructure, high-end technology, and better video processing.

2. REQUIREMENTS

2.1 SOFTWARE REQUIREMENT SPECIFICATION

A Software Requirement Specification (SRS) is a description of a software system to be developed. It lays out functional requirements and may include a set of use cases that describe user interactions that the software must provide.

Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers (in market-driven projects, these roles may be played by the marketing and development divisions) on what the software product is to do as well as what it is not expected to do. Software requirements specification permits a rigorous assessment of requirements before design can begin and reduces later redesign.

It should also provide a realistic basis for estimating product costs, risks and schedules. Used appropriately, software requirements specifications can help prevent software project failure

The software requirements specification document enlists enough and necessary requirements that are required for the project development. To derive the requirements, the developer needs to have a clear and thorough understanding of the products to be developed or being developed. This is achieved and refined with detailed and continuous communications with the project team and customer till the completion of the software.

The SRS may be one of a contact deliverable Data Item Description or have other forms of organizationally-mandated content.

2.2 USE OF SRS

The purpose of the document is to collect and analyse all assorted ideas that have come up to define the system, the requirements with respect to consumers. Also, we shall predict and sort out how we hope this product will be used in order to gain a better understanding of the project, outline concepts that may be developed later, and document ideas that are being considered, but may be discarded as the product developers.

In short, the purpose of this SRS document is to provide a detailed overview of our software product, its parameters and goals. This document describes the project's target audience and it's user interface, hardware and software requirements. It defines how our client, team and audience see the product and its functionality. Nonetheless, it helps any designer and developer to assist in software delivery lifecycle (SDLC) processes.

2.3 DATASETS REQUIREMENT

Datasets of the images and videos are taken from visual database. The ImageNet project is a large visual database designed for use in visual object recognition software research.

More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided.

ImageNet contains more than 20,000 categories with a typical category, such as "balloon" or "strawberry", consisting of several hundred images

2.4 SOFTWARE REQUIREMENTS

On Developer Side:

- Operating System : Windows 10 or later/macOS/Linux (64-bit)
- Languages : Python, Dockerfile scripting
- Libraries : Twilio, OpenCV, Yolov3, Streamlit
- Dependencies : Docker, Docker-compose, AWS
- IDE : Visual Studio Code
- Commands : Docker, Docker-compose
- Browser : Chrome/Mozilla/Safari/IE 9

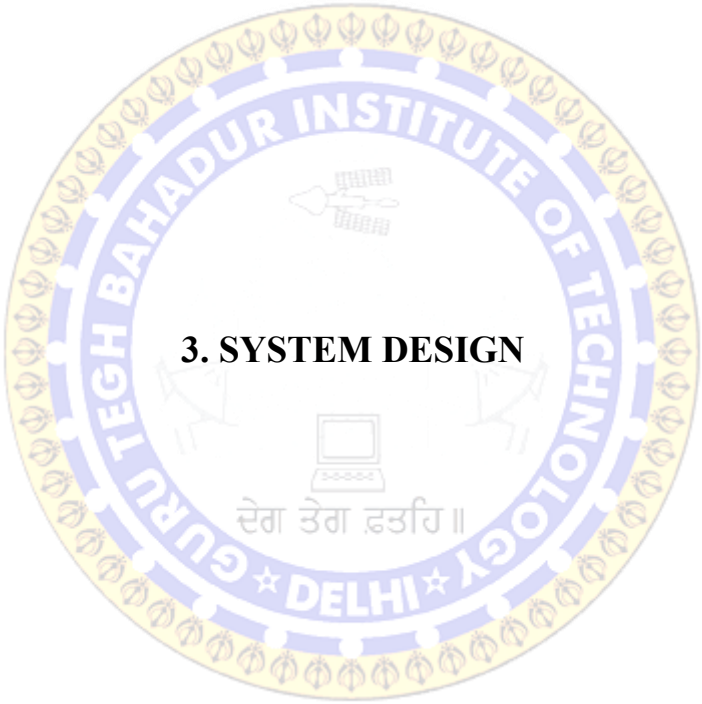
On Client Side:

- Operating System : Windows 7 or later/macOS/Linux (64-bit)
- Dependencies : None

- Browser : Chrome/Mozilla/Safari/IE9

2.5 HARDWARE REQUIREMENTS

- Processor : Dual core or above
- RAM : 4 GB or above
- Hard disk : 10GB or above
- Monitor : 13 inch LCD Montior / Mobile phone
- Keyboard : Standard keyboard
- AWS EC2 instance
- RAM : 2GB or above



3. SYSTEM DESIGN

3.1 Entity Relationship Diagram

An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties.

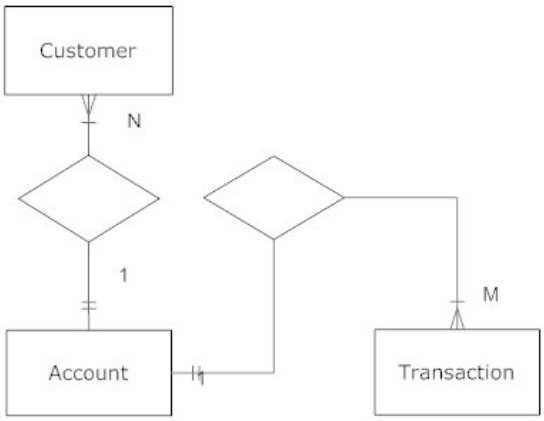
By defining the entities, their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases.

ER diagrams are used to sketch out the design of a database.

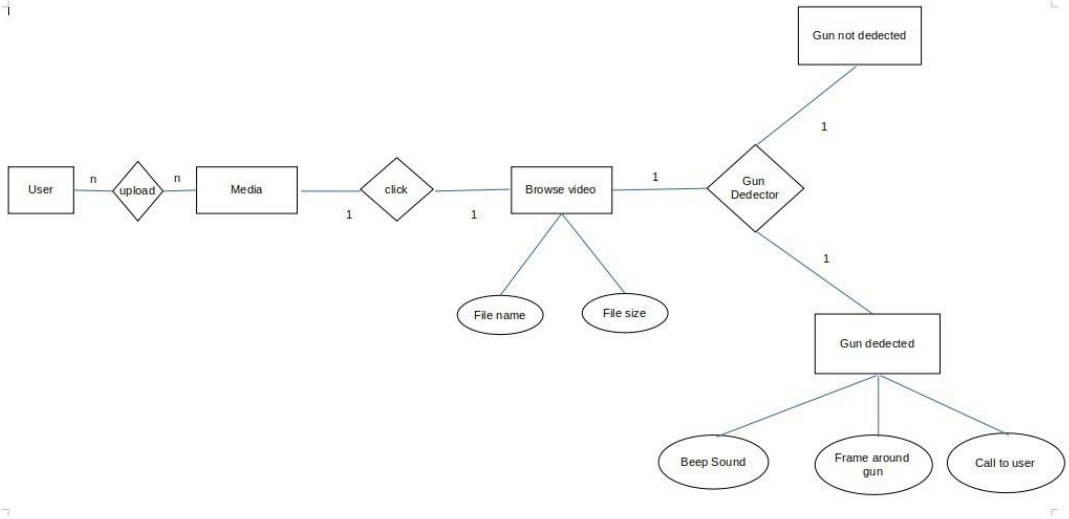
Common Entity Relationship Diagram Symbols

An ER diagram is a means of visualizing how the information a system produces is related. There are five main components of an ERD:

1. **Entities**, which are represented by rectangles. An entity is an object or concept about which you want to store information.
- **Actions**, which are represented by diamond shapes, show how two entities share information in the database.
- **Attributes**, which are represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity.
- **Connecting lines**, solid lines that connect attributes to show the relationships of entities in the diagram.
- **Cardinality** specifies the maximum number of relationships and **Ordinality** specifies the absolute minimum number of relationships.



3.2 ER Diagram of The Colourizer



Entities and Attributes

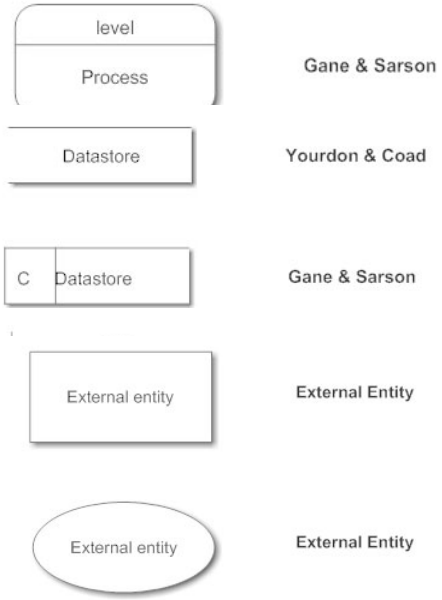
- User - The collection of all the users.
- Media - The collection of all photos and videos of the user that will get uploaded.
- Browse image / video – Browsing image and video from user's local system and upload it on application. They have File name and File size as attributes.
- Coloured image/video - Black & white photo or video will get colourize and output will be visible.

3.3 Data Flow Diagrams

A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on the flow of information, where data comes from, where it goes and how it gets stored. It maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled.

Data Flow Diagrams Symbols

There are essentially two different types of notations for data flow diagrams (Yourdon & Coad or Gane & Sarson) defining different visual representations for processes, data stores, data flow and external entities.



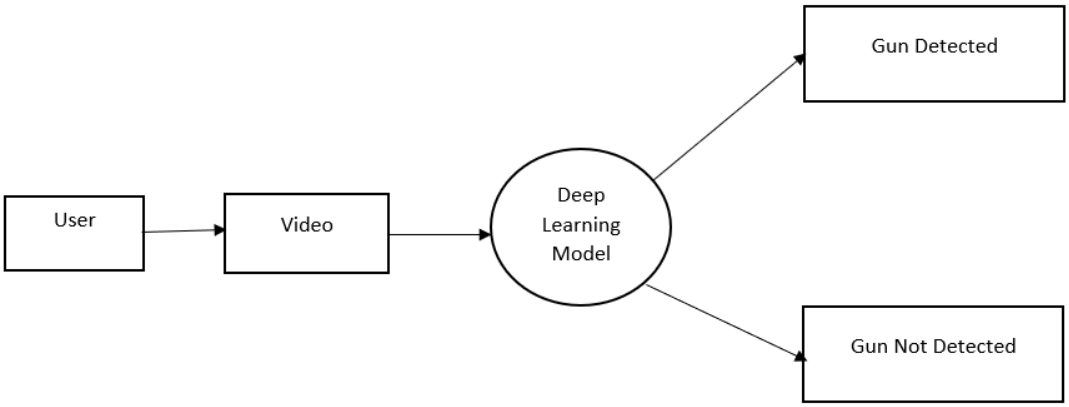
Process Notations: A process transforms incoming data flow into outgoing data flow.

Datastore Notations: Datastores are repositories of data in the system. They are sometimes also referred to as files.

Data flow Notations: Dataflows are pipelines through which packets of information flow. Label the arrows with the name of the data that moves through it.

External Entity Notations: External entities are objects outside the system, with which the system communicates. External entities are sources and destinations of the system's inputs and outputs.

3.4 DFD Level 0 (The Colourizer)



3.5 Use Case Diagram

The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and Statechart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

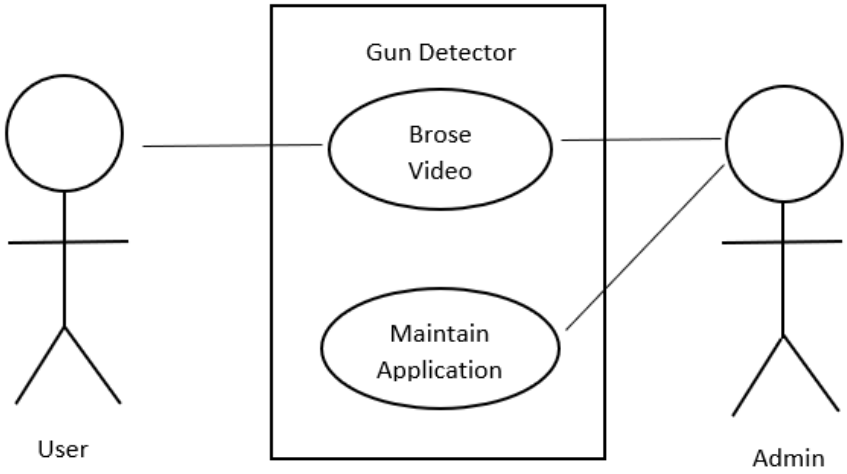
Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.

