

CHAPTER 1

INTRODUCTION

Fire detection and prevention using machine learning by UAV (Unmanned Aerial Vehicle) involves the use of drones equipped with sensors and cameras to identify and alert authorities about potential fire outbreaks in remote areas. Machine learning algorithms are trained to analyze data collected by these sensors to identify signs of fire, such as heat signatures, smoke, and flames. The drones can also be equipped with firefighting equipment to help prevent fires from spreading. In addition to detecting and preventing fires, these UAVs can be used to monitor fire-prone areas and provide real-time information to firefighters to aid in their efforts to extinguish fires. The algorithms can be trained to differentiate between various types of heat sources and can even distinguish between controlled fires, such as campfires, and uncontrolled wildfires.

In addition to detection, UAVs can also be used for prevention by monitoring areas that are at high risk of fires, such as forests or grasslands. This information can then be used to deploy resources to prevent fires before they start. Once a fire has been detected, UAVs can be equipped with firefighting equipment, such as water tanks and fire-retardant sprayers, to help extinguish the fire before it spreads. This can be particularly useful in hard-to-reach areas, where traditional firefighting methods may not be feasible. Overall, the use of machine learning by UAVs for fire detection and prevention has the potential to significantly reduce the risk of wildfires and minimize the damage caused by them. With the ability to quickly detect fires, prevent them from spreading, and provide real-time information to firefighters, these technologies have the potential to save lives, protect property, and preserve the environment.

1.1 DOMAIN OVERVIEW

Learning algorithms work on the basis that strategies, algorithms, and inferences that worked well in the past are likely to continue working well in the future. These inferences can be obvious, such as "since the sun rose every morning for the last 10,000 days, it will probably rise tomorrow morning as well". The term machine learning was coined in 1959 by Arthur Samuel, an IBM employee and pioneer in the field of computer gaming and artificial intelligence. Self-learning as a machine learning paradigm was introduced in 1982 along with a neural network capable of self-learning named crossbar adaptive array (CAA). It is a learning with no external rewards and no external teacher advice. The CAA self-learning algorithm computes, in a crossbar fashion, both decisions about actions and emotions (feelings) about consequence situations. The system is driven by the interaction between cognition and emotion.

1.2 OVERVIEW OF THE PROJECT

To train the SSD algorithm on a large dataset of thermal images, including different types of fires and non-fire thermal signatures, to detect fires accurately. To evaluate the proposed system using real-world datasets and measure its detection rate and false alarm rate. To compare the proposed system with traditional methods of fire detection and demonstrate its effectiveness in reducing the response time and minimizing the damage caused by fire.

To compare the proposed system with traditional methods of fire detection and demonstrate its effectiveness in reducing the response time and minimizing the damage caused by fires. To explore the potential of the proposed system for various applications, such as public safety, industrial monitoring, and environmental monitoring.

1.3 OBJECTIVE OF THE PROJECT

To develop a reliable and accurate system that can detect and classify fire-related characteristics in real-time. The system will provide early warning to emergency responders and deliver firefighting agents or extinguishers to the location of the fire. The project will involve developing a user interface for monitoring the system's data and testing it in various real-world scenarios. The overall goal is to increase the safety of people and property while reducing the risk of loss due to fires.

CHAPTER 2

LITERATURE SURVEY

A Literature review is a text of a scholarly paper, which includes the current knowledge, including substantive findings, as well as theoretical and methodological contributions to a particular topic. Literature reviews use secondary sources and do not report new or original experimental work. A literature review usually precedes the methodology and results section.

2.1 REVIEW OF LITERATURE:

1. Title: Fire Detection using Deep Learning in UAV Imagery

Author: E. Ozan, M. Ozcelik, and Y. Kumbasar :

Year: 2021

This paper proposes a system for fire detection using deep learning in UAV imagery, specifically focusing on the use of the SSD algorithm for object detection. The proposed system was trained on a large dataset of thermal images and evaluated on a test set, showing promising results with an accuracy of over 90%. The authors suggest that the proposed system could be integrated with other systems for enhanced fire detection capabilities.

2. Title: Real-Time Fire Detection using Deep Learning on UAV's

Author: A.E. Del Pozo-Banos and M. Fernandez-Carrobles

Year: 2020

This paper proposes a real-time fire detection system using deep learning on UAVs, which utilizes the YOLO algorithm for object detection and compares its performance with the SSD algorithm. The proposed system was evaluated on a public dataset and showed high accuracy in detecting fires with a low false alarm rate. The authors suggest that the proposed system could be used in various applications, such as wildfire detection and monitoring.

2. Title: Fire Detection in Aerial Images using Deep Learning Techniques

Author: J. V. Prakash and K. Latha

Year: 2019

This paper proposes a fire detection system in aerial images using deep learning techniques, specifically utilizing the SSD algorithm for object detection. The proposed system was trained on a dataset of aerial images and evaluated on a test set, showing promising results with an accuracy of over 90%. The authors suggest that the proposed system could be used in various applications, such as detecting fires in forests and urban areas.

4. Title: Fire Detection in Wildland-Urban Interface using UAV's and Deep Learning

Author: J. H. Lim, J. H. Jeong, and H. R. Kim

Year: 2018

The authors propose a system for fire detection in the wildland-urban interface using UAVs and deep learning. They specifically use the SSD algorithm for object detection and train their proposed system on a dataset of UAV images.

They evaluate their proposed system on a separate test set, showing promising results in accurately detecting fires with a low false alarm rate. The authors suggest that their proposed system could be used in various applications, such as in monitoring large areas for fire outbreaks and in reducing the response time for fire suppression.

5. Title: Fire Detection using Unmanned Aerial Vehicles and Deep Learning

Author: M. A. Hanif, M. A. Naeem, and S. A. Malik

Year: 2017

The authors propose a system for fire detection using unmanned aerial vehicles (UAVs) and deep learning. They use the SSD algorithm for object detection and train their proposed system on a dataset of UAV images. They evaluate their proposed system on a separate test set, showing high accuracy in detecting fires with a low false alarm rate. The authors suggest that their proposed system could be used in various applications, such as in monitoring forests, detecting fires in urban areas, and in disaster management. They also suggest that their proposed system could be used to reduce the response time for fire suppression and to minimize the risk of firefighter injuries.

2.2 PROBLEM STATEMENT

- The first problem statement is to improve the speed and accuracy of fire detection using machine learning algorithms. Traditional fire detection methods rely on human intervention and are often prone to errors, delays, and false alarms.
- The second problem statement is to develop a system that can detect fires in complex and cluttered environments using machine learning algorithms. In many scenarios, fires can be hidden or obscured by smoke, dust, or other obstacles, making it difficult to detect them using traditional methods. The project aims to develop a system that can accurately identify fires in these challenging environments and provide timely alerts to emergency services.
- The third problem statement is to develop a system that can detect fires in remote or inaccessible areas using machine learning algorithms.

In many cases, fires can occur in areas that are difficult to access or monitor, such as forests, mountains, or offshore oil rigs. The project aims to develop a system that can detect fires in these areas using drones equipped with cameras and machine learning algorithms, allowing emergency services to respond quickly and effectively.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The existing system for Fire Detection using Machine Learning by drone typically involves using a drone equipped with a camera to capture images or videos of a designated area. The images are then analyzed using machine learning algorithms to identify the presence of flames, smoke, or other indicators of fire.

One of the most used machine learning algorithms for fire detection is the Convolutional Neural Network (CNN). The CNN model is trained on a large dataset of images containing fire and non-fire objects to learn the features that distinguish between them. Once the model is trained, it can be used to detect fires in real-time by analyzing the images captured by the drone's camera.

Another commonly used algorithm for fire detection is the Support Vector Machine (SVM). The SVM model uses a similar approach to CNN, but instead of learning the features directly from the images, it separates the images into two categories (fire and non-fire) based on a set of pre-defined features.

3.1.1 DISADVANTAGES

- **Limited accuracy:** While machine learning algorithms have shown promising results in detecting fires, they are not always accurate.
- **Dependence on environmental factors:** The accuracy of fire detection can be affected by environmental factors such as lighting, weather conditions, and the quality of the camera used. This can lead to missed detections or false alarms.
- **Limited coverage area:** Drones have a limited range and battery life, which can limit the area that can be monitored for fires. This can make it difficult to detect fires in remote or inaccessible or cover large geographical areas.

3.2 PROPOSED SYSTEM

Drone: The drone would be equipped with a camera, sensors, and a communication module. It would fly over the area to be monitored, collecting visual and thermal data.

Machine learning algorithms: The data collected by the drone would be fed into machine learning algorithms, which would be trained to detect the presence of fires based on patterns in the data. The algorithms could be trained using historical fire data and data collected by the drone during training flights.

Fire detection: Once the machine learning algorithms detect the presence of a fire, the drone would alert a central monitoring system and send real-time data about the fire's location, size, and intensity. The monitoring system could then notify authorities and emergency services to act.

Fire prevention: In addition to fire detection, the drone could also be equipped with fire prevention measures, such as spraying water or fire-retardant chemicals on the fire to contain its spread.

3.2.1 ADVANTAGES

- **Early Detection:** The use of machine learning algorithms enables early detection of fires before they become large, spreading disasters. This early detection can enable quicker response times and more effective firefighting efforts.
- **Improved Accuracy:** Machine learning algorithms can analyze data collected by drones with a higher degree of accuracy and speed than human operators, increasing the accuracy of fire detection.
- **Safety:** Drones can be used to monitor dangerous areas, reducing the need for human firefighters to enter dangerous areas. This can help reduce the risk of injury or death for firefighters.

