Professor: Lorenzo Carnevale

Arya Khosravirad

Matricola: 534 170

Università degli Studi di Messina

Drone-Based Forest Fire Detection and Suppression System

ABBSTRACT

Forest fires are a growing problem, causing damage to the environment, property, and even risking human lives. Traditional ways of detecting and fighting these fires, like lookout towers, satellites, and planes, can be slow, expensive, and not always effective. To address this, I've developed a system using drones to detect and suppress forest fires quickly and efficiently. These drones are equipped with machine learning technology to spot early signs of fire from real-time data collected by wireless sensors scattered throughout the forest. Once a fire is detected, the drones immediately fly to the location and self-destruct to release fire-suppressing foam directly on the flames. This method is much faster, cheaper, and safer than current firefighting approaches. My research compares different fire detection technologies and shows how drones, supported by wireless sensors, are the best solution. After training an AI model to recognize fire images accurately, the drone simulation demonstrated quick and accurate fire detection and response. Overall, this project provides an innovative and practical approach to managing forest fires, offering a way to control them before they cause major damage.

INTRODUCTION

Context Definition

- Forest fires are big fires that spread quickly through forests and other areas with trees. They can cause a lot of damage and destroy plants, animals, homes, and sometimes even lives. They also release a lot of smoke and gases, which makes the air quality worse and adds to climate change.

- Lately, forest fires are happening more often and are getting worse. This is mostly because of climate change and human activities. Hotter temperatures, long dry seasons, and extreme weather make it easy for forests to catch fire. Right now, we use things like lookout towers, satellite images, and fire-fighting teams or planes to detect and fight fires, but these methods can be slow and expensive. By the time help arrives, the fire might have already spread.

- Because of this, we need a better way to quickly find and put out these fires before they cause too much damage. This project looks at a new solution: using drones to quickly find fires and respond right away to stop them from spreading.

Problem Identification

- Current methods for fighting forest fires have several big challenges. First, they are often slow to respond. By the time a fire is detected and fire fighting teams are sent out, the fire might have already spread over a large area, making it much harder to control.

- Another issue is the high cost of these traditional methods. Using planes, helicopters, and ground teams to fight fires is very expensive. It requires a lot of resources, including fuel, equipment, and personnel, all of which add up quickly.

- Lastly, these methods are not always efficient, especially in remote or difficult to reach areas. Dense forests, rough terrain, and areas far from roads can make it hard for fire-fighters to get there quickly. This delay gives the fire more time to spread and makes it harder to contain.

- Because of these challenges, there is a need for faster, cheaper, and more efficient ways to detect and respond to forest fires.

Proposed Solution

- The solution proposed is to use "suicidal drones" to detect and quickly respond to forest fires. These drones are equipped with machine learning technology that can recognize signs of fire early from images they capture. Once a fire is detected, the drone flies directly to the location and self-destructs, spreading fire-fighting materials like water or chemicals over the flames to quickly suppress the fire.

- This approach is fast and efficient. Unlike traditional methods, the drones can respond almost immediately, reducing the time it takes to start fighting the fire. The self-destruct mechanism means that the drones can reach hard to access areas and directly drop fire-fighting materials where they are needed most, stopping the fire before it grows out of control. This makes the entire process of detection and suppression much quicker, cheaper, and more effective.

Objectives

- The main goals of this project are:

- Enhance Fire Detection Speed: Use best method to detect fires quickly and accurately, allowing for an immediate response before the fire spreads.

- Reduce Response Time: Deploy suicidal drones to reach fires faster than traditional fire-fighting teams or aircraft, especially in remote or hard to reach areas.

- Improve Cost-Efficiency: Provide a cheaper alternative to current fire-fighting methods by using drones, which require fewer resources and can operate autonomously.

- Demonstrate Feasibility: Create a simulation to prove that this system can effectively detect and suppress fires, showing how the proposed drone solution can work in real world scenarios.

State of the Art

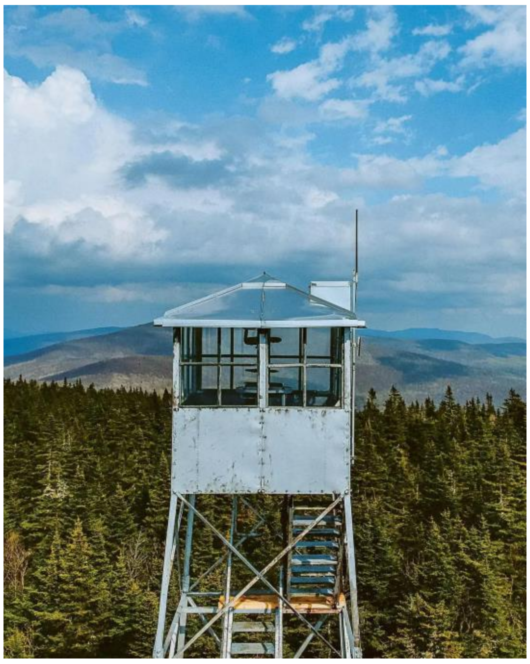
In this section, we will look at the current ways to detect forest fires and figure out which one works best to help drones respond quickly. Detecting fires fast and accurately is key to stopping them before they spread. We will compare different fire detection methods based on how quickly they can find a fire, how much they cost, how well they work in different situations, and how accurate they are at spotting fires without false alarms. The goal is to find the best technology that can quickly alert drones, so they can reach the fire fast and effectively