When the initial task is complete, use case diagrams are modelled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows −

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Show the interaction among the requirements are actors.

Use Case Diagram of The Colourizer



4. LANGUAGES AND TECHNOLOGY USED

In this project, python language is used to write the whole code. We have used different python libraries such as Keras, Twilio, OpenCV, Streamlit, Yolov3.

4.1 Python

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side)
- software development,
- mathematics,
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python an connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

4.2 Keras

Keras is an open-source library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3 Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, R, Theano, and PlaidML. As of version 2.4, only TensorFlow is supported. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research

effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

Features

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU).

Keras back ends

Keras proper does not do its own low-level operations, such as tensor products and convolutions; it relies on a back-end engine for that. Even though Keras supports multiple back-end engines, its primary (and default) back end is TensorFlow, and its primary supporter is Google. The Keras API comes packaged in TensorFlow as tf.keras, which as mentioned earlier will become the primary TensorFlow API as of TensorFlow 2.0.

To change back ends, simply edit your $HOME/.keras/keras.json file and specify a different back-end name, such as theano or CNTK. Alternatively, you can override the configured back end by defining the environment variable KERAS_BACKEND, either in your shell or in your Python code using the os.environ["KERAS_BACKEND"] property.

Keras models

The Model is the core Keras data structure. There are two main types of models available in Keras: the Sequential model, and the Model class used with the functional API.

Keras Sequential models

The Sequential model is a linear stack of layers, and the layers can be described very simply. Here is an example from the Keras documentation that uses model.add() to define two dense layers in a Sequential model:

```
import keras
from keras.models import Sequential
from keras.layers import Dense
#Create Sequential model with Dense layers, using the add method
model = Sequential()
#Dense implements the operation:
#  output = activation(dot(input, kernel) + bias)
#Units are the dimensionality of the output space for the layer,
# which equals the number of hidden units
#Activation and loss functions may be specified by strings or classes
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))
#The compile method configures the model's learning process
model.compile(loss='categorical_crossentropy',
    optimizer='sgd',
    metrics=['accuracy'])
#The fit method does the training in batches
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn API.
model.fit(x_train, y_train, epochs=5, batch_size=32)
#The evaluate method calculates the losses and metrics
#     for the trained model
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
#The predict method applies the trained model to inputs
#     to generate outputs
classes = model.predict(x_test, batch_size=128)
```

The comments in the code above are worth reading. It's also worth noting how little cruft there is in the actual code compared to, say, the low-level TensorFlow APIs. Each layer definition requires one line of code, the compilation (learning process definition) takes one line of code, and fitting (training), evaluating (calculating the losses and metrics), and predicting outputs from the trained model each take one line of code.

Keras functional API

The Keras Sequential model is simple but limited in model topology. The Keras functional API is useful for creating complex models, such as multi-input/multi-output models, directed acyclic graphs (DAGs), and models with shared layers.

The functional API uses the same layers as the Sequential model but provides more flexibility in putting them together. In the functional API you define the layers first, and then create the Model, compile it, and fit (train) it. Evaluation and prediction are essentially the same as in a Sequential model, so have been omitted in the sample code below.

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))
# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
 metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

Keras layers

In the previous examples we only used Dense layers. Keras has a wide selection of predefined layer types, and also supports writing your own layers.

Core layers include Dense (dot product plus bias), Activation (transfer function or neuron shape), Dropout (randomly set a fraction of input units to 0 at each training update to avoid overfitting), Lambda (wrap an arbitrary expression as a Layer object), and several others. Convolution layers (the use of a filter to create a feature map) run from 1D to 3D and include the most common variants, such as cropping and transposed convolution layers for each dimensionality. 2D convolution, which was inspired by the functionality of the visual cortex, is commonly used for image recognition.

Pooling (downscaling) layers run from 1D to 3D and include the most common variants, such as max and average pooling. Locally connected layers act like convolution layers, except that the weights are unshared. Recurrent layers include simple (fully connected recurrence), gated, LSTM, and others; these are useful for language processing, among other applications. Noise layers help to avoid overfitting.

Keras datasets

Keras supplies seven of the common deep learning sample datasets via the keras.datasets class. That includes cifar10 and cifar100 small color images, IMDB movie reviews, Reuters newswire topics, MNIST handwritten digits, MNIST fashion images, and Boston housing prices.

Keras applications and examples

Keras also supplies ten well-known models, called Keras Applications, pretrained against ImageNet: Xception, VGG16, VGG19, ResNet50, InceptionV3, InceptionResNetV2, MobileNet, DenseNet, NASNet, MobileNetV2TK. You can use these to predict the classification of images, extract features from them, and fine-tune the models on a different set of classes.

By the way, fine-tuning existing models is a good way to speed up training. For example, you can add layers as you wish, freeze the base layers to train the new layers, then unfreeze some of the base layers to fine-tune the training. You can freeze a layer with by setting layer.trainable = False.

The Keras examples repository contains more than 40 sample models. They cover vision models, text and sequences, and generative models.

Deploying Keras

Keras models can be deployed across a vast range of platforms, perhaps more than any other deep learning framework. That includes iOS, via CoreML (supported by Apple); Android, via the TensorFlow Android runtime; in a browser, via Keras.js and WebDNN; on Google Cloud, via TensorFlow-Serving; in a Python webapp back end; on the JVM, via DL4J model import; and on Raspberry Pi.

To get started with Keras, read the documentation, check out the code repository, install TensorFlow (or another backend engine) and Keras, and try out the Getting Started tutorial for the Keras Sequential model. From there you can advance to other tutorials, and eventually explore the Keras Examples.

4.3 Twilio

Twilio is an American cloud communications platform as a service (CPaaS) company based in San Francisco, California. Twilio allows software developers to programmatically make and receive phone calls, send and receive text messages, and perform other communication functions using its web service APIs.

Twilio provides a simple entry point into the telephony world, and helps your business avoid *many* of the traditional complexities. Developers can quickly get worldwide connectivity by interacting with Twilio using common internet protocols and simple markup. This bridge between the internet and the telephone network is just the basic foundation of Twilio's product offerings – but it illustrates the power of what Twilio can do.

Developers can build software on the internet using technology they already know. Interacting with Twilio's platform is much simpler than interacting with the expensive, complex, and costly world of telecommunications.

Twilio Use Cases in the Wild

- Airbnb uses Twilio to allow people to call one another about their rooms, without revealing either person's phone number!
- Intuit uses it Twilio to send users confirmation codes via text message
- John Keefe at WNYC used Twilio to allow Brooklyn bus riders to find out how soon there bus would be at their stop
- Turbo Vote let users set up an text message reminder to go out and vote!
- GroupMe used the Twilio API to build their group messaging service!

4.4 Streamlit

Streamlit is an open source app framework specifically designed for ML engineers working with Python. It allows you to create a stunning looking application with only a few lines of code.

I want to take this opportunity to demonstrate the apps you can build using Streamlit. But mostly, to show you the steps necessary to bring your application into production using Heroku.

A few of the advantages of using Streamlit tools like Dash and Flask:

- It embraces Python scripting; No HTML knowledge is needed!
- Less code is needed to create a beautiful application
- No callbacks are needed since widgets are treated as variables
- Data caching simplifies and speeds up computation pipelines.

Installation

Streamlit can easily be installed with the following command:

pip install streamlit

Use the following command to see a demonstration of an application with example code:

streamlit hello

Doing this will result in the following page to be opened:

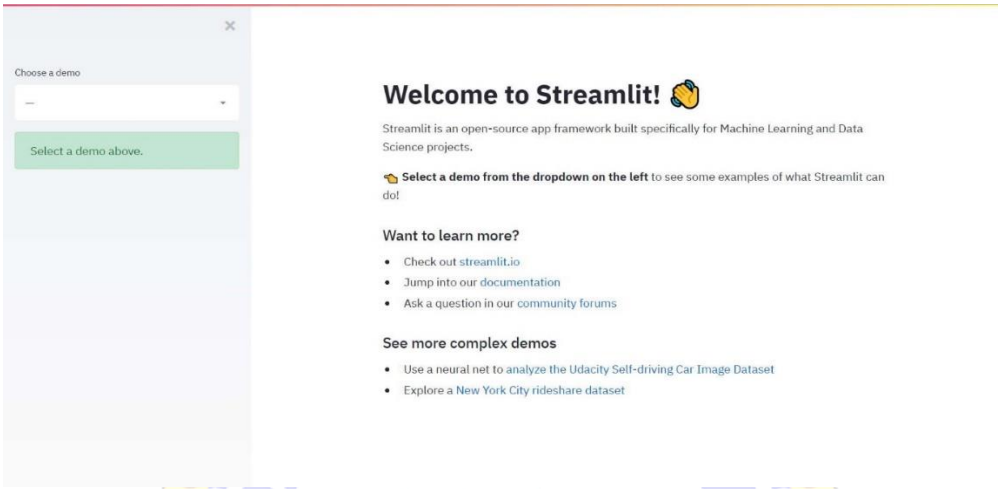


Fig 1: Streamlit

Demos

Above from the streamlit hello demo above there are a few more complex demos available online for you to try out. I will list a few (including mine) below such that you will have an idea of the possibilities:

The Udacity Self-driving Car Image Browser

This demo shows how the Udacity car dataset can be combined with object detection in less than 300 lines of code to create a complete Streamlit Demo app.

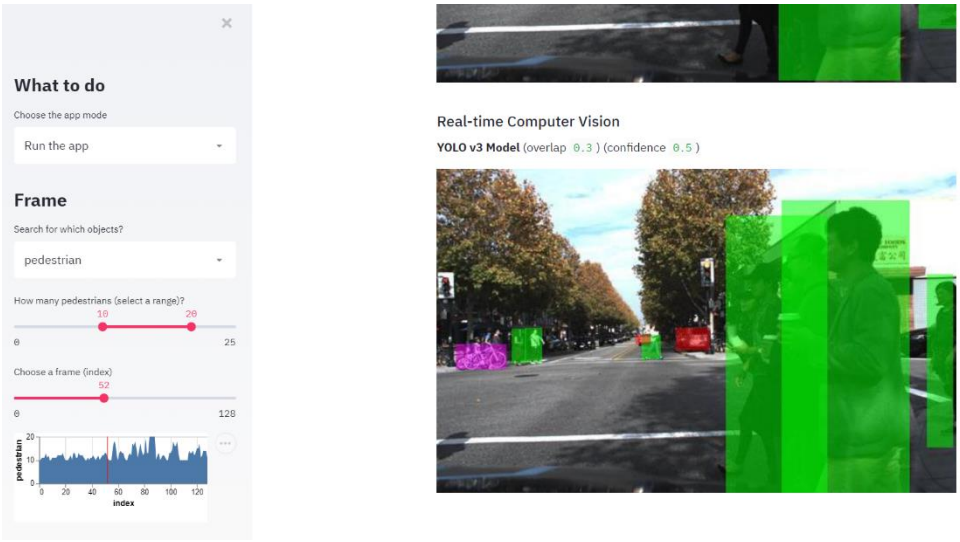


Fig 2: Streamlit

First, install OpenCV so that the images can be analyzed:

pip install --upgrade streamlit opencv-python

Next, simply run the app:

streamlit run https://raw.githubusercontent.com/streamlit/demo-self-driving/master/app.py

Uber Pickups in New York City

This is a Streamlit demo to show how you can interactively visualize Uber pickups in New York City.

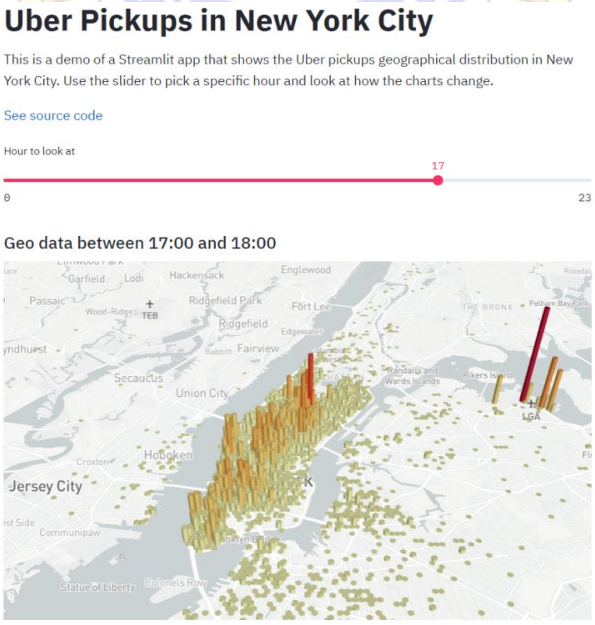


Fig 3: Streamlit

Simply run the code below after installing Streamlit:

streamlit run https://raw.githubusercontent.com/streamlit/demo-uber-nyc-pickups/master/app.py

Board Game Exploration

As many Board Game Geeks like myself track the scores of board game matches I decided to create an application allowing for the exploration of this data. You can view this application live here, or you can run it locally by following the steps below.