CHAPTER 4

REQUIREMENT ANALYSIS

Requirement analysis determines the requirements of a new system. This project analyzes product and resource requirements, which is required for this successful system.

The product requirement includes input and output requirements it gives the want in terms of input to produce the required output. The resource requirements give in brief about the software and hardware that are needed to achieve the required functionality.

4.1 FUNCTIONAL REQUIREMENTS

A Functional requirement defines the function of a system or its components. A function is described as the set of inputs, the behavior, and the outputs. Functional requirements may be calculations, technical details, data manipulation and other specific functionalities that show how a use case is to be fulfilled. They are supported by non-functionalities requirements, which impose constraints on the design or implementation.

4.2 NON-FUNCTIONAL REQUIREMENTS

Non-Functional Requirements are requirements that specify criteria that can be used to judge the operations of a system.

Rather than specific behaviors, Non-functional requirements are often called qualities of the system. The non-functional requirements in this system are:

1. The system should be accurate and efficient.
2. The system should be able to meet all user requirements.

4.3 HARDWARE REQUIREMENTS

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware.

A hardware requirements list is often accompanied by a hardware compatibility list, especially in the case of operating systems.

The minimal hardware requirements are as follows,

1. NVIDIA Jetson Nano
2. Camera
3. SD Card
4. Power Adapter System
5. Processor: AMD Ryzen 5
6. RAM: 4 GB
7. Processor speed: 2.10 GHZ
8. Main Memory: 8GB RAM
9. Hard Disk Drive: 64 GB
10. Display (Monitor)
11. Spraying System

1. NVIDIA JETSON NANO

NVIDIA Jetson Nano is a small, powerful computer designed for embedded applications and edge computing. It features a quad-core ARM Cortex-A57 CPU and an NVIDIA Maxwell GPU with 128 CUDA cores, as well as 4GB of LPDDR4 memory and support for multiple cameras and sensors. The Jetson Nano is optimized for running deep learning and AI applications, It also includes built-in hardware acceleration for computer vision tasks like object detection and recognition. With its compact size and low power consumption, the Jetson Nano is well-suited for a wide range of applications, from autonomous vehicles and drones to robotics and smart cameras.

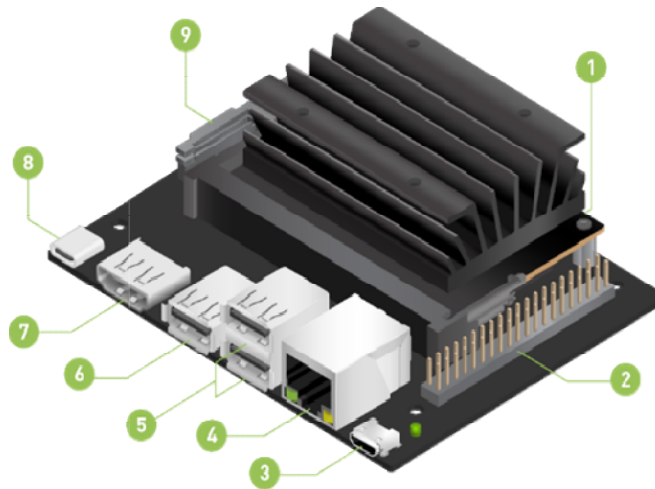


Fig.4.1 NVIDIA Jetson Nano

1. Micro SD Card Slot for main storage
2. 40-pin expansion header
3. Micro-USB Port for Device Mode
4. Gigabit Ethernet Port
5. USB 2.0 Ports(X2)
6. USB 3.0 Ports(X1)
7. HDMI Output Port
8. USB-C for 5V Power Input
9. MIPI CSI-2 Camera Connector

2. CAMERA MODULE

Camera module is an image sensor integrated with a lens, control electronics, and interface like CSI, Ethernet or plain raw low-voltage differential signaling.

3. SD CARD MODULE:

SD Card Module is a breakout board used for SD card processes such as reading and writing with a microcontroller. The board is compatible with microcontroller systems like Arduino. A standard SD card can be directly inserted into the board, but to use microSD cards, you need to use an adapter.

4. POWER ADAPTER:

The power adapter serves the purpose of converting AC voltage to a single DC voltage for your computer. It operates as an external battery for your computer so that the computer's size does not need to be so large.

5. SPRAYING SYSTEM

The spraying system for a drone typically includes a tank or container for the liquid or chemical to be sprayed, a pump to propel the liquid out of the tank, a nozzle to create a spray pattern, and a control system to manage the flow and distribution of the liquid

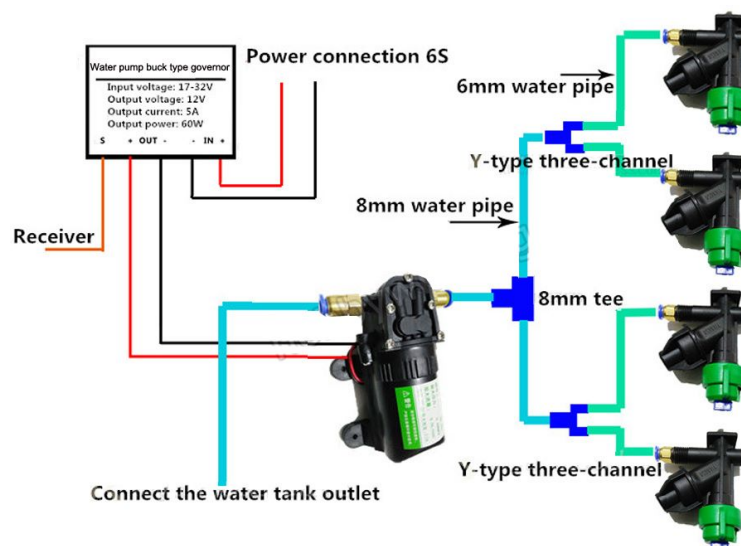


Fig. 4.2 Spraying system

4.4 SOFTWARE REQUIREMENTS

Software requirements deal with defining resource requirements and prerequisites that need to be installed on a computer to provide the functioning of an application. The minimal software requirements are as follows,

1. Programming Language: Python
2. IDE: PyCharm, Python
3. Operating System: Linux
4. Algorithm: SSD Model

1. Python:

Python is a widely-used general-purpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python is a high-level, general-purpose, and very popular programming language. Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting-edge technology in the Software Industry.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

2. Linux:

Linux was originally developed for personal computers based on the Intel x86 architecture, but has since been ported to more platforms than any other operating system. Because of the dominance of the Linux-based Android on smartphones, Linux, including Android, has the largest installed base of all general-purpose operating systems, as of May 2022. Although Linux is, as of November 2022, used by only around 2.6 percent of desktop computers, the Chromebook, which runs the Linux kernel-based ChromeOS, dominates the US K–12 education market and represents nearly 20 percent of sub-\$300 notebook sales in the US.

Linux is a free, open-source operating system, released under the GN General Public License (GPL). Anyone can run, study, modify, and redistribute the source code, or even sell copies of their modified code, as long as they do so under the same license Every version of the Linux OS manages hardware resources, launches and handles applications, and provides some form of user interface. The enormous community for developers and wide range of distributions means that a Linux version is available for almost any task, and Linux has penetrated.

CHAPTER 5

DESIGN ENGINEERING

5.1 ARCHITECTURE DIAGRAM

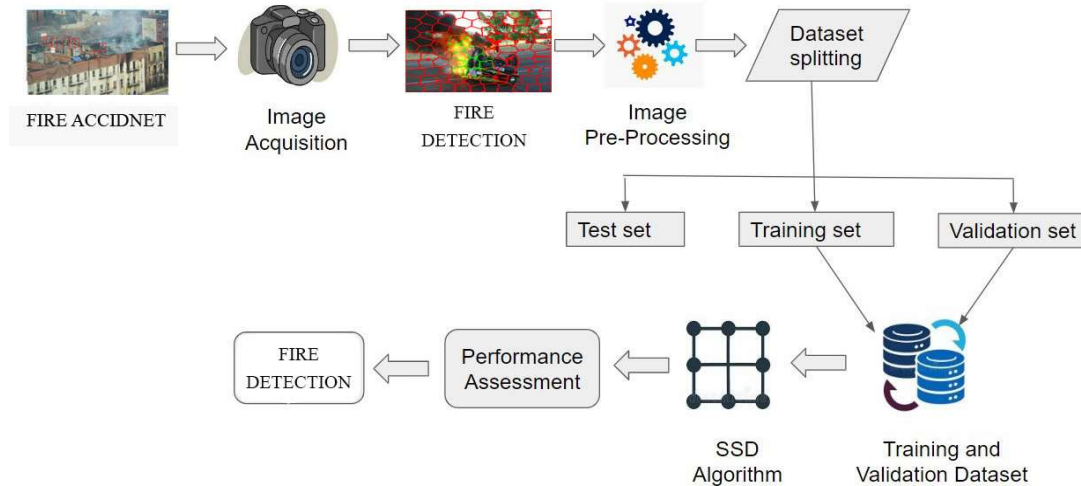


Fig. 5.1 Architecture Diagram

An architecture diagram is a visual representation of a system's components and their relationships, depicted using shapes and lines. It provides a high-level overview of the system's architecture, including external systems or resources, security controls, and performance metrics. The diagram's specific format and details vary based on the system's complexity and requirements.

5.2 DATA FLOW DIAGRAM

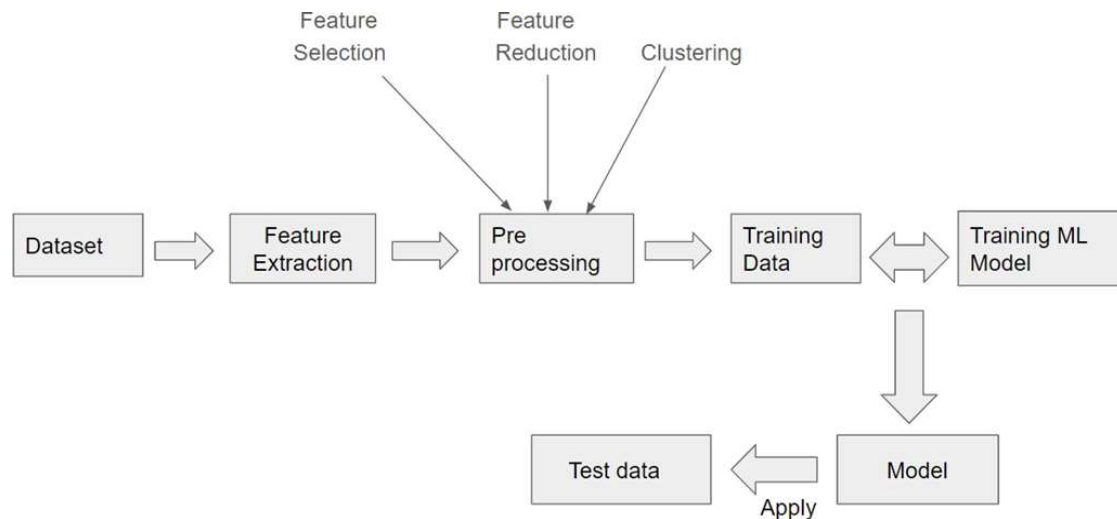


Fig. 5.2 Data Flow Diagram

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

5.3 USE CASE DIAGRAM

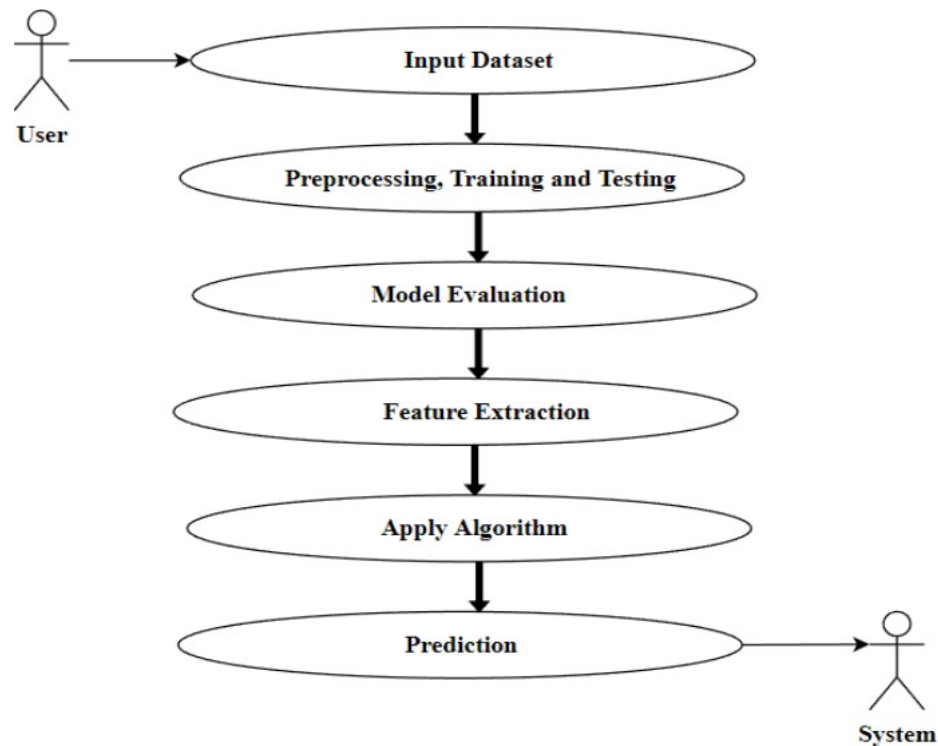


Fig. 5.3 Use Case Diagram

Use case diagrams overview the usage requirement for system. They are useful for presentations to management and project stakeholders, but for actual development use cases provide significantly more value because they describe the meant of the actual requirements. A use case describes a sequence of action that provides something of measurable value to an action.

CHAPTER 6

MODULES DESCRIPTION

6.1 IMPORTING NECESSARY LIBRARIES

A library is a collection of functions that can be added to your Python code and called as necessary, just like any other function. There is no reason to rewrite code that will perform a standard task. With libraries, you can import pre-existing functions and efficiently expand the functionality of your code.

6.2 DATA PREPROCESSING

Analyzing data sets to summarize and visualize data properties is the main area of business. Before considering any inferences, listen to the data by examining all variables in the data.

This EDA process specifically focuses on the following Aspects;

1. Understanding characters of data
2. Finding meaningful patterns in data
3. Post modeling strategies
4. Debugging strategies
5. Visualization of results

6.3 MODEL DEVELOPMENT

Model Development ML Model Lifecycle refers to the process that covers right from source data identification to model development, model deployment and model maintenance. At a high level, the entire activities fall under two broad categories, such as ML Model Development and ML Model Operations.

6.4 MODEL EVALUATION

In machine learning, model validation is alluded to as the procedure where a trained model is assessed with a testing data set. The testing data set is a different bit of similar data set from which the training set is inferred. The principal for utilizing the testing data set is to test the speculation capacity of a prepared model.

By using an evaluation data source and an ML model, the creation of an evaluation and review of the results of the evaluation can be viewed. When creating and training an ML model, the goal is to select the model that makes the best predictions, which means selecting the model with the best settings (ML model settings or hyperparameters).

6.5 MODEL DEPLOYMENT

Once a user has found and tuned a viable model of his problem it is time to make use of that model. Users may need to revisit that reason and remind themselves in what form they need a solution for the problem that it is important to solve. The problem is not addressed until the user does something with the results.

In this presentation of results, it is important to learn tactics for presenting user results in answer to a question and considerations when turning a prototype model into a production system. Depending on the type of problem the user is trying to solve, the presentation of results will be very different.

6.6 SSD ALGORITHM

SSD (Single shot multi box Detector) is a popular algorithm in object detection. SSD divides the image using a grid and has each grid cell be responsible for detecting objects in that region of the image. Image classification in computer vision takes an image and predicts the object in an image, while object detection not only predicts the object but also finds their location in terms of bounding boxes for illustrative purpose, assuming there is at most one class and one object in an image, the output of an object detection model should include:

- Probability that there is an object,
- Height and width of the bounding box,
- Horizontal coordinate of the center point of the bounding box,
- Vertical coordinate of the center point of the bounding box.

CHAPTER 7

DRONE TECHNOLOGY

7.1 INTRODUCTION

With the development of the drone's technical capabilities, there is a growing range and areas of uses in which the drone is already in use, or is planned to be used in the future. Since the use of drones poses a great hazard to aircraft operations, due to the shared use of airspace, an aspect that is given particular attention is the safety aspect of drone use. Because the use of drones is planned in the airspace where many aircraft appear, the tendency to use drones at airports for inspection of contamination of runways represents a huge challenge from a safety perspective.

With the development of the drone's technical capabilities, there is a growing range and areas of uses in which the drone is already in use, or is planned to be used in the future.

Since the use of drones poses a great hazard to aircraft operations, due to the shared use of airspace, an aspect that is given particular attention is the safety aspect of drone use. Because the use of drones is planned in the airspace where many aircraft appear.

One of the benefits of drone usage for condition inspections include availability of high-resolution photographs that can be documented. Runway Inspections Using Drone - Safety Issues and Associated Risks This provides an opportunity to improve many of the existing runway contamination activities and procedures.

Inspections of airport surfaces by using drone are very effective, but they require careful planning of each of the following steps:

- Definition of the area to be inspected
- On-site drone inspections

The idea of this innovative method is that there is an operational center behind the drone operations, with technical support, from which the controller operates the drone, and communicates with the necessary services: Air Traffic Control, airport services, etc. Technical support also monitors video record from drone cameras on the appropriate screens and, if necessary, stops the drone at specific parts of the runway to identify the presence of contamination on the surface.

7.2 WORKING PRINCIPLE

1. Any aircraft or flying machine operated without a human pilot such machines is called an unmanned aerial vehicle (UAV). It can be guided autonomously or remotely by a human operator using onboard computers and robots.
2. During surveillance or military operation, UAVs can be a part of an unmanned aircraft system (UAS), Drones are separately for air and water
3. Drones have become increasingly popular in recent years. They are used for a variety of purposes, including photography, videography, surveying, inspection, and even delivery.

7.3 COMPONENTS

The basic components of a drone are the frame, motors, propellers, battery, flight controller, and sensors. Let us take a closer look at each of these components.

- Frame
- BLDC motor
- ESC
- Flight controllers
- Battery
- Propellers
- Transmitter & receiver

Types of drones based on the number of Propellers:

- Bicopter (2 propellers)
- Triplecopter (3 propellers)
- Quadcopter (4 propellers)
- Hexacopter (6 propellers)
- Octocopter (8 propellers)

1. **Throttle/ Hover:** up and down movement of the drone is called throttle

- If all four propellers run at normal speed, then the drone will move down.
- If all four propellers run at a higher speed, then the drone will move up. This is called the hovering of a drone

2. **Pitch:** movement of a drone about a lateral axis (either forward or backward) is called pitching motion

- If two rear propellers run at high speed, then the drone will move in a forwarding direction.
- If two front propellers run at high speed, then the drone will move in the backward direction.