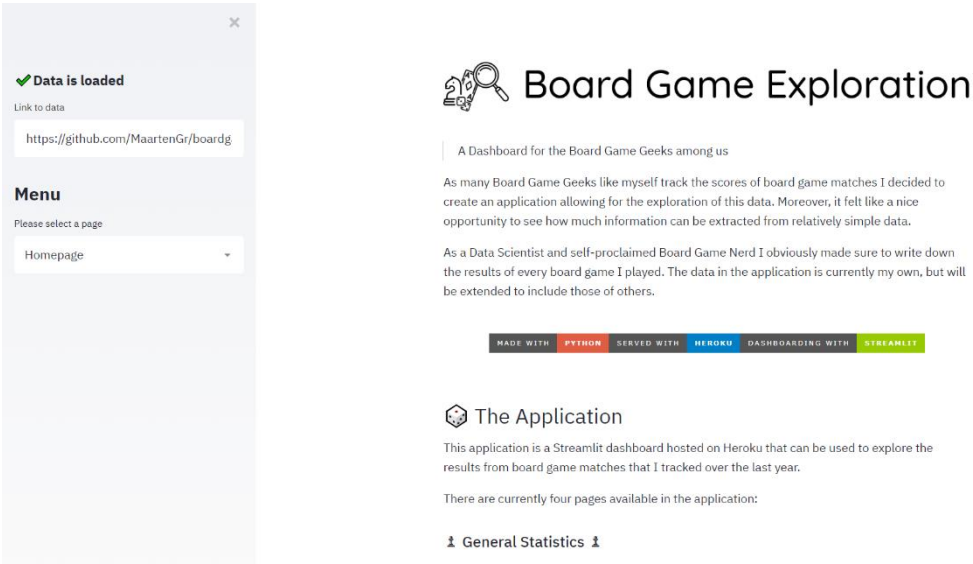


Fig 4: Streamlit

Since I make use of several.py files, you first need to clone the repository:

git clone https://github.com/MaartenGr/boardgame.git BoardGame

Then, simply go to the folder and run Streamlit:

cd BoardGame
streamlit run app.py

NOTE: You can use your own data if you like by simply providing the URL to your data.

4.5 OpenCV

OpenCV is a cross-platform library using which we can develop real-time computer vision applications. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.

Let's start the chapter by defining the term "Computer Vision".

Computer Vision

Computer Vision can be defined as a discipline that explains how to reconstruct, interrupt, and understand a 3D scene from its 2D images, in terms of the properties of the structure present in the scene. It deals with modeling and replicating human vision using computer software and hardware.

Computer Vision overlaps significantly with the following fields −

- **Image Processing** − It focuses on image manipulation.
- **Pattern Recognition** − It explains various techniques to classify patterns.
- **Photogrammetry** − It is concerned with obtaining accurate measurements from images.

Computer Vision Vs Image Processing

Image processing deals with image-to-image transformation. The input and output of image processing are both images.

Computer vision is the construction of explicit, meaningful descriptions of physical objects from their image. The output of computer vision is a description or an interpretation of structures in 3D scene.

Applications of Computer Vision

Here we have listed down some of major domains where Computer Vision is heavily used.

Robotics Application

- Localization − Determine robot location automatically
- Navigation

- Obstacles avoidance
- Assembly (peg-in-hole, welding, painting)
- Manipulation (e.g. PUMA robot manipulator)
- Human Robot Interaction (HRI) − Intelligent robotics to interact with and serve people

Medicine Application

- Classification and detection (e.g. lesion or cells classification and tumor detection)
- 2D/3D segmentation
- 3D human organ reconstruction (MRI or ultrasound)
- Vision-guided robotics surgery

Industrial Automation Application

- Industrial inspection (defect detection)
- Assembly
- Barcode and package label reading
- Object sorting
- Document understanding (e.g. OCR)

Security Application

- Biometrics (iris, finger print, face recognition)
- Surveillance − Detecting certain suspicious activities or behaviors

Transportation Application

- Autonomous vehicle
- Safety, e.g., driver vigilance monitoring

Features of OpenCV Library

Using OpenCV library, you can −

- Read and write images
- Capture and save videos
- Process images (filter, transform)
- Perform feature detection
- Detect specific objects such as faces, eyes, cars, in the videos or images.
- Analyze the video, i.e., estimate the motion in it, subtract the background, and track objects in it.

OpenCV was originally developed in C++. In addition to it, Python and Java bindings were provided. OpenCV runs on various Operating Systems such as windows, Linux, OSx, FreeBSD, Net BSD, Open BSD, etc.

This tutorial explains the concepts of OpenCV with examples using Java bindings.

OpenCV Library Modules

Following are the main library modules of the OpenCV library.

Core Functionality

This module covers the basic data structures such as Scalar, Point, Range, etc., that are used to build OpenCV applications. In addition to these, it also includes the multidimensional array Mat, which is used to store the images. In the Java library of OpenCV, this module is included as a package with the name org.opencv.core.

Image Processing

This module covers various image processing operations such as image filtering, geometrical image transformations, color space conversion, histograms, etc. In the Java library of OpenCV, this module is included as a package with the name org.opencv.imgproc.

Video

This module covers the video analysis concepts such as motion estimation, background subtraction, and object tracking. In the Java library of OpenCV, this module is included as a package with the name org.opencv.video.

Video I/O

This module explains the video capturing and video codecs using OpenCV library. In the Java library of OpenCV, this module is included as a package with the name org.opencv.videoio.

calib3d

This module includes algorithms regarding basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence and elements of 3D reconstruction. In the Java library of OpenCV, this module is included as a package with the name org.opencv.calib3d.

features2d

This module includes the concepts of feature detection and description. In the Java library of OpenCV, this module is included as a package with the name org.opencv.features2d.

Objdetect

This module includes the detection of objects and instances of the predefined classes such as faces, eyes, mugs, people, cars, etc. In the Java library of OpenCV, this module is included as a package with the name org.opencv.objdetect.

Highgui

This is an easy-to-use interface with simple UI capabilities. In the Java library of OpenCV, the features of this module is included in two different packages namely, org.opencv.imgcodecs and org.opencv.videoio.

A Brief History of OpenCV

OpenCV was initially an Intel research initiative to advise CPU-intensive applications. It was officially launched in 1999.

- In the year 2006, its first major version, OpenCV 1.0 was released.

- In October 2009, the second major version, OpenCV 2 was released.
- In August 2012, OpenCV was taken by a nonprofit organization OpenCV.org.

4.6 Yolov3

YOLOv3 (You Only Look Once, Version 3) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. Versions 1-3 of YOLO were created by Joseph Redmon and Ali Farhadi.

The first version of YOLO was created in 2016, and version 3, which is discussed extensively in this article, was made two years later in 2018. YOLO is implemented using the Keras or OpenCV deep learning libraries.

Object classification systems are used by Artificial Intelligence (AI) programs to perceive specific objects in a class as subjects of interest. The systems sort objects in images into groups where objects with similar characteristics are placed together, while others are neglected unless programmed to do otherwise.

How does YOLOv3 work? (Overview)

YOLO is a Convolutional Neural Network (CNN) for doing object detection. CNNs are classifier-based systems that can process input images as structured arrays of data and identify patterns between them. YOLO has the advantage of being much faster than other networks and still maintains accuracy.

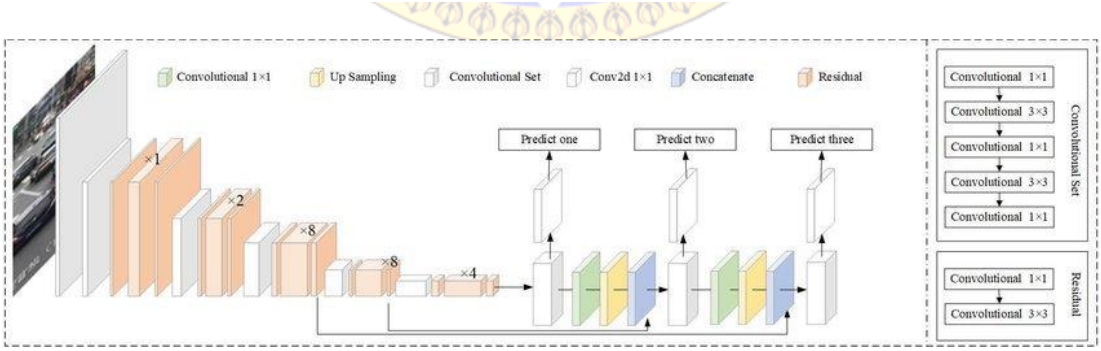


Fig 5: Yolov3

How YOLOv3 works – Source: Mini-YOLOv3: Real-Time Object Detector for Embedded Applications

It allows the model to look at the whole image at test time, so its predictions are informed by the global context in the image. YOLO and other convolutional neural network algorithms “score” regions based on their similarities to predefined classes.

High-scoring regions are noted as positive detections of whatever class they most closely identify with. For example, in a live feed of traffic, YOLO can be used to detect different kinds of vehicles depending on which regions of the video score highly in comparison to predefined classes of vehicles.

The Architecture at a Glance

The YOLOv3 algorithm first separates an image into a grid. Each grid cell predicts some number of boundary boxes (sometimes referred to as anchor boxes) around objects that score highly with the aforementioned predefined classes.

Each boundary box has a respective confidence score of how accurate it assumes that prediction should be and detects only one object per bounding box. The boundary boxes are generated by clustering the dimensions of the ground truth boxes from the original dataset to find the most common shapes and sizes. Other comparable algorithms that can carry out the same objective are R-CNN (Region-based Convolutional Neural Networks made in 2015) and Fast R-CNN (R-CNN improvement developed in 2017), and Mask R-CNN.

However, unlike systems like R-CNN and Fast R-CNN, YOLO is trained to do classification and bounding box regression at the same time.

What’s New in YOLOv3?

There are major differences between YOLOv3 and older versions in terms of speed, precision, and specificity of classes. The following paragraphs will give you an overview of what’s new in YOLOv3.

Speed

YOLOv2 was using Darknet-19 as its backbone feature extractor, while YOLOv3 now uses Darknet-53. Darknet-53 is a backbone also made by the YOLO creators Joseph Redmon and Ali Farhadi.

Darknet-53 has 53 convolutional layers instead of the previous 19, making it more powerful than Darknet-19 and more efficient than competing backbones (ResNet-101 or ResNet-152).

Backbone	Top-1	Top-5	Ops	BFLOP/s	FPS
Darknet-19	74.1	91.8	7.29	1246	**171**
ResNet-101	77.1	93.7	19.7	1039	53
ResNet-152	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Comparison of backbones. Accuracy, billions of operations (Ops), billion floating-point operations per second (BFLOP/s), and frames per second (FPS) for various networks – Source: YOLOv3 Paper

Using the chart provided in the YOLOv3 paper by Redmon and Farhadi, we can see that Darknet-52 is 1.5 times faster than ResNet101. The depicted accuracy doesn't entail any trade-off between accuracy and speed between Darknet backbones either since it is still as accurate as ResNet-152 yet two times faster.

YOLOv3 is fast and accurate in terms of mean average precision (mAP) and intersection over union (IOU) values as well. It runs significantly faster than other detection methods with comparable performance (hence the name – You only look once).

Moreover, you can easily trade-off between speed and accuracy simply by changing the size of the model, and no retraining required.