3. **Roll:** movement of a drone about the longitudinal axis is called rolling motion

- If two right propellers run at high speed, then the drone will move in the left direction
- If two left propellers run at high speed, then the drone will move in the right Direction

4. **Yawn:** the rotation of the head of the drone about the vertical axis (either the left or right) is called Yawning motion

- If two propellers of a right diagonal run at high speed, then the drone will rotate in an anti-clockwise direction

- If two propellers of a left diagonal run at high speed, then the drone will rotate in a clockwise direction.

The basic components of a drone are the frame, motors, propellers, battery, flight controller, and sensors. Let us take a closer look at each of these components.

1. Frame
2. BLDC motor
3. ESC
4. Flight controllers
5. Battery
6. Propellers
7. Transmitter & receiver

1. Frame

The drone frame is like the foundation of the drone. It's the structure that keeps everything in place, like the motors, propellers, and all that other gear. It's also like a little fortress for sensitive electronics and other equipment inside, It keeps everything safe and secure.

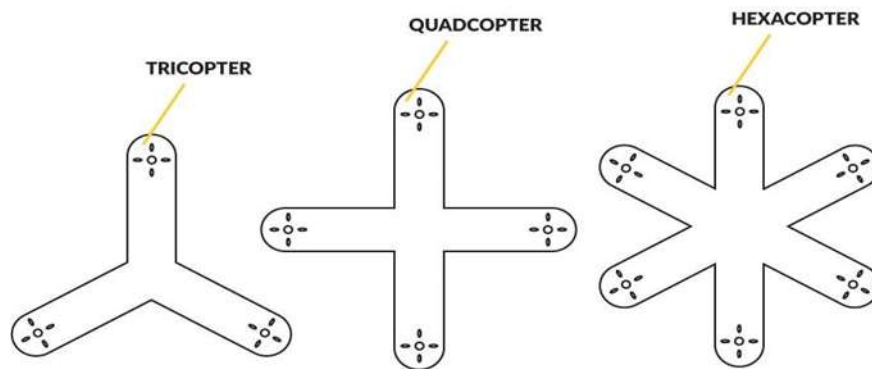


Fig. 7.1 Drone Frames

2. BLDC motor

A brushless DC (BLDC) motor is a type of electric motor that relies on repulsive and attractive forces between permanent magnets and electromagnets to drive its

rotation. Due to the lack of friction, BLDC motors require less maintenance and have a longer lifespan. They are more efficient than brushed motors and have a higher torque to weight ratio. Many DC motors use brushes to transfer current from the commutator to the rotor.



Fig. 7.2 Brushless DC Motor 1100kv

3. ESC:

Electronic speed controllers (ESCs) are devices that allow drone flight controllers to control and adjust the speed of the aircraft's electric motors. A signal from the flight controller causes the ESC to raise or lower the voltage to the motor as required, thus changing the speed of the propeller.



Fig. 7.3 Electronic Speed Controllers

4. Battery

The most common batteries used in drones are lithium polymer (LiPo) batteries. LiPo batteries are composed of a lithium-based cathode and anode separated by a polymer electrolyte.

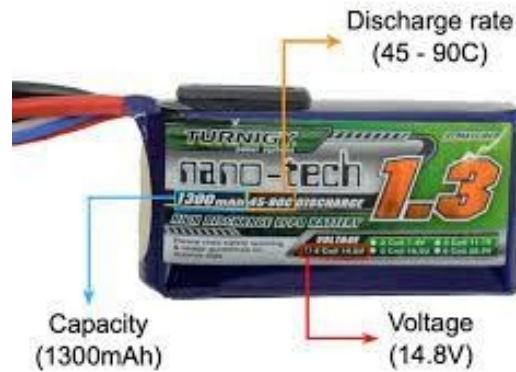


Fig. 7.4 Lithium Polymer (Lipo) Batteries

5. Propellers

Propellers for Drones and UAVs. Propellers are devices that transform rotary motion into linear thrust. Drone propellers provide lift for the aircraft by spinning and creating an airfoil, which results in a pressure difference between the top and bottom surfaces of the propeller.



Fig. 7.5 Propellers

6. Transmitter & receiver

The Transmitter is an electronic device that uses radio signals to transmit commands wirelessly via a set radio frequency over to the Radio Receiver, which is connected to an aircraft or multirotor being remotely controlled.

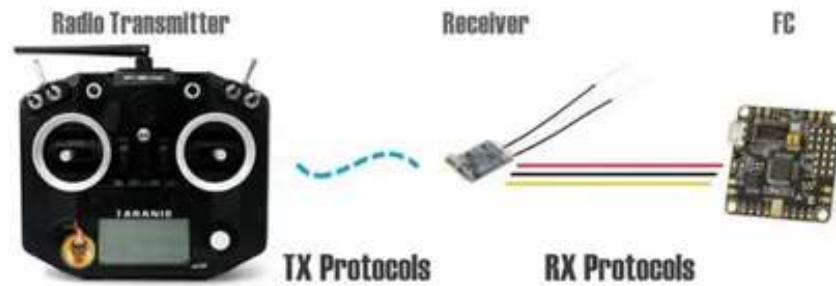


Fig. 7.6 Transmitter & receiver

7.4 CLASSIFICATION OF DRONE

Drones are classified based on the number of rotors, shape of the frame, and airframe. Drones could also be classified based on the degree of autonomy in their flight operations. ICAO classifies uncrewed aircraft as either remotely piloted aircraft or fully autonomous. Some UAVs offer intermediate degrees of autonomy.

Types of drones based on airframe:

1. Fixed wings
2. Multi-copter
3. Hybrid

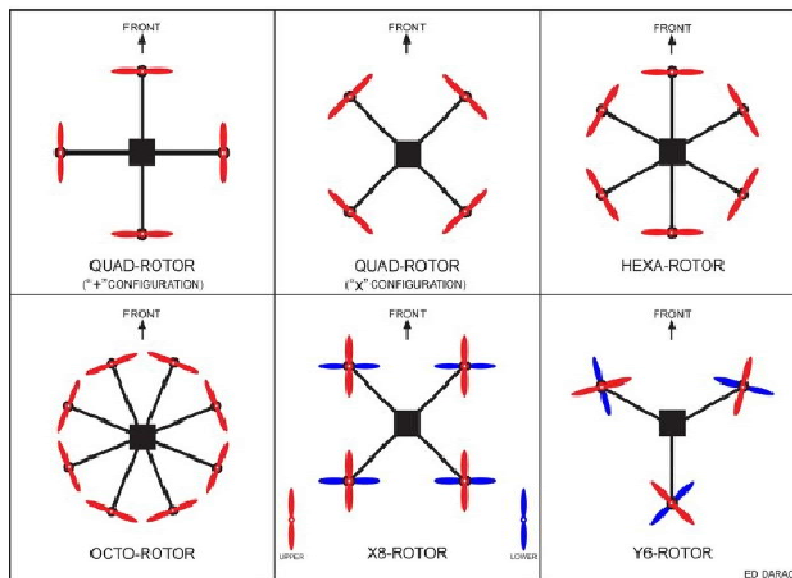


Fig.7.7 Classification of Drone Based on Arrangement of Rotors

7.5 FLIGHT CONTROLLER

The flight controller is the brain of the drone, which controls the motors and ESCs in the drone. It is an electronics board in which sensors, processors, communication protocols, and transmitter pins are installed.

A flight controller controls every aspect of the drone. It moves the drone by changing the motors' RPM. The Flight Controller oversees the safety of flight control operations by reviewing flight data and flight control systems.

He/She resolves issues during real-time flight control operations. He/She leads investigation into operational issues and recovers the flight networks.

Microcontroller:

KK 2.1. 5 flight controller board: Atmel Mega 644PA 8-bit AVR.

Ardupilot APM 2.8 flight controller board: ATMEGA328P.

Open Pilot CC3D flight controller: STM32F4 32-bit series.

Ardupilot APM 2.6 flight controller: ATMEGA2560 and ATMEGA32U-2.



Fig.7.8 APM2.8 Flight Controller with In-Built Compass for Drone

The flight controller uses the data gathered by the sensors to calculate the desired speed for each of the four motors. The flight controller sends this desired speed to the Electronic Speed Controllers (ESC's), which translates this desired speed into a signal that the motors can understand.

7.5.1 Pixhawk flight controller

Pixhawk is the world's most famous open-source flight control hardware manufacturer 3DR launched open-source flight control. Pixhawk has an open-source hardware and has powerful features, reliable performance has been favored by most users. After doing some specification research about flight controllers, we decided that Pixhawk satisfies our project needs.



Fig.7.9 Pixhawk 32-bit ARM controller

1. Features

1. The advanced 32-bit ARM CortexM4 high-performance processors
2. 14 PWM / servo output
3. Bus interface (UART, I2C, SPI, CAN)
4. The integrated backup power and backup controller fails, the primary controller fails over to the backup control is safe
5. Provide automatic and manual modes
6. Provide redundant power input and failover
7. Multi color LED lights
8. Provide multi-tone buzzer Interface
9. Micro SD recording flight data

CHAPTER 8

IMPLEMENTATION

STEP 1: FLASH NVIDIA's JETSON NANO DEVELOPER KIT.img TO A MICROSD FOR JETSON NANO

In this step, we will download NVIDIA's Jetpack 4.2 Ubuntu-based OS image and flash it to a microSD. download the "Jetson Nano Developer Kit SD Card image download and install balenaEtcher, a disk image flashing tool You will need a suitable microSD card and microSD reader hardware. We recommend either a 32GB or 64GB microSD card (SanDisk's 98MB/s cards are high quality, and Amazon carries them if they are a distributor in your locale).

Any microSD card reader should work. Insert the microSD into the card reader, and then plug the card reader into a USB port on your computer. From there, fire up the balenaEtcher and proceed to flash.

STEP 2: BOOT OUR JETSON NANO WITH THE MICROSD AND CONNECT TO THE NETWORK

In this step, we will power up our Jetson Nano and establish network connectivity. This step requires the following:

1. The flashed microSD from Step #1
2. An NVIDIA Jetson Nano dev board
3. HDMI screen
4. USB keyboard + mouse

5. A power supply — either (1) a 5V 2.5A (12.5W) microSD power supply or (2) a 5V 4A (20W) barrel plug power supply with a jumper at the J48 connector.
6. Network connection — either (1) an Ethernet cable connecting your Nano to your network or (2) a wireless module. The wireless module can come in the form of a USB WiFi adapter or a WiFi module installed under the Jetson Nano heatsink.

STEP 3: INSTALL SYSEM-LEVEL SYSTEM-LEVEL DEPENDENCIES

The first set of software we need to install includes a selection of development tools:

```
$ sudo apt-get install git cmake
```

```
$ sudo apt-get install libatlas-base-dev gfortran
```

```
$ sudo apt-get install libhdf5-serial-dev hdf5-tools
```

```
$ sudo apt-get install python3-dev
```

```
$ sudo apt-get install nano locate
```

Next, we'll install SciPy prerequisites (gathered from **NVIDIA's devtalk forums**) and a system-level **Cython** library:

```
$ sudo apt-get install libfreetype6-dev python3-setuptools
```

```
$ sudo apt-get install protobuf-compiler libprotobuf-dev openssl
```

```
$ sudo apt-get install libssl-dev libcurl4-openssl-dev
```

```
$ sudo apt-get install cython3
```

STEP 4: UPDATE CMAKE

Now we'll update the CMake precompiler tool as we need a newer version in order to successfully compile OpenCV.

First, download and extract the CMake update:

```
$ wget http://www.cmake.org/files/v3.13/cmake-3.13.0.tar.gz
```

```
$ tar xpvf cmake-3.13.0.tar.gz cmake-3.13.0/
```

Next, compile CMake:

```
$ cd cmake-3.13.0/
```

```
$ ./bootstrap --system-curl
```

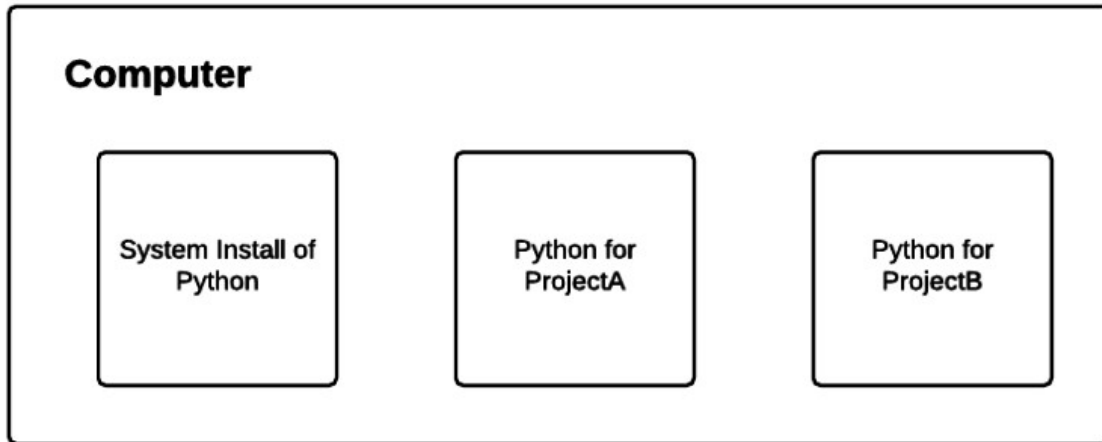
```
$ make -j4
```

And finally, update bash profile:

```
$ echo 'export PATH=/home/nvidia/cmake-3.13.0/bin/:$PATH' >> ~/.bashrc
```

```
$ source ~/.bashrc
```

STEP 5: SET UP PYTHON VIRTUAL ENVIRONMENTS IN JETSON NANO



First, we will install the *de facto* Python package management tool, pip:

```
$ wget https://bootstrap.pypa.io/get-pip.py
```

```
$ sudo python3 get-pip.py
```

```
$ rm get-pip.py
```

And then we'll install my favorite tools for managing virtual environments, virtualenv and virtualenvwrapper:

```
$ sudo pip install virtualenv virtualenvwrapper
```

The virtualenvwrapper tool is not fully installed until you add information to your bash profile. Go ahead and open up your `~/.bashrc` with the nano editor:

```
$ nano ~/.bashrc
```

And then insert the following at the bottom of the file:

```
# virtualenv and virtualenvwrapper
```

```
export WORKON_HOME=$HOME/.virtualenvs
```

```
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
```

```
source /usr/local/bin/virtualenvwrapper.sh
```

```
pyimagesearch@pyimagesearch-nano: ~  
pyimagesearch@pyimagesearch-nano:~$ mkvirtualenv py3cv4 -p python3  
created virtual environment CPython3.6.9.final.0-64 in 579ms  
creator CPython3Posix(dest=/home/pyimagesearch/.virtualenvs/py3cv4, clear=False,  
e, global=False)  
seeder FromAppData(download=False, pip=latest, setuptools=latest, wheel=latest  
, via=copy, app_data_dir=/home/pyimagesearch/.local/share/virtualenv/seed-app-da  
ta/v1)  
activators BashActivator,CShellActivator,FishActivator,PowerShellActivator,Pyt  
honActivator,XonshActivator  
virtualenvwrapper.user_scripts creating /home/pyimagesearch/.virtualenvs/py3cv4/  
bin/postdeactivate  
virtualenvwrapper.user_scripts creating /home/pyimagesearch/.virtualenvs/py3cv4/  
bin/postactivate  
virtualenvwrapper.user_scripts creating /home/pyimagesearch/.virtualenvs/py3cv4/  
bin/preactivate  
virtualenvwrapper.user_scripts creating /home/pyimagesearch/.virtualenvs/py3cv4/  
bin/postactivate  
virtualenvwrapper.user_scripts creating /home/pyimagesearch/.virtualenvs/py3cv4/  
bin/get_env_details  
(py3cv4) pyimagesearch@pyimagesearch-nano:~$
```

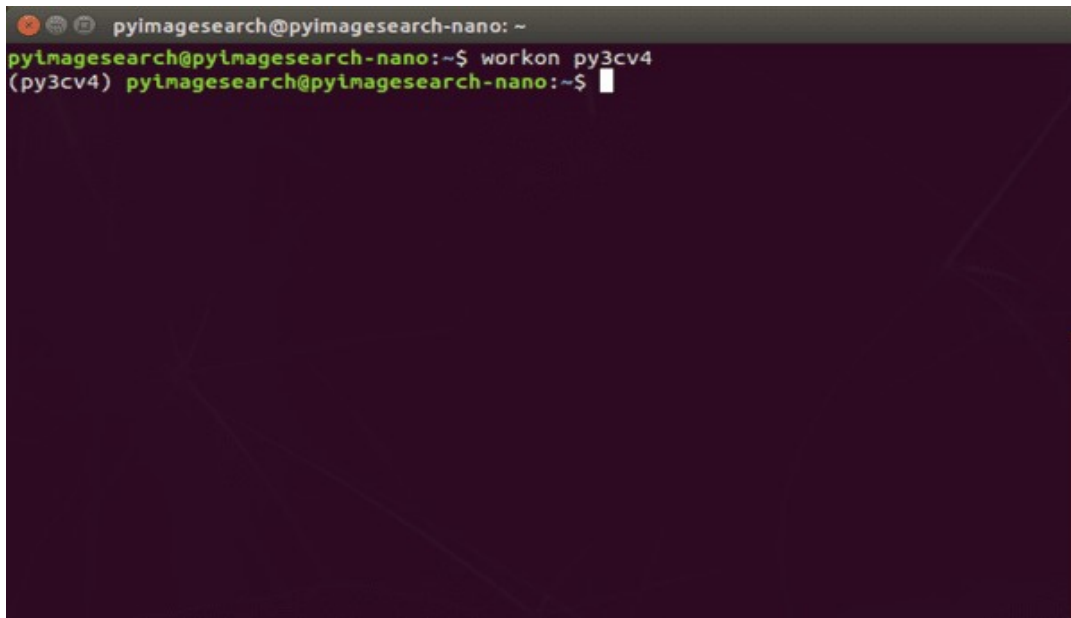
Fig. 8.1 Set Up Python Virtual Environment

STEP 6: CREATE YOUR ‘py3cv4’ VIRTUAL ENVIRONMENT

This step is dead simple once you have installed virtualenv and virtualenvwrapper in the previous step. The virtualenvwrapper tool provides the following commands to work with virtual environments:

- **mkvirtualenv**: Create a Python virtual environment
- **lsvirtualenv**: List virtual environments installed on your system
- **rmvirtualenv**: Remove a virtual environment
- **workon**: Activate a Python virtual environment
- **deactivate**: Exits the virtual environment taking you back to your system environment.

When our environment is ready, our Bash prompt will be preceded by (py3cv4). If prompt is not preceded by the name of virtual environment name, at any time you can use the **workon** command as follows: `$ workon py3cv4`

A terminal window with a dark purple background. The title bar shows 'pyimagesearch@pyimagesearch-nano: ~'. The prompt is 'pyimagesearch@pyimagesearch-nano:~\$'. The user enters 'workon py3cv4'. The prompt changes to '(py3cv4) pyimagesearch@pyimagesearch-nano:~\$' with a white cursor at the end.