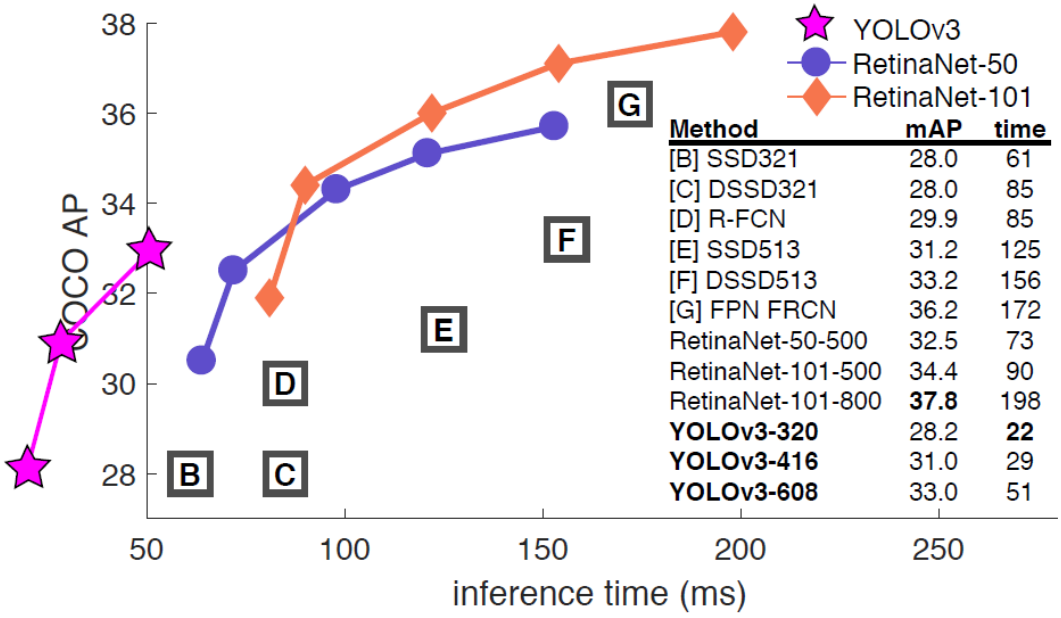


Fig 6: Yolov3

YOLOv3 runs much faster than other detection methods with a comparable performance using an M40/Titan X GPU – Source: Focal Loss for Dense Object Detection

Precision for Small Objects

The chart below (taken and modified from the YOLOv3 paper) shows the average precision (AP) of detecting small, medium, and large images with various algorithms and backbones. The higher the AP, the more accurate it is for that variable.

The precision for small objects in YOLOv2 was incomparable to other algorithms because of how inaccurate YOLO was at detecting small objects. With an AP of 5.0, it paled compared to other algorithms like RetinaNet (21.8) or SSD513 (10.2), which had the second-lowest AP for small objects.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI	Inception-ResNet-v2	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2	DarkNet-19	21.6	44.0	19.2	5.0	22.4	35.5
SSD513	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608×608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

YOLOv3 comparison for different object sizes showing the average precision (AP) for AP-S (small object size), AP-M (medium object size), AP-L (large object size) – Source: Focal Loss for Dense Object Detection

YOLOv3 increased the AP for small objects by 13.3, which is a massive advance from YOLOv2. However, the average precision (AP) for all objects (small, medium, large) is still less than RetinaNet.

Specificity of Classes

The new YOLOv3 uses independent logistic classifiers and binary cross-entropy loss for the class predictions during training. These edits make it possible to use complex datasets such as Microsoft's Open Images Dataset (OID) for YOLOv3 model training. OID contains dozens of overlapping labels, such as "man" and "person" for images in the dataset.

YOLOv3 uses a multilabel approach which allows classes to be more specific and be multiple for individual bounding boxes. Meanwhile, YOLOv2 used a softmax, which is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector.

Using a softmax makes it so that each bounding box can only belong to one class, which is sometimes not the case, especially with datasets like OID.

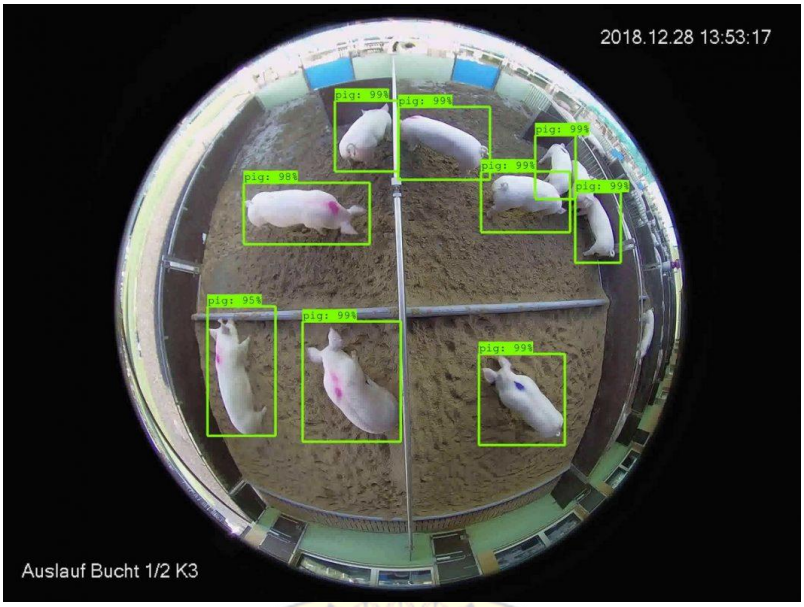


Fig 7: Yolov3

Object Detection to recognize animals with YOLO in a farming application

Disadvantages of YOLOv3 vs. Other Algorithms

The YOLOv3 AP does indicate a trade-off between speed and accuracy for using YOLO when compared to RetinaNet since RetinaNet training time is greater than YOLOv3. However, the accuracy of detecting objects with YOLOv3 can be made equal to the accuracy when using RetinaNet by having a larger dataset, making it an ideal option for models that can be trained with large datasets.

An example of this would be common detection models like traffic detection, where plenty of data can be used to train the model since the number of images of different vehicles is plentiful. On the other hand, YOLOv3 may not be ideal to use with niche models where large datasets can be hard to obtain.

4.7 Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production. Docker provides the ability to package and run an application in a loosely isolated environment called a container.

The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines.

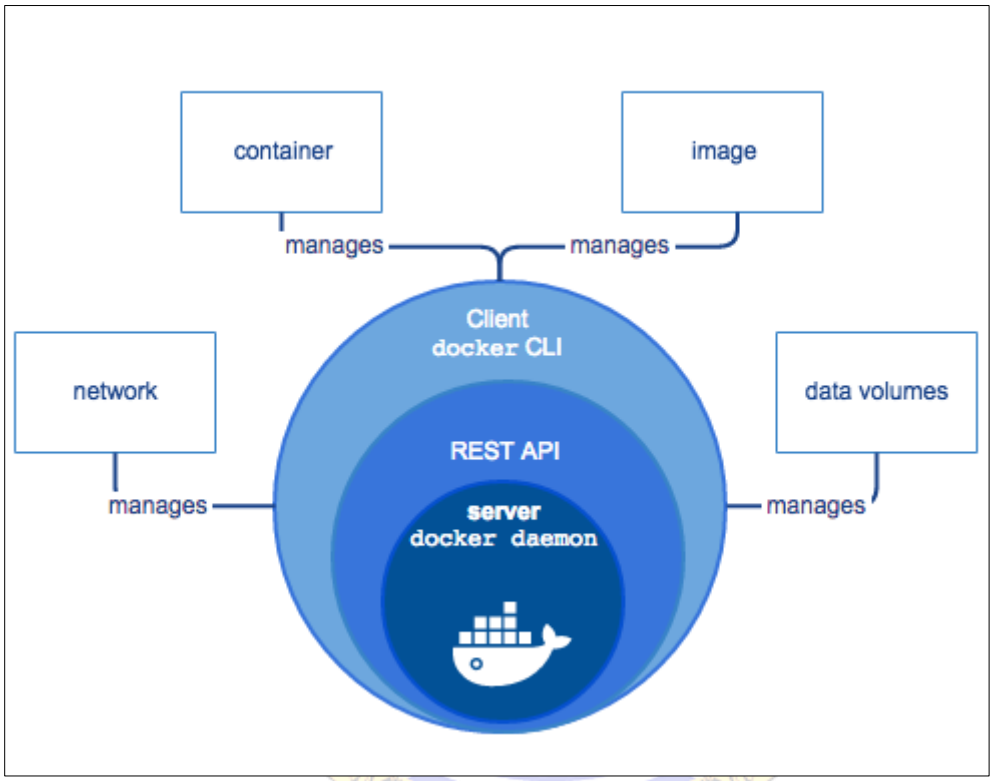


Fig 8: Docker Engine

The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI.

The daemon creates and manages Docker *objects*, such as images, containers, networks, and volumes.

4.7.1 Use of Docker

Containers work a little like VMs, but in a far more specific and granular way. They isolate a single application and its dependencies—all of the external software libraries the app requires to run—both from the underlying operating system and from other

containers. All of the containerized apps share a single, common operating system (either Linux or Windows), but they are compartmentalized from one another and from the system at large.

The benefits of Docker containers show up in many places. Here are some of the major advantages of Docker and containers:

- **Docker enables faster software delivery cycles:** Enterprise software must respond quickly to changing conditions. That means both easy scaling to meet demand and easy updating to add new features as the business requires. Docker containers make it easy to put new versions of software, with new business features, into production quickly—and to quickly roll back to a previous version if you need to. They also make it easier to implement strategies like blue/green deployments.

- **Docker enables application portability:** Where you run an enterprise application matters—behind the firewall, for the sake of keeping things close by and secure; or out in a public cloud, for easy public access and high elasticity of resources. Because Docker containers encapsulate everything an application needs to run (and only those things), they allow applications to be shuttled easily between environments. Any host with the Docker runtime installed—be it a developer's laptop or a public cloud instance—can run a Docker container.

- **Docker shines for microservices architecture:** Lightweight, portable, and self-contained, Docker containers make it easier to build software along forward-thinking lines, so that you're not trying to solve tomorrow's problems with yesterday's development methods.One of the software patterns containers make easier is microservices, where applications are constituted from many loosely coupled components.

4.7.2 Docker-compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. To learn more about all the features of Compose, see the list of features.

Compose works in all environments: production, staging, development, testing, as well as CI workflows. You can learn more about each case in Common Use Cases.

4.8 Amazon Web Services (AWS)

Amazon Web Services (AWS) is a cloud service from Amazon, which provides services in the form of building blocks, these building blocks can be used to create and deploy any type of application in the cloud.

These services or building blocks are designed to work with each other, and result in applications that are sophisticated and highly scalable.

What are the services provided by AWS?

Each type of service in this "What is AWS" blog, is categorized under a domain, the few domains which are widely used are:

- Compute
- Storage
- Database
- Migration
- Network and Content Delivery
- Management Tools
- Security & Identity Compliance
- Messaging

4.8.1 EC2 instance of AWS

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment.

Amazon EC2 offers the broadest and deepest compute platform with choice of processor, storage, networking, operating system, and purchase model. We offer the fastest processors in the cloud and we are the only cloud with 400 Gbps ethernet networking. We have the most powerful GPU instances for machine learning training and graphics workloads, as well as the lowest cost-per-inference instances in the cloud. More SAP, HPC, Machine Learning, and Windows workloads run on AWS than any other cloud.

Deploy EC2 instance :-

Step 1: Login to AWS Management Console.

Step 2: Select a region from the drop down.

Step 3: Click EC2 under Compute section. This will take you to EC2 dashboard.

Step 4: Select Launch Instance and hence select an AMI, for our example in this What is AWS blog, we will be selecting a Windows 2016 Server Instance which falls under free tier.

Step 5: Once you select your desired AMI, select your instance type, this is basically where you decide how much computing power you need to start, since ours is a small application, we shall suffice with the free tier.

Step 6: Configure all the details and then click on add storage.

Step 7: Here you will be configuring your storage devices, once done click on tag instance.

Step 8: Here you will be tagging your instance, this is how your instance will be identified.

Step 9: Now you will be configuring your security group.

Step 10: Check all your settings, once verified launch your instance!

Step 11: In the next step you will be prompted for a key pair, create one and download at a handy location.

Step 12: Select your instance and click on Connect.

Step 13: Once you click connect, you will be prompted with the following screen. Copy the public IP and then click on Get Password.

Step 14: Select the key-pair that you downloaded, then click decrypt password.

Step 15: Copy the password and the public IP, keep it handy for the next step.

Step 16: We have the Public IP and the password now, let's connect to our instance! Open the remote desktop manager. Enter the public IP address and click on Connect.

Step 17: Enter the saved password here and click on OK.

Step 18: Congratulations! Windows Server on EC2 at your service!