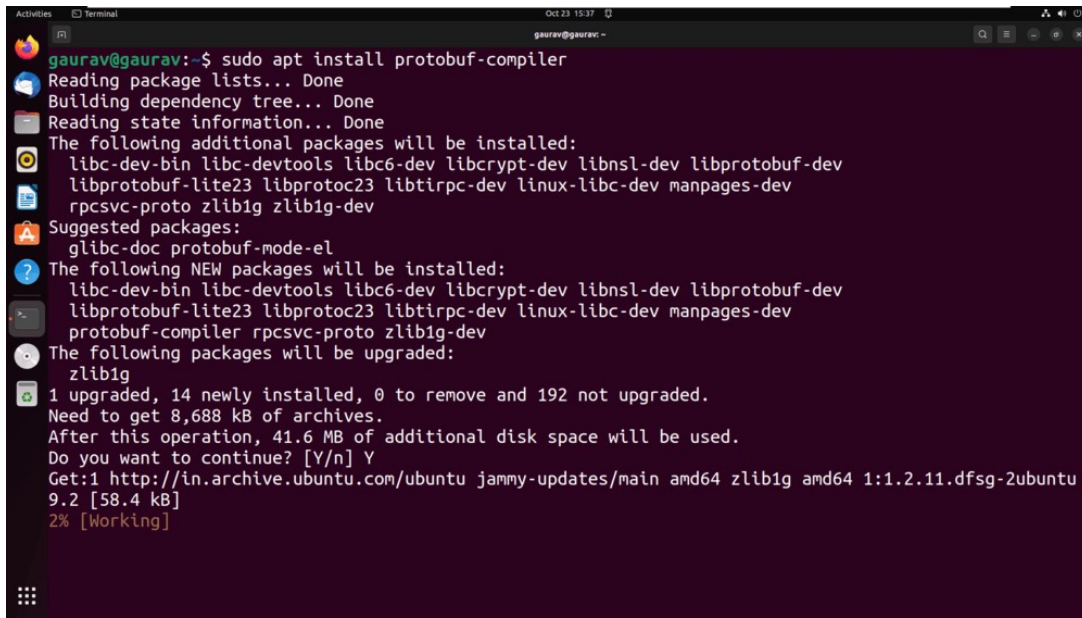
```
pyimagesearch@pyimagesearch-nano: ~  
pyimagesearch@pyimagesearch-nano:~$ workon py3cv4  
(py3cv4) pyimagesearch@pyimagesearch-nano:~$
```

Fig. 8.2 Creating Virtual Environment For py3cv4

STEP 7: INSTALL THE PROTOBUF COMPILER

First, download and install an efficient implementation of the protobuf compiler (source):

```
$ wget https://raw.githubusercontent.com/jkjung  
avt/jetson_nano/master/install_protobuf-3.6.1.Sh  
$ sudo chmod +x install_protobuf-3.6.1.sh  
$ ./install_protobuf-3.6.1.sh
```


A terminal window showing the command 'sudo apt install protobuf-compiler' and its output. The output lists additional packages to be installed, suggested packages, new packages to be installed, and packages to be upgraded. It also shows the disk space requirements and the progress of downloading the zlib1g package.

```
gaurav@gaurav:~$ sudo apt install protobuf-compiler
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libc-dev-bin libc-devtools libc6-dev libcrypt-dev libnsl-dev libprotobuf-dev
  libprotobuf-lite23 libprotoc23 libtirpc-dev linux-libc-dev manpages-dev
  rpcsvc-proto zlib1g zlib1g-dev
Suggested packages:
  glibc-doc protobuf-mode-el
The following NEW packages will be installed:
  libc-dev-bin libc-devtools libc6-dev libcrypt-dev libnsl-dev libprotobuf-dev
  libprotobuf-lite23 libprotoc23 libtirpc-dev linux-libc-dev manpages-dev
  protobuf-compiler rpcsvc-proto zlib1g-dev
The following packages will be upgraded:
  zlib1g
1 upgraded, 14 newly installed, 0 to remove and 192 not upgraded.
Need to get 8,688 kB of archives.
After this operation, 41.6 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://in.archive.ubuntu.com/ubuntu jammy-updates/main amd64 zlib1g amd64 1:1.2.11.dfsg-2ubuntu
9.2 [58.4 kB]
2% [Working]
```

Fig. 8.3 Installing the Protobuf Compiler

STEP 8: INSTALL TENSORFLOW, KERAS, NUMPY, AND SCIPY ON JETSON NANO

In this section, we'll install TensorFlow/Keras and their dependencies.

First, ensure you're in the virtual environment:

```
$ workon py3cv4
```

And then install NumPy and Cython:

```
$ pip install numpy cython
```

```
$ mv opencv_contrib-4.1.2 opencv_contrib
```

And change into the OpenCV directory, followed by creating and entering a build directory:

```
$ cd opencv
```

```
$ mkdir build
```

```
$ cd build
```

```
$ cd ~
```

```
$ workon py3cv4
```

```
File Edit View Search Terminal Help
torchvision-0.10.0a0+300a8a4-cp36-cp36m-linux_aarch64.whl
jetson@nano:~/vision/dist$ sudo -H pip3 install torchvision-0.10.0a0+300a8a4-cp3
6-cp36m-linux_aarch64.whl
Processing ./torchvision-0.10.0a0+300a8a4-cp36-cp36m-linux_aarch64.whl
Requirement already satisfied: numpy in /root/.local/lib/python3.6/site-packages
(from torchvision==0.10.0a0+300a8a4)
Requirement already satisfied: torch in /usr/local/lib/python3.6/dist-packages (
from torchvision==0.10.0a0+300a8a4)
Requirement already satisfied: pillow>=5.3.0 in /root/.local/lib/python3.6/site-
packages (from torchvision==0.10.0a0+300a8a4)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.6/dis
t-packages (from torch->torchvision==0.10.0a0+300a8a4)
Requirement already satisfied: dataclasses; python_version < "3.7" in /usr/local
/lib/python3.6/dist-packages (from torch->torchvision==0.10.0a0+300a8a4)
Installing collected packages: torchvision
Successfully installed torchvision-0.10.0a0+300a8a4
jetson@nano:~/vision/dist$ python3
Python 3.6.9 (default, Oct 8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torchvision as tv
>>> tv.__version__
'0.10.0a0+300a8a4'
>>>
```

Fig. 8.4 Installing Install Tensorflow/Keras And Their Dependencies

STEP 9: INSTALL THE TENSORFLOW OBJECT DETECTION API ON JETSON NANO

In this step, we will install the TFOD API on our Jetson Nano. TensorFlow's Object Detection API (TFOD API) is a library that we typically know for developing object detection models.

We also need it to optimize models for the Nano's GPU. NVIDIA's `tf_trt_models` is a wrapper around the TFOD API, which allows for building frozen graphs, a necessary for model deployment.

More information on `tf_trt_models` can be found in this NVIDIA repository.

Again, ensure that all actions take place "in" your `py3cv4` virtual environment:

```
$ cd ~
```

```
$ workon py3cv4
```

First, clone the models repository from TensorFlow

```
$ git clone https://github.com/tensorflow/models
```

In order to be reproducible, you should checkout the following commit that supports TensorFlow 1.13.1:

```
$ cd models && git checkout -q b00783d
```

From there, install the COCO API for working with the COCO dataset and, in particular, object detection:

```
$ cd ~
```

```
$ git clone https://github.com/cocodataset/cocoapi.git
```

```
$ cd cocoapi/PythonAPI
```

```
$ python setup.py install
```

STEP 10: INSTALL OPENCV 4.1.2 ON JETSON NANO

In this section, we will install the OpenCV library with CUDA support on our Jetson Nano.

OpenCV is the common library we use for image processing, deep learning via the DNN module, and basic display tasks. I've created an OpenCV Tutorial for you if you're interested in learning some of the basics.

CUDA is NVIDIA's set of libraries for working with their GPUs. Some non-deep learning tasks can actually run on a CUDA-capable GPU faster than on a CPU.

Therefore, we'll install OpenCV with CUDA support, since the NVIDIA Jetson Nano has a small CUDA-capable GPU.

```
$ cd ~
```

```
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/4.1.2.zip
```

```
$ wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.1.2.zip
```

From there, extract the files and rename the directories for convenience:

```
$ unzip opencv.zip
```

```
$ unzip opencv_contrib.zip
```

```
$ mv opencv-4.1.2 opencv
```

```
$ mv opencv_contrib-4.1.2 opencv_contrib
```

And change into the OpenCV directory, followed by creating and entering a build directory:

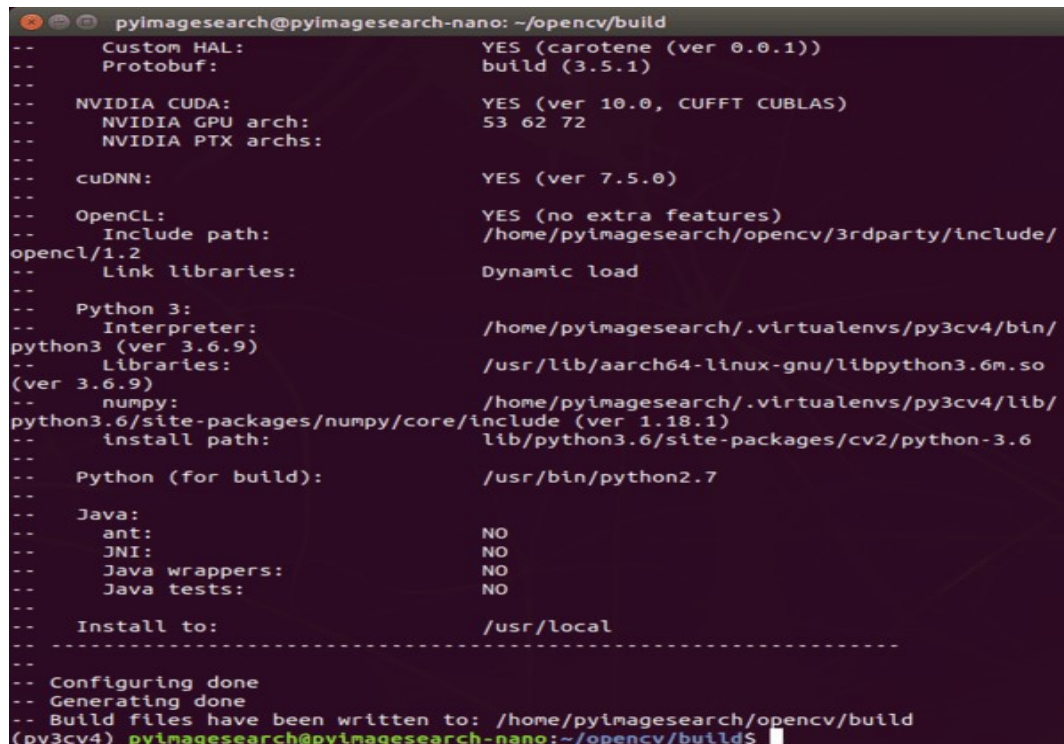
```
$ cd opencv
```

```
$ mkdir build
```

```
$ cd build
```

It is very important that you enter the next CMake command while you are inside (1) the ~/opencv/build directory and (2) the py3cv4 virtual environment. Take a second now to verify:

```
(py3cv4) $ pwd/home/nvidia/opencv/build
```



```
pyimagesearch@pyimagesearch-nano: ~/opencv/build
-- Custom HAL: YES (carotene (ver 0.0.1))
-- Protobuf: build (3.5.1)
--
-- NVIDIA CUDA: YES (ver 10.0, CUFFT CUBLAS)
-- NVIDIA GPU arch: 53 62 72
-- NVIDIA PTX archs:
--
-- cuDNN: YES (ver 7.5.0)
--
-- OpenCL: YES (no extra features)
-- Include path: /home/pyimagesearch/opencv/3rdparty/include/
-- opencv/1.2
-- Link libraries: Dynamic load
--
-- Python 3:
-- Interpreter: /home/pyimagesearch/.virtualenvs/py3cv4/bin/
-- python3 (ver 3.6.9)
-- Libraries: /usr/lib/aarch64-linux-gnu/libpython3.6m.so
-- (ver 3.6.9)
-- numpy: /home/pyimagesearch/.virtualenvs/py3cv4/lib/
-- python3.6/site-packages/numpy/core/include (ver 1.18.1)
-- install path: lib/python3.6/site-packages/cv2/python-3.6
--
-- Python (for build): /usr/bin/python2.7
--
-- Java:
-- ant: NO
-- JNI: NO
-- Java wrappers: NO
-- Java tests: NO
--
-- Install to: /usr/local
-----
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pyimagesearch/opencv/build
(py3cv4) pyimagesearch@pyimagesearch-nano:~/opencv/build$
```

Fig. 8.5 Installing Opencv Directory On NVIDIA

Compiling OpenCV will take approximately 2.5 hours. When it is done, you'll see 100%, and your bash prompt will return:

```
pyimagesearch@pyimagesearch-nano: ~/opencv/build
[100%] Building CXX object modules/videostab/CMakeFiles/opencv_videostab.dir/src/frame_source.cpp.o
[100%] Building CXX object modules/videostab/CMakeFiles/opencv_videostab.dir/src/global_motion.cpp.o
[100%] Building CXX object modules/superres/CMakeFiles/opencv_superres.dir/opencv_kernels_superres.cpp.o
[100%] Building CXX object modules/videostab/CMakeFiles/opencv_videostab.dir/src/inpainting.cpp.o
[100%] Building CXX object modules/videostab/CMakeFiles/opencv_videostab.dir/src/log.cpp.o
[100%] Linking CXX shared library ../../lib/libopencv_superres.so
[100%] Built target opencv_superres
[100%] Building CXX object modules/videostab/CMakeFiles/opencv_videostab.dir/src/motion_stabilizing.cpp.o
[100%] Building CXX object modules/videostab/CMakeFiles/opencv_videostab.dir/src/optical_flow.cpp.o
[100%] Building CXX object modules/videostab/CMakeFiles/opencv_videostab.dir/src/outlier_rejection.cpp.o
[100%] Building CXX object modules/videostab/CMakeFiles/opencv_videostab.dir/src/stabilizer.cpp.o
[100%] Building CXX object modules/videostab/CMakeFiles/opencv_videostab.dir/src/wobble_suppression.cpp.o
[100%] Linking CXX shared library ../../lib/libopencv_videostab.so
[100%] Built target opencv_videostab
Scanning dependencies of target opencv_python3
[100%] Building CXX object modules/python3/CMakeFiles/opencv_python3.dir/src2/cv2.cpp.o
[100%] Linking CXX shared module ../../lib/python3/cv2.cpython-36m-aarch64-linux-gnu.so
[100%] Built target opencv_python3
(py3cv4) pyimagesearch@pyimagesearch-nano:~/opencv/build$
```

DETECTION PHASE

Step 1: login

step 2: connect wifi

step 3: open terminal

step 4: open folder jetson inference by using this command this ----| cd jetson-inference

step 5: activate docker after changing directory ----| docker/run.sh

step 6: open code directory ----| cd python/training/detection/ssd

step 7: run your command ----| detectnet --model=models/fire/ssd-mobilenet.onnx -
-labels=models/fire/labels.txt --input-blob=input_0 --output-cvg=scores --output-
bbox=boxes /dev/video0

step 8: close

TRAINING PHASE

Step 1: login

Step 2: connect wifi

Step 3: open terminal

Step 4: open folder jetson inference by using this command this ----| `cd jetson-inference`

Step 5: activate docker after changing directory ----| `docker/run.sh`

Step 6: open code directory ----| `cd python/training/detection/ssd`

Step 7: create new folder in code base by navigating storage > Home > jetson_inference > python > trainign > detection > ssd > data (create new folder here)

Step 8: open the new folder created by you

Step 9: create new .txt folder call labels.txt

Step 10: open labels.txt file and enter your label name for e.g 'fire' 'matured fruit' and save NOTE: no empty spaces should be given

Step 11: enter the command for training images ----| `camera-capture /dev/video0`

Step 12: navigate to the gui input and set the field given below

Dataset type : Detection

Dataset path : navigate to your folder

Class Labels : navigate to your labels.txt

Current set : train

Step 13: focus your camera to the object you are trying to train

Step 14: uncheck clear on unfreeze

Step 15: click on Freeze/edit(space) button

Step 16: draw bounding box on the object by using mouse

Step 17: after drawing bounding box classes will be automatically loaded, now you have to map the class in the dropdown button

Step 18: save by using save button

Step 19: click freeze/edit button to unfreeze the screen

Step 20: continue till 60 images for a class

Step 21: After capturing the needed image close the window.

Step 22: In the terminal enter the command for training ----| python3 train_ssd.py -
-dataset-type=voc --data=data/fire --model-dir=models/fire --batch-size=2 --
workers=1 --epochs=30

Step 23: wait for the system to complete training\

Step 24: export the trained files by using this command ----| python3
onnx_export.py --model-dir=models/fire

Step 25: run the model by using the command ----| detectnet --
model=models/fire/ssd-mobilenet.onnx --labels=models/fire/labels.txt --input-
blob=input_0 --output-cvg=scores --output-bbox=boxes /dev/video0

DATA_PREPROCESSING.PY

```
from ..transforms.transforms import *
```

```
class TrainAugmentation:
```

```
    def __init__(self, size, mean=0, std=1.0):
```

```
        """
```

```
        Args:
```

```
            size: the size the of final image.
```

```
            mean: mean pixel value per channel.
```

```
        """
```

```
        self.mean = mean
```

```
        self.size = size
```

```
        self.augment = Compose([
```

```

        ConvertFromInts(),
        PhotometricDistort(),
        Expand(self.mean),
        RandomSampleCrop(),
        RandomMirror(),
        ToPercentCoords(),
        Resize(self.size),
        SubtractMeans(self.mean),
        lambda img, boxes=None, labels=None: (img / std, boxes, labels),
        ToTensor(),])

    def __call__(self, img, boxes, labels):
        """
            Args:
                img: the output of cv.imread in RGB layout.
                boxes: bounding boxes in the form of (x1, y1, x2, y2).
                labels: labels of boxes.
        """
        return self.augment(img, boxes, labels)

class TestTransform:
    def __init__(self, size, mean=0.0, std=1.0):
        self.transform = Compose([
            ToPercentCoords(),
            Resize(size),
            SubtractMeans(mean),
            lambda img, boxes=None, labels=None: (img / std, boxes, labels),
            ToTensor(),
        ])
    def __call__(self, image, boxes, labels):

```



```

        return self.transform(image, boxes, labels)
class PredictionTransform:
    def __init__(self, size, mean=0.0, std=1.0):
        self.transform = Compose([
            Resize(size),
            SubtractMeans(mean),
            lambda img, boxes=None, labels=None: (img / std, boxes, labels),
            ToTensor()    ])
    def __call__(self, image):
        image, _, _ = self.transform(image)
        return image