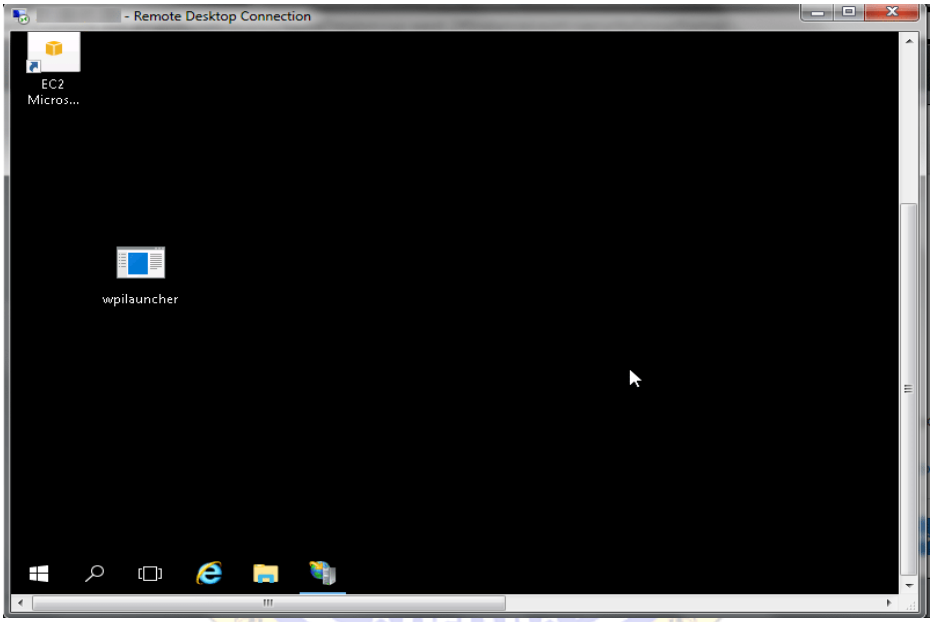


Fig 9: EC2 instance of Windows Server

4.11.2 Docker on AWS EC2 instance

I will show you how to install **Docker** on **AWS EC2** instance and run your first **Docker container**.

1—Setup EC2 instance

I already did a tutorial on how to create an **EC2** instance, so I won't repeat it. There are few ways you'll want to differ from the tutorial:

1 We select the "**Amazon Linux AMI 2017.03.1 (HVM), SSH Volume Type**" as **AMI**. The exact versions may change with time.

2 We configure the **security groups** as below. This setting allows access to port **80** (**HTTP**) from anywhere, and **SSH** access also.

Go ahead and launch the instance, it will take couple of minutes:

2—Install Docker

Once your instance is ready to use, connect via **SSH** to the server using the **public DNS** and the **public key**:

Once connected, use **yum** configuration manager to install **Docker**, by typing the following commands:

sudo yum update -y

sudo yum install -y docker

Next, start the docker service

In order to user docker command without **root** privileges (**sudo**), we need to add **ec2-user** to the **docker group**:

sudo usermod -aG docker ec2-user

To verify that docker is correctly installed, just type:

As you can see the latest version of docker has been installed (**v17.03.1-ce**)

Congratulation ! you have now an **EC2** instance with **Docker** installed.

3—Deploy Docker Container

It's time to run your first container.

We will create an **nginx** container with this command:

If we run the list command "**docker ps**", we can see that an **nginx container** has been created from the **nginx official image**.

Finally, you visit your instance **public DNS name** in your browser, you should see something like this below:

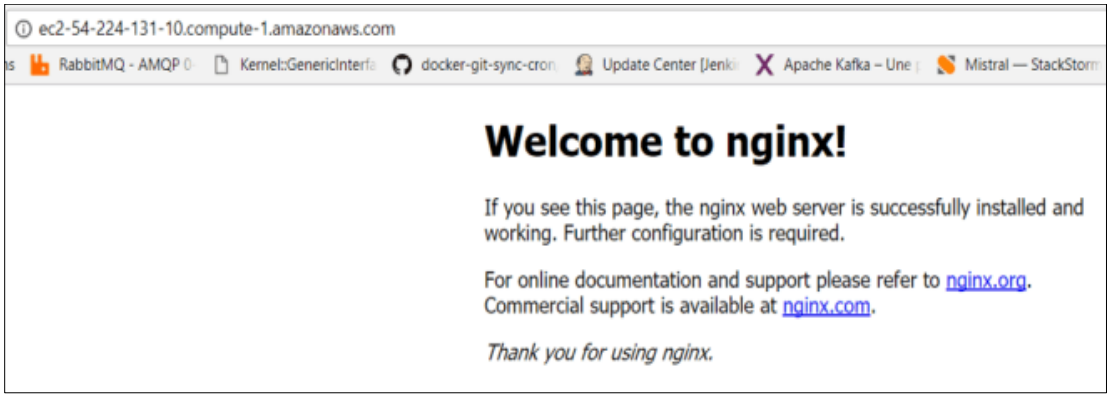


Fig 10: Nginx

5. OBJECT DETECTION

5.1 Introduction

Object detection is a computer vision technique that works to identify and locate objects within an image or video. Specifically, object detection draws bounding boxes around these detected objects, which allow us to locate where said objects are in (or how they move through) a given scene.

Object detection is commonly confused with image recognition, so before we proceed, it's important that we clarify the distinctions between them.

Image recognition assigns a label to an image. A picture of a dog receives the label "dog". A picture of two dogs, still receives the label "dog". Object detection, on the other hand, draws a box around each dog and labels the box "dog". The model predicts where each object is and what label should be applied. In that way, object detection provides more information about an image than recognition.

Here's an example of how this distinction looks in practice:

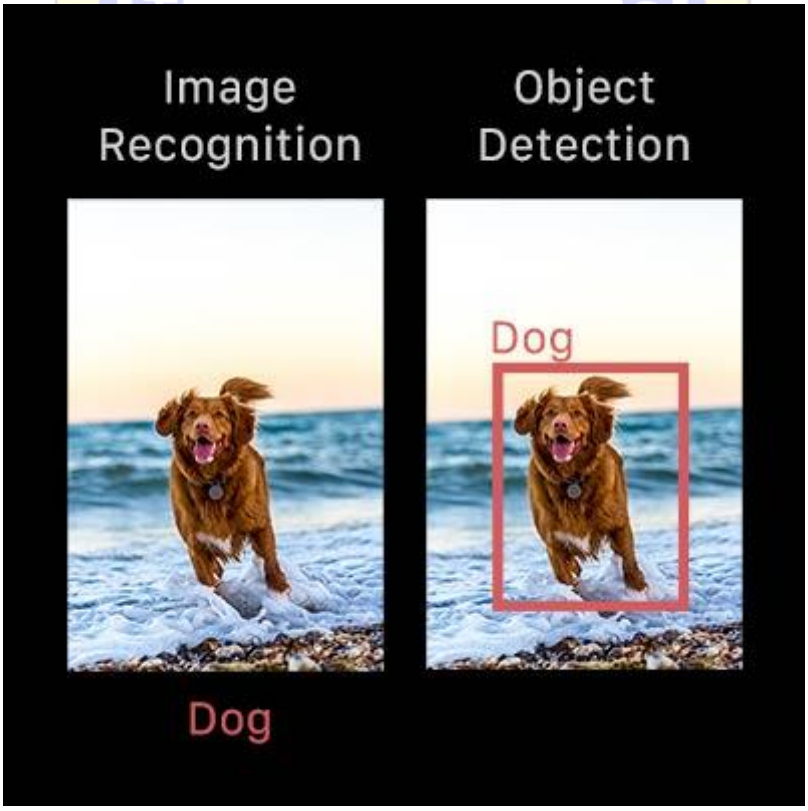


Fig 11: Object Detection

5.2 Modes and types of object detection

Broadly speaking, object detection can be broken down into machine learning-based approaches and deep learning-based approaches.

In more traditional ML-based approaches, computer vision techniques are used to look at various features of an image, such as the color histogram or edges, to identify groups of pixels that may belong to an object. These features are then fed into a regression model that predicts the location of the object along with its label.

On the other hand, deep learning-based approaches employ convolutional neural networks (CNNs) to perform end-to-end, unsupervised object detection, in which features don't need to be defined and extracted separately. For a gentle introduction to CNNs, check out this overview.

Because deep learning methods have become the state-of-the-art approaches to object detection, these are the techniques we'll be focusing on for the purposes of this guide.

5.3 Why is object detection important?

Object detection is inextricably linked to other similar computer vision techniques like image recognition and image segmentation, in that it helps us understand and analyze scenes in images or video.

But there are important differences. Image recognition only outputs a class label for an identified object, and image segmentation creates a pixel-level understanding of a scene's elements. What separates object detection from these other tasks is its unique ability to locate objects within an image or video. This then allows us to count and then track those objects.

Given these key distinctions and object detection's unique capabilities, we can see how it can be applied in a number of ways:

- Crowd counting
- Self-driving cars
- Video surveillance

- Face detection
- Anomaly detection

Of course, this isn't an exhaustive list, but it includes some of the primary ways in which object detection is shaping our future.

5.4 Basic structure

Deep learning-based object detection models typically have two parts. An encoder takes an image as input and runs it through a series of blocks and layers that learn to extract statistical features used to locate and label objects. Outputs from the encoder are then passed to a decoder, which predicts bounding boxes and labels for each object.

The simplest decoder is a pure regressor. The regressor is connected to the output of the encoder and predicts the location and size of each bounding box directly. The output of the model is the X, Y coordinate pair for the object and its extent in the image. Though simple, this type of model is limited. You need to specify the number of boxes ahead of time. If your image has two dogs, but your model was only designed to detect a single object, one will go unlabeled. However, if you know the number of objects you need to predict in each image ahead of time, pure regressor-based models may be a good option.

An extension of the regressor approach is a region proposal network. In this decoder, the model proposes regions of an image where it believes an object might reside. The pixels belonging to these regions are then fed into a classification subnetwork to determine a label (or reject the proposal). It then runs the pixels containing those regions through a classification network. The benefit of this method is a more accurate, flexible model that can propose arbitrary numbers of regions that may contain a bounding box. The added accuracy, though, comes at the cost of computational efficiency.

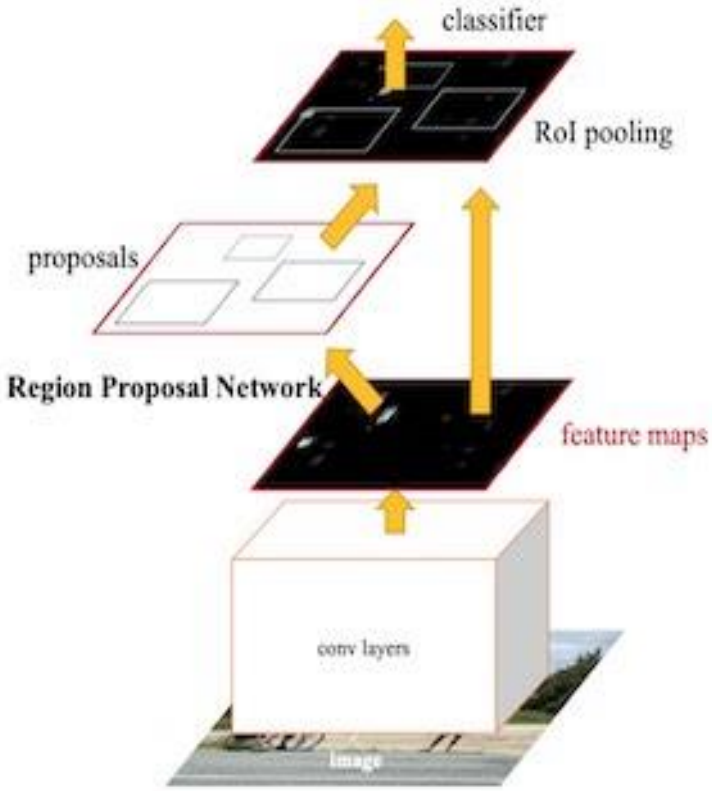


Fig 12: Object Detection

Single shot detectors (SSDs) seek a middle ground. Rather than using a subnetwork to propose regions, SSDs rely on a set of predetermined regions. A grid of anchor points is laid over the input image, and at each anchor point, boxes of multiple shapes and sizes serve as regions. For each box at each anchor point, the model outputs a prediction of whether or not an object exists within the region and modifications to the box's location and size to make it fit the object more closely. Because there are multiple boxes at each anchor point and anchor points may be close together, SSDs produce many potential detections that overlap. Post-processing must be applied to SSD outputs in order to prune away most of these predictions and pick the best one. The most popular post-processing technique is known as non-maximum suppression.

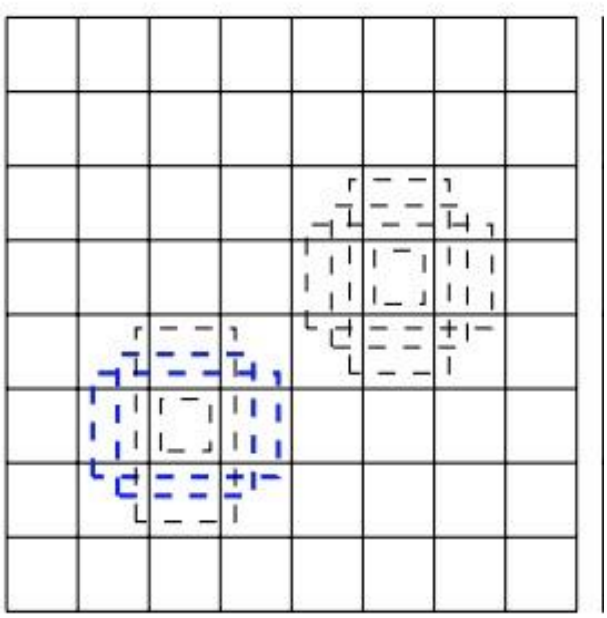


Fig 13: Object Detection

Finally, a note on accuracy. Object detectors output the location and label for each object, but how do we know how well the model is doing? For an object's location, the most commonly-used metric is intersection-over-union (IOU). Given two bounding boxes, we compute the area of the intersection and divide by the area of the union. This value ranges from 0 (no interaction) to 1 (perfectly overlapping). For labels, a simple "percent correct" can be used.

5.5 Use cases and applications

In this section, we'll provide an overview of real-world use cases for object detection. We've mentioned several of them in previous sections, but here we'll dive a bit deeper and explore the impact this computer vision technique can have across industries.

Specifically, we'll examine how object detection can be used in the following areas:

- Video surveillance
- Crowd counting
- Anomaly detection (i.e. in industries like agriculture, health care)
- Self-driving cars

5.5.1 Video surveillance

Because state-of-the-art object detection techniques can accurately identify and track multiple instances of a given object in a scene, these techniques naturally lend themselves to automating video surveillance systems.

For instance, object detection models are capable of tracking multiple people at once, in real-time, as they move through a given scene or across video frames. From retail stores to industrial factory floors, this kind of granular tracking could provide invaluable insights into security, worker performance and safety, retail foot traffic, and more.

5.5.2Crowd counting

Crowd counting is another valuable application of object detection. For densely populated areas like theme parks, malls, and city squares, object detection can help businesses and municipalities more effectively measure different kinds of traffic—whether on foot, in vehicles, or otherwise. This ability to localize and track people as they maneuver through various spaces could help businesses optimize anything from logistics pipelines and inventory management, to store hours, to shift scheduling, and more. Similarly, object detection could help cities plan events, dedicate municipal resources, etc.

5.5.3 Anomaly detection

In agriculture, for instance, a custom object detection model could accurately identify and locate potential instances of plant disease, allowing farmers to detect threats to their crop yields that would otherwise not be discernible to the naked human eye.

And in health care, object detection could be used to help treat conditions that have specific and unique symptomatic lesions. One such example of this comes in the form of skin care and the treatment of acne—an object detection model could locate and identify instances of acne in seconds.

What's particularly important and compelling about these potential use cases is how they leverage and provide knowledge and information that's generally only available to agricultural experts or doctors, respectively.

5.5.4 Self-driving cars

Real-time car detection models are key to the success of autonomous vehicle systems. These systems need to be able to identify, locate, and track objects around them in order to move through the world safely and efficiently.

And while tasks like image segmentation can be (and often are) applied to autonomous vehicles, object detection remains a foundational task that underpins current work on making self-driving cars a reality.

6. METHODOLOGY

6.1 Introduction

In this work, we have attempted to develop an integrated framework for reconnaissance security that distinguishes the weapons progressively, if identification is positively true it will caution/brief the security personals to handle the circumstance by arriving at the place of the incident through IP cameras. We propose a model that provides a visionary sense to a machine to identify the unsafe weapon and can also alert the human administrator when a gun or firearm is obvious in the edge. Moreover, we have programmed entryways locking framework when the shooter seems to carry appalling weapon. On the off chance conceivable, through IP webcams we can likewise share the live photo to approach security personals to make the move in meantime. Also, we have constructed the information system for recording all the exercises to convey impact activities in the metropolitan territories for a future crisis. This further ends up in designing the database for recording all the activities in order to take prompt actions for future emergency. Figure below presents the overall generalized approach of our research work divided into three parts.

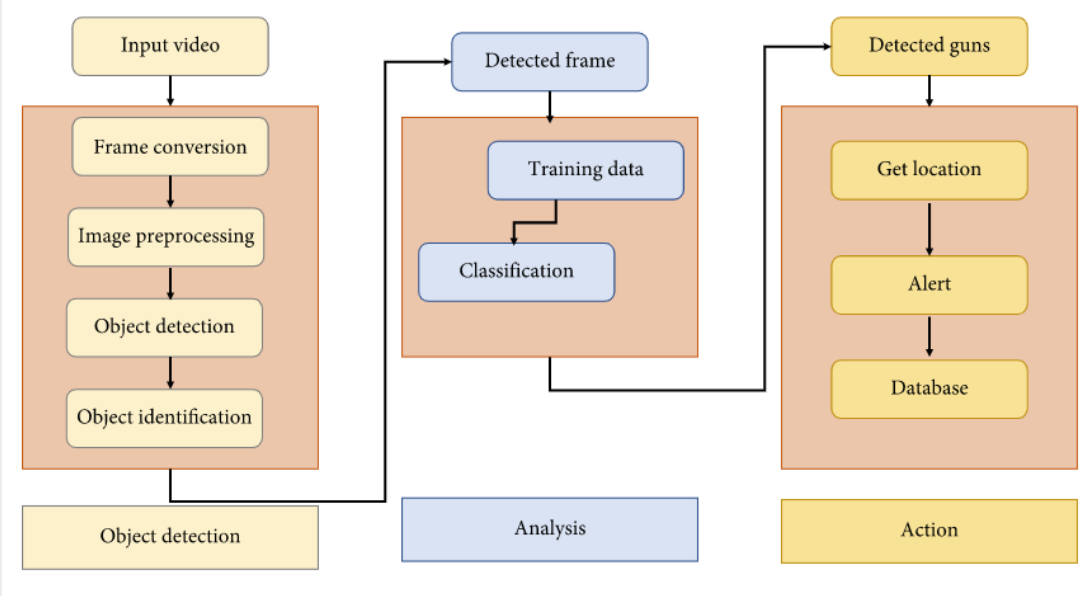


Fig 14: Methodology

The most important and crucial part of any application is to have a desired and suitable dataset in order to train the machine learning models. Therefore, we manually collected huge amount of images from Google. A few of the image samples are shown in Figure. For each weapon class, we collected at least 50 images. Using google-images-download is one of the best ways to collect images for constructing one's own dataset. We

further saved those images to a folder called “images.” One must save images in “.jpg” form; if the images are in different extensions, it will be a little troublesome and will generate errors when provided for training. Alternatively, since the images are processed in terms of batches, therefore prior to training, the sizes of all the images are transformed into the same width and height 416 × 416 pixels.

Fig 15 : Methodology

Object detection is primarily related to computer vision that includes distinguishing objects in computerized images. Object detection is a domain that has benefited immensely from the recent advancements in the realm of deep learning. YOLO is basically a pretrained object detector. It is a CNN model. A CNN is a deep learning algorithm which can take in a raw input image and assign learnable weights and biases to various aspects/objects in the image. A convolutional layer in CNN model is responsible of extracting the high-level features such as edges, from the input image. This works by applying kxk filter known as kernel repeatedly over raw image. This further results in activation maps or feature maps. These feature maps are the presence of detected features from the given input. Thus, the preprocessing required is much lower

as compared to other classification algorithms, whereas in standard approach, filters are hand-engineered and in CNN these are learned through a number of iterations and training. Figure indicates a basic CNN architecture as classification model for 10 different weapons. Subsequently, the next layer is Max-Pooling or Subsampling layer, which is responsible for reducing the spatial size of the convolved features. This is to decrease the computational power required to process the data through dimensionality reduction. ReLU is a rectified linear unit activation expressed in, which is related to the feature of nonsaturating activation. It eliminates undesirable values from an activation map effectively by setting them to nil. Finally, the last layers are fully connected layers transforming the data into a 1-dimensional array. To create a particular long feature vector, the flattened output is fed to a feedforward neural network and backpropagation is applied to every iteration of training. These layers are liable to learn nonlinear combinations of the high-level features as represented by the output of the convolutional layer.

$$\text{ReLU}: f(x) = \max(0, x).$$

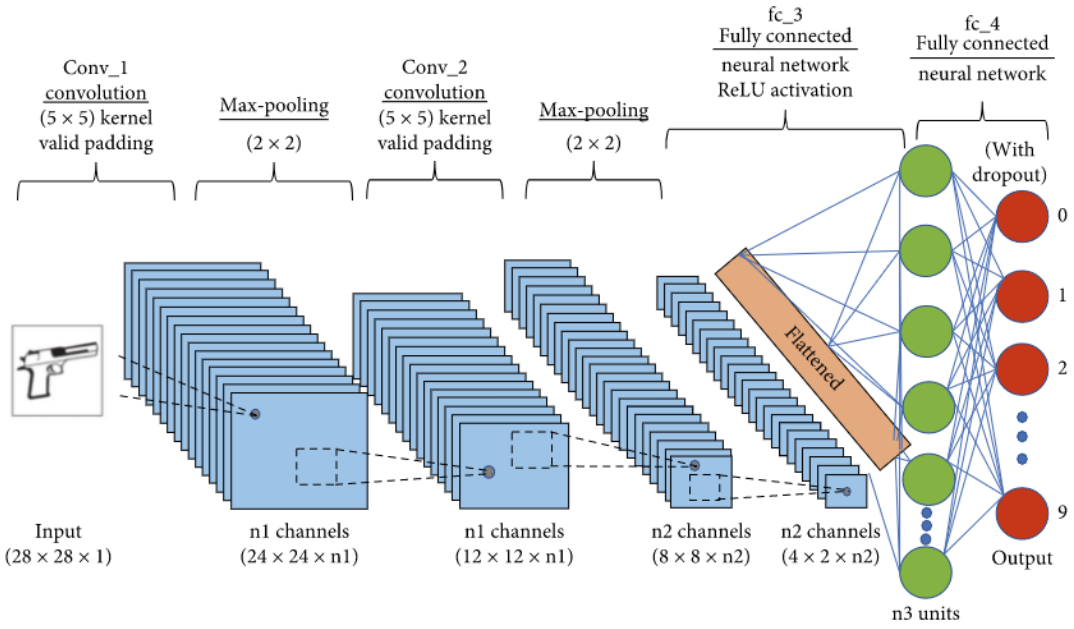


Fig 16: Methodology

As mentioned earlier that YOLO is a pretrained object detector, a pretrained model simply means that another dataset has been trained on it. It is extremely time consuming to train a model from scratch; it can take weeks or a month to complete the training step. A pretrained model has already seen tons of objects and knows how each of them

must be classified. The weights in the abovementioned pretrained model have been obtained by training the network on COCO and Imagenet dataset. Thus, it can only detect objects belonging to the classes present in the dataset used to train the network. It uses Darknet-53 as the backbone network for feature extraction and uses three scale predictions. The DarkNet-53 is again convolutional neural network that has 53 layers as elucidated in Figure. DarkNet-53 is a fully convolutional neural network. Pooling layer is replaced with a convolution operation with stride 2. Furthermore, residual units are applied to avoid the gradient dispersion.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3/2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3/2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3/2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3/2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3/2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fig 17: Methodology

Initially, CNN architectures were quite linear. Recently, numerous variations are introduced, for example, middle blocks, skip connections, and aggregations of data between layers. These network models have already acquired rich feature representations by getting trained over a wide range of images. Thus, selecting a pretrained network and using it as a starting point to learn a new task is a concept behind transfer learning.

In order to recognize the weapons, we took the weights of a pretrained model and trained another YOLO V3 model.

YOLO V3 is designed to be a multiscaled detector rather than image classifier. Therefore, for object detection, classification head is replaced by appending a detection head to this architecture. Henceforth, the output is vector with the bounding box coordinates and probability classes. YOLO V3 inherits Darknet-53 as its backbone, a framework to train neural networks with 53 layers as indicated in Figure 4. Moreover, for object detection task additional 53 layers are stacked over it, accumulating to a total of a 106-layer fully convolutional architecture. Due to its multiscale feature fusion layers, YOLO V3 uses 3 feature maps of different scales for target detection as shown in Figure.