```

PREDICTOR.PY

```

import torch
from ..utils import box_utils
from .data_preprocessing import PredictionTransform
from ..utils.misc import Timer
class Predictor:
    def __init__(self, net, size, mean=0.0, std=1.0, nms_method=None,
        iou_threshold=0.45,    filter_threshold=0.01,    candidate_size=200,
sigma=0.5, device=None):
        self.net = net
        self.transform = PredictionTransform(size, mean, std)
        self.iou_threshold = iou_threshold
        self.filter_threshold = filter_threshold
        self.candidate_size = candidate_size
        self.nms_method = nms_method
        self.sigma = sigma

```

```

        if device:
            self.device = device
        else:
            self.device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
        self.net.to(self.device)
        self.timer = Timer()
    def predict(self, image, top_k=-1, prob_threshold=None):
cpu_device = torch.device("cpu")
        height, width, _ = image.shape
        image = self.transform(image)
        images = image.unsqueeze(0)
        images = images.to(self.device)
        self.net.eval()
        with torch.no_grad():
            self.timer.start()
            scores, boxes = self.net.forward(images)
            #print("Inference time: ", self.timer.end())
        boxes = boxes[0]
        scores = scores[0]
        if not prob_threshold:
            prob_threshold = self.filter_threshold
        # this version of nms is slower on GPU, so we move data to CPU.
        boxes = boxes.to(cpu_device)
        scores = scores.to(cpu_device)
        picked_box_probs = []
        picked_labels = []
        for class_index in range(1, scores.size(1)):

```

```

probs = scores[:, class_index]
mask = probs > prob_threshold
probs = probs[mask]
if probs.size(0) == 0:
    continue
subset_boxes = boxes[mask, :]
box_probs = torch.cat([subset_boxes, probs.reshape(-1, 1)], dim=1)
box_probs = box_utils.nms(box_probs, self.nms_method,
                           score_threshold=prob_threshold,
                           iou_threshold=self.iou_threshold,
                           sigma=self.sigma,
                           top_k=top_k,
                           candidate_size=self.candidate_size)
picked_box_probs.append(box_probs)
picked_labels.extend([class_index] * box_probs.size(0))
if not picked_box_probs:
    return torch.tensor([]), torch.tensor([]), torch.tensor([])
    picked_box_probs = torch.cat(picked_box_probs)
picked_box_probs[:, 0] *= width
picked_box_probs[:, 1] *= height
picked_box_probs[:, 2] *= width
picked_box_probs[:, 3] *= height
return picked_box_probs[:, :4], torch.tensor(picked_labels),
picked_box_probs[:, 4]

```

CHAPTER 9

SNAPSHOTS

9.1 SAMPLE INPUT: FIRE

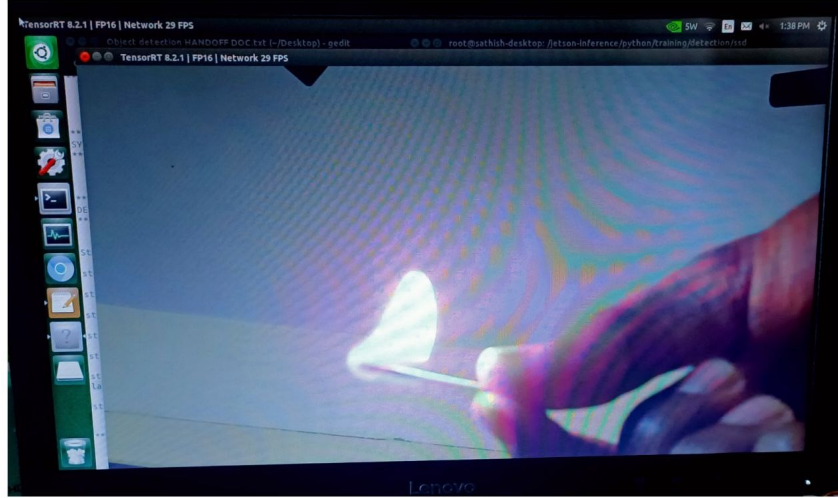


Fig. 9.1 Sample Input

Fig. 9.1 shows the Fire Detection using the SSD algorithm (sample input fire is analysing).

9.2 SAMPLE OUTPUT : FIRE DETECTION

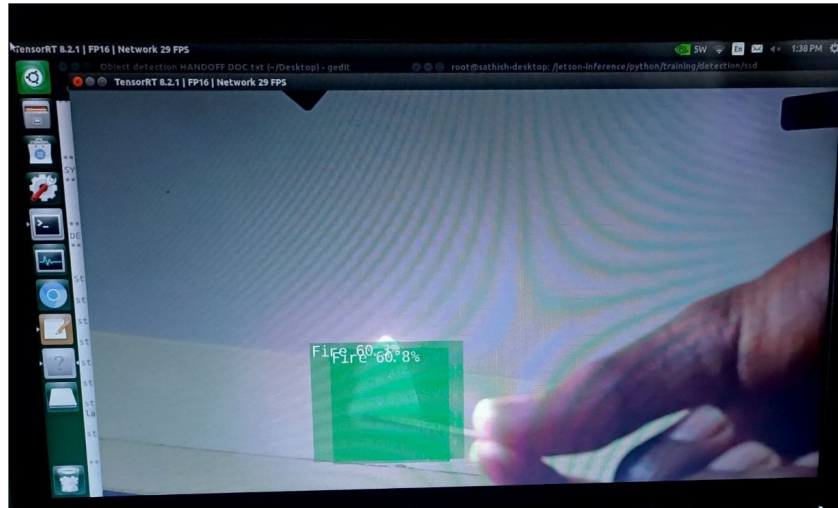


Fig. 9.2 Sample Output

Fig. 9.2 shows the Fire Detection using machine learning and SSD algorithm is successfully detected.

CHAPTER 10

CONCLUSION AND FUTURE ENHANCEMENT

10.1 CONCLUSION

In conclusion, our project aimed to develop and train an SSD algorithm on a small dataset. Despite the challenges of working with a small dataset, we were able to improve the performance of the model through the use of techniques such as data augmentation and transfer learning. Our results showed that the model was able to achieve reasonable accuracy on the test dataset, but there is certainly room for further improvement. We identified potential limitations of the model, such as the risk of overfitting, and future work could explore ways to mitigate these issues. Overall, this project provided valuable experience in working with machine learning algorithms, data preprocessing, and model evaluation. It also highlighted the importance of carefully selecting and preparing datasets for machine learning tasks.

10.2 FUTURE ENHANCEMENT

In terms of future enhancements, further research and development can be done to improve the accuracy and speed of fire detection and prevention systems. This can be achieved by using more advanced machine learning algorithms and by incorporating additional sensors and data sources to improve the overall reliability of the system. With continued practice and exploration, we believe that the skills and techniques developed through this project can be applied to more advanced machine learning projects in the future.

REFERENCES

1. K. S. Kotecha and V. A. Gulhane, "Fire Detection Using Machine Learning Techniques: A Review," 2021 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), pp. 1-5, 2021.
2. R. Tiwari and S. P. Tiwari, "Fire Detection using Machine Learning Techniques," International Journal of Computer Applications, vol. 181, no. 3, pp. 1-7, 2018.
3. M. M. Hasan, M. A. Hossain, and M. A. Rahman, "Real-time Fire Detection System using Machine Learning Technique," 2020 International Conference on Computer, Communication, Chemical, Materials and Electronic Engineering (IC4ME2), pp. 1-6, 2020.
4. S. Mehta, S. Jain, and S. Jain, "Fire Detection and Prevention using Machine Learning Techniques," International Journal of Advanced Research in Computer Science, vol. 9, no. 5, pp. 188-193, 2018.
5. A. A. M. El-Hefnawy and M. A. Fathalla, "Fire Detection Using Machine Learning Algorithms," International Journal of Intelligent Computing and Cybernetics, vol. 10, no. 2, pp. 119-135, 2017.
6. Y. Zhang, J. Qian, and H. Chen, "Research on Fire Detection Algorithm Based on Machine Learning," 2021 IEEE 7th International Conference on Computer and Communications (ICCC), pp. 34-38, 2021.
7. M. Khalid and J. Qadir, "Intelligent Fire Detection System using Machine Learning Algorithms," 2021 6th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1-5, 2021.
8. C. M. Anjana and P. Sudhakar, "Fire Detection using Machine Learning Algorithms: A Review," 2020 International Conference on Inventive Computation Technologies (ICICT), pp. 1-5, 2020.

9. F. C. Yu, Y. Zhang, and Y. Luo, "A Fire Detection Algorithm Based on Deep Learning and Computer Vision," 2021 IEEE 6th International Conference on Computer and Communications (ICCC), pp. 295-299, 2021.
10. D. H. Kim, Y. S. Kim, and S. W. Lee, "Fire Detection in Video Surveillance System using Machine Learning Techniques," 2018 10th International Conference on Advanced Communication Technology (ICACT), pp. 652-656, 2018.
11. R. B. V. Shanthini and N. K. Narendranath, "A Review on Fire Detection Techniques using Machine Learning Algorithms," 2021 International Conference on Electronics and Sustainable Communication Systems (ICESC), pp. 114-119, 2021.
12. A. Yadav, N. Jaiswal, and A. Jain, "A Machine Learning-based Fire Detection and Control System," 2021 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pp. 1-5, 2021.
13. T. Gao, Y. Xiong, and S. Xiong, "Fire Detection Method Based on Multi-feature Fusion and Machine Learning Algorithm," 2021 IEEE 8th International Conference on Electrical and Electronics Engineering (ICEEE), pp. 137-141, 2021.
14. H. Choi, J. Park, and J. Yoo, "Real-time Fire Detection System using Deep Learning and Thermal Imaging Cameras," 2021 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW), pp. 1-2, 2021.