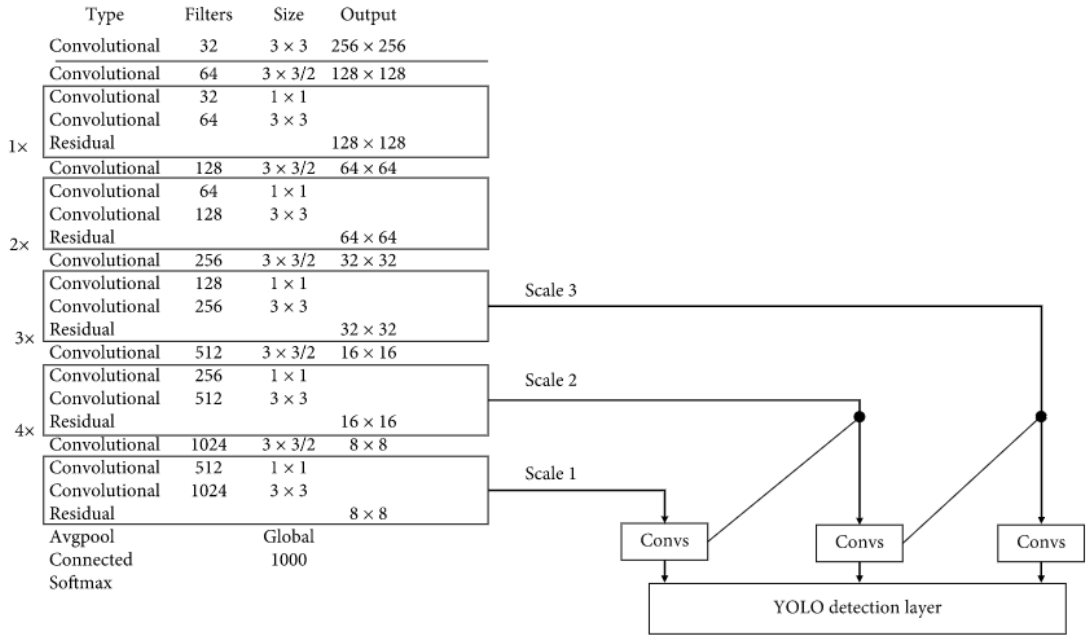


Fig 18: Methodology

7. TEST PLAN

7.1 Test Plan of The Gun Detector

ACTIVITY NETWORK FOR SYSTEM TESTING

The test plan entails the following activities:

1. Prepare test plan.
2. Specify conditions for user acceptance testing.
3. Prepare test data for program testing.
4. Prepare test data for transaction path testing.
5. Plan user training.
6. Compile/assemble programs.
7. Prepare job performance aids.
8. Prepare operational documents.

PREPARE TEST: A workable test plan must be prepared in accordance with established design specifications. It includes the following items:

- Outputs expected from the system.
- Criteria for evaluating outputs.
- A volume of test data.
- Procedure for using test data.
- Personnel and training requirements.

SPECIFY CONDITIONS FOR USER ACCEPTANCE TESTING

Planning for user acceptance testing it calls for the analyst and the user to agree on conditions for the test.

PREPARE TEST DATA FOR PROGRAM TESTING

As each program is coded, test data are prepared and documented to ensure that all aspects of the program are properly tested.

PLAN USER TRAINING

User training is designed to prepare the user for testing and converting the system. User involvement and training take place parallel with programming for three reasons:

• The system group has time available to spend on training while the programs are being written.

• Initiating a user-training program gives the systems group a clearer image of the user's interest in the new system.

• A trained user participates more effectively in system testing.

The training plan is followed by preparation of the user training manual and other text materials.

COMPILE / ASSEMBLE PROGRAMS

All programs have to be compiled / assembled for testing.

PREPARE JOB PERFORMANCE AIDS

In this activity the materials to be used by personnel to run the system are specified and scheduled. This includes a display of materials.

PREPARE OPERATIONAL DOCUMENTS

During the test plan stage, all operational documents are finalized including copies of the operational formats required by the candidate system.

SYSTEMS TESTING

The computer department to ensure that the system functions as specified does this testing. This testing is important to ensure that a working system is handed over to the user for acceptance testing.

7.2 Test Cases

Test	**Ensure video is uploaded**
Type	Integration
Pre-Condition	Setup server and visit Gun Detection for Security Surveillance
Steps	1 Clilk on Browse Files button. 2 Select video from your local machine.
Expected Result	Video will get uploaded

Test Case – 1

Test	**Ensure gun is detected**
Type	Integration
Pre-Condition	Setup server and visit Gun Detection for Security Surveillance
Steps	1 Click on Browse Files button. 2 Select video from your local machine.
Expected Result	• Gun is detected from the video • Beep sound • Call on your phone

Test Case - 2

Test	**Ensure gun not detected**
Type	Integration
Pre-Condition	Setup server and visit The Colourizer Site
Steps	1 Click on Browse Files button. 2 Select video from your local machine.
Expected Result	Gun is not detected from the video

Test Case – 3

8. CONCLUSIONS & FUTURE SCOPE

The weapon detection in surveillance system using yolov3 algorithm is faster than the previous CNN, R-CNN and faster CNN algorithms. In this era where things are automated, object detection becomes one of the most interesting field. When it comes to object detection in surveillance systems, speed plays an important role for locating an object quickly and alerting the authority. This work tried to achieve the same and its able to produce a faster result compared to the previously existing systems.

In future we will increase the file size limit the video. We will optimize our project so that it can convert a whole black and white movie into coloured movie. We will develop a new CNN model to increase the video resolution. This feature enhances the quality of converted video and help to study the video much precisely.

REFERENCES

[1] F. Enr´ıquez, L. M. Soria, J. A. Alvarez-Garc´ıa, F. S. Caparrini, F. Velasco, O. Deniz, N. Vallez, Vision and crowdsensing technology for an optimal response in physical-security, in: International Conference on Computational Science, Springer, 2019, pp. 15–26.

[2] J. L. Salazar, L. M. Soria, J. A. Alvarez-Garc´ıa, F. Enr´ıquez, A. R. Jim´enez, Energy-efficient indoor localization wifi-fingerprint system: An experimental study, IEEE Access 7 (2019) 162664–162682.

[3] L. W. Sommer, T. Schuchert, J. Beyerer, Fast Deep Vehicle Detection in Aerial Images, in: 2017 IEEE Winter Conference on Applications of Computer Vision (WACV 2017), 2017, pp. 311–319.

[4] A. V. Etten, You only look twice: Rapid multi-scale object detection in satellite imagery (2018).

[5] J. Li, X. Liang, Y. Wei, T. Xu, J. Feng, S. Yan, Perceptual Generative Adversarial Networks for Small Object Detection, in: 30TH IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017), 2017, pp. 1951–1959.

[6] M. Kisantal, Z. Wojna, J. Murawski, J. Naruniec, K. Cho, Augmentation for small object detection (2019).

[7] Hariharan, B., Arbel´aez, P., Girshick, R., Malik, J.: Hypercolumns for object segmentation and fine-grained localization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2015) 447–456

[8] Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., Oliva, A.: Learning deep features for scene recognition using places database (2014) 487–495

[9] A. Arcos-Garcia, J. A. Alvarez-Garcia, L. M. Soria-Morillo, Evaluation of deep neural networks for traffic sign detection systems, Neurocomputing 316 (2018) 332–344.

[10] Alvaro Arcos-Garc´ıa, J. A. Alvarez Garc´ıa, L. M. Soria-Morillo, Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods, Neural Networks 99 (2018) 158 – 165.
URL http://www.sciencedirect.com/science/article/pii/S0893608018300054

[11] Cheng, Z., Yang, Q., Sheng, B.: Deep colorization. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 415–423

[12] Luciano Ramalho, “Fluent Python”, O’Reilly, Pg: 73-80

[13] James Turnbull, “The Docker Book: Containerization is the new virtualization”, Kindle Edition Pg 115- 124, 220- 240

[14] Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms, Nicholas Locascio and Nikhil Buduma

[15] A. Joshi, N. Jagdale, R. Gandhi, and S. Chaudhari, “Smart surveillance system for detection of suspicious behaviour using machine learning,” in Intelligent Computing, Information and Control Systems. ICICCS 2019. Advances in Intelligent Systems and Computing, A. Pandian, K. Ntalianis, and R. Palanisamy, Eds., vol. 1039, Springer, Cham, Berlin, Germany, 2020.

[16]https://docs.docker.com/engine/reference/builder

[17] Baldassarre, F., Morín, D. G., & Rodés-Guirao, L. (2017). Deep koalarization: Image colorization using cnns and inception-resnet-v2. arXiv preprint arXiv:1712.03400.

[18] S. B. Kibria and M. S. Hasan, “An analysis of feature extraction and classification algorithms for dangerous object detection,” in Proceedings of the 2017 2nd International Conference on Electrical & Electronic Engineering (ICEEE), pp. 1–4, IEEE, Rajshahi, Bangladesh, December 2017.

[19] A. Castillo, S. Tabik, F. Pérez, R. Olmos, and F. Herrera, “Brightness guided preprocessing for automatic cold steel weapon detection in surveillance videos with deep learning,” Neurocomputing, vol. 330, pp. 151–161, 2019.

[20] G. K. Verma and A. Dhillon, “A handheld gun detection using faster r-cnn deep learning,” in Proceedings of the 7th International Conference on Computer and Communication Technology, pp. 84–88, Kurukshetra, Haryana, November 2017.

[21] https://docs.docker.com/compose/

[22] R. Xu, S. Y. Nikouei, Y. Chen et al., “Real-time human objects tracking for smart surveillance at the edge,” in Proceedings of the 2018 IEEE International Conference on Communications (ICC), pp. 1–6, Kansas City, MO, USA, May 2018

[23]https://docs.streamlit.io/en/stable/api.html?highlight=fil %20upload#streamlit.file_uploader

[24] United Nations, *Office on Drugs and Crime, Report on “Global Study of Homicide”*, https://www.unodc.org/documents/data-and-analysis/gsh/Booklet1.pdf.

[25] https://towardsdatascience.com/how-to-deploy-a-semantic-search-engine-with-streamlit-and-doc[ker-on-aws-elastic-beanstalk-42ddce0422f3

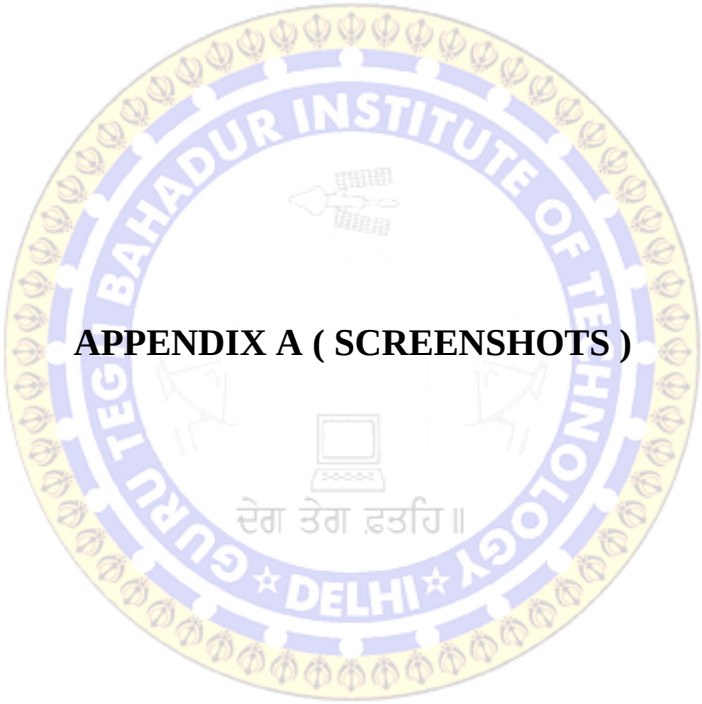
[26] https://maelfabien.github.io/project/Streamlit/

[27] Pramod Singh, “Deploy Machine Learning Models to Production”, Apress, Pg 150-155, 201-210

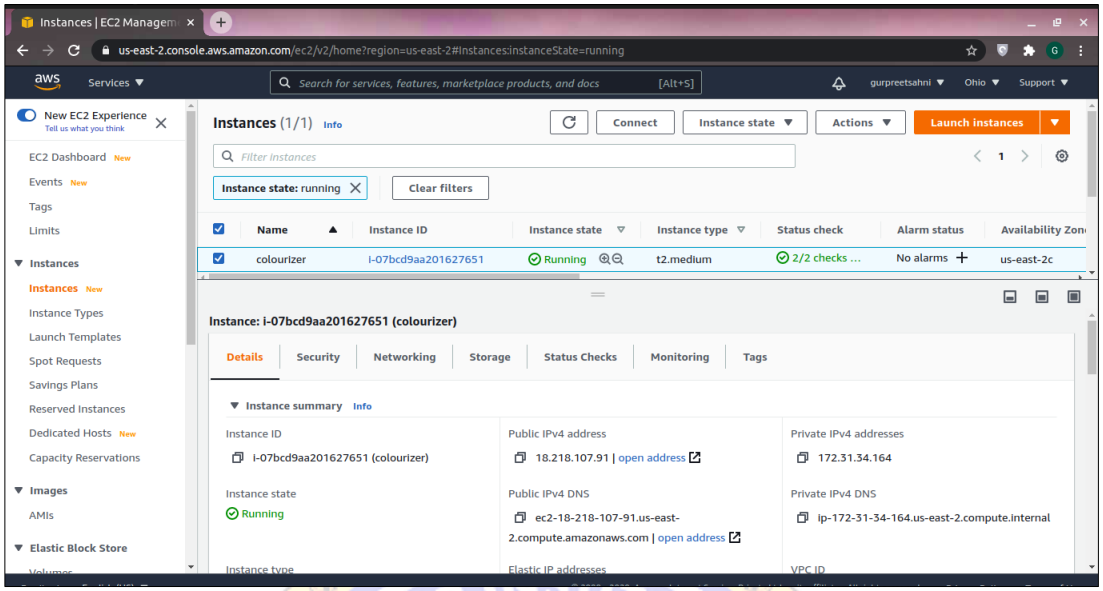
[28] https://github.com/MrTomerLevi/streamlit-docker

[29] https://aws.amazon.com/machine-learning/containers/

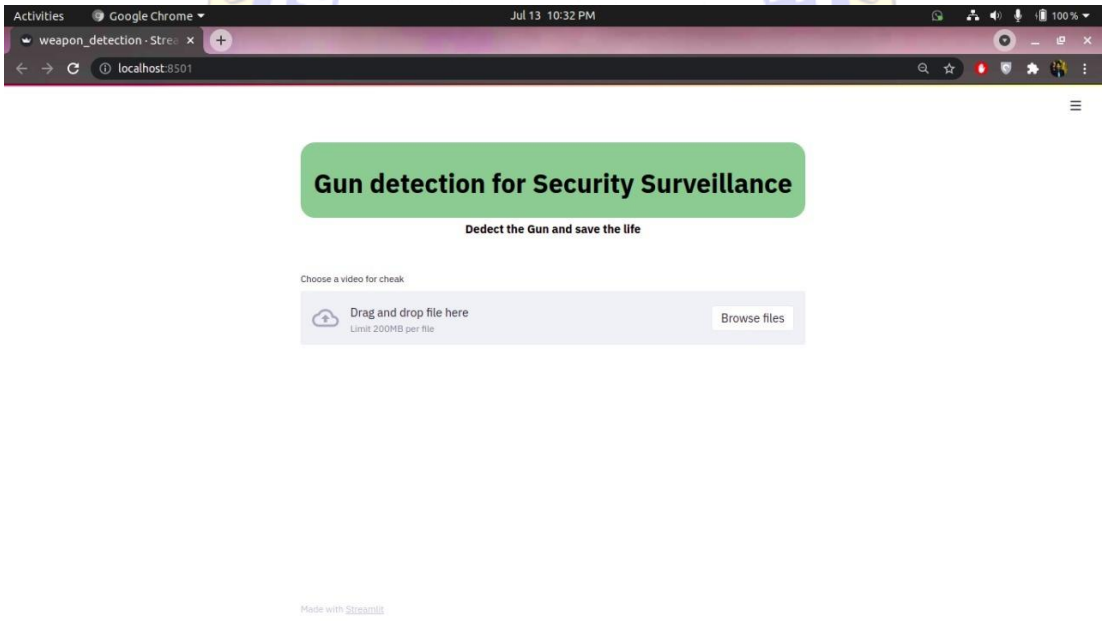
[30]https://aws.amazon.com/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc



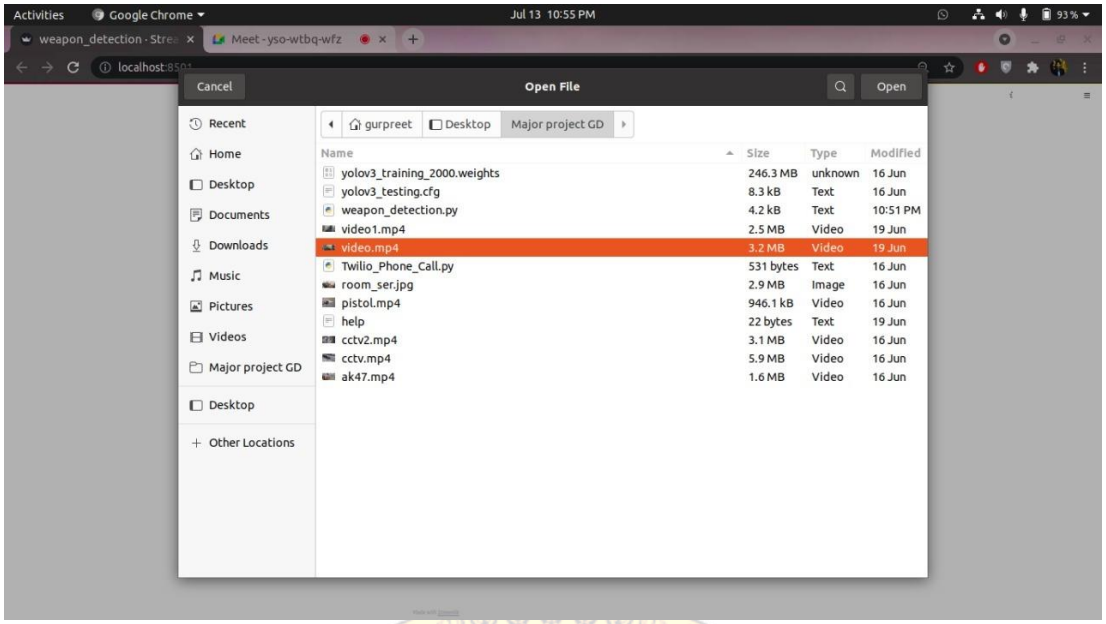
APPENDIX A (SCREENSHOTS)



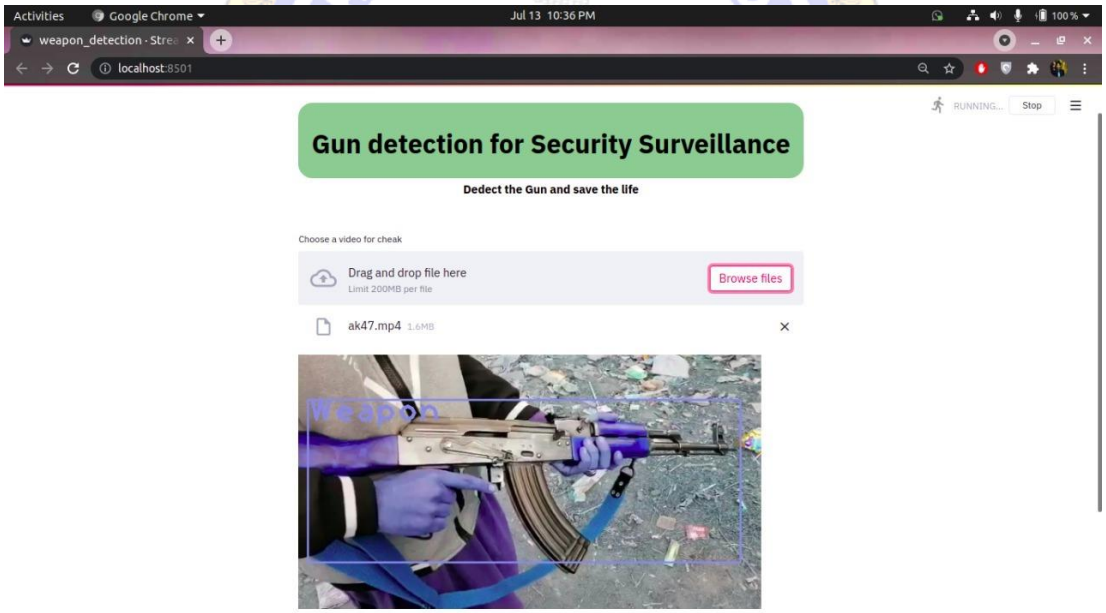
Run The Gun Detector instance on AWS



Gun Detection for Security Surveillance



Select Video



Gun Detected

APPENDIX B (SOURCE CODE)

```
import cv2

import numpy as np

import tempfile

#import winsound

import streamlit as st

from twilio.rest import Client

import os

# My Twilio recovery Code :- 9zuJ6L0MS0I734uZX70N4S6-nLY3rgHvR6N2Adds

account_sid= 'AC783bccd15ef770c344a9ab74eaa68778'

auth_token ='fb3fc986dc276b76e41c307a0a97eedb'

client = Client(account_sid, auth_token)

net = cv2.dnn.readNet(r"./yolov3_training_2000.weights", r"./yolov3_testing.cfg")

classes = ["Weapon"]

layer_names = net.getLayerNames()

output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

colors = np.random.uniform(0, 255, size=(len(classes), 3))

# html Gun Dedection

html = '''

<div style="background-color:#9AC293;padding:13px;border-radius:20px">

<h1 style="color:black;text-align:center;"> Gun detection for Security Surveillance </h1>

'''

st.markdown(html, unsafe_allow_html=True)

st.markdown('<h4 style="color:black;text-align:center;"> Dedect the Gun and save the life </h4>', unsafe_allow_html=True)
```

```
st.write("#")
uploaded_file = st.file_uploader("Choose a video for cheak")
if uploaded_file is not None:
        tfile = tempfile.NamedTemporaryFile(delete=False)
        tfile.write(uploaded_file.read())
        #st.video(tfile.name)
        cap = cv2.VideoCapture(tfile.name)

        def value(val):
           #val = uploaded_file.get
           #val = input("Enter file name or press enter to start webcam : \n")
           if val == "":
              val = 0
           return val

        count=0
        gun=0
        final_img = st.empty()
        while True:
           _, img = cap.read()
           height, width, channels = img.shape
           blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True,
crop=False)
           net.setInput(blob)
           outs = net.forward(output_layers)
```

```
class_ids = []
confidences = []
boxes = []
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)
            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)
indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
font = cv2.FONT_HERSHEY_PLAIN
print(indexes)
if indexes == 0:
    duration = 2 #second
    frequency = 500 #hz
```

```
    gun+=1
    print("weapon detected in frame")
    # winsound.Beep(frequency,duration)

    os.system('play -nq -t alsa synth {} sine {}'.format(duration, frequency))
    count+=1
    if count==1:
    #st.error("Alert, Weapon dedected in the video")
        call = client.calls.create(
                    url='http://demo.twilio.com/docs/voice.xml',
                    from_='+16194859230',
                    to='[+][91][8700549261]'
                )

        print(call.sid)
else:
    print("weapon not detected in a frame")
    #cv2.putText(img,"No Weapon", (50,50), font, 3,(255,0,255), 3)
for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        color = colors[class_ids[i]]
        cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
        cv2.putText(img, label, (x, y + 30), font, 3, color, 3)
```

```
            #cv2.imshow("Image", img)
            # cv2.namedWindow("output", cv2.WINDOW_NORMAL)
            # cv2.namedWindow('Image', cv2.WINDOW_NORMAL)
            # cv2.imshow("Image", ing)
            #st.image(img)
            final_img.image(img)
            key = cv2.waitKey(1)
            if key == 27:
                break
        cap.release()
        cv2.destroyAllWindows()
        if gun == 0:
                st.error(" No Gun found in video ")
        else:
        #st.image(img)
                st.success(" Gun dedected in video ")
```

Dockerfile for deploying application on AWS

```
FROM python:3.8.5
RUN apt-get update &&\
    apt-get install ffmpeg libsm6 libxext6 -y
EXPOSE 8501
WORKDIR /app
COPY requirements.txt ./requirements.txt
```

```
RUN pip3 install -r requirements.txt
COPY . .
CMD streamlit run app.py
```

docker-compose.yml file

```
version: '3'
services:
    web:
     build: .
     restart: always
     ports:
          - 80:8501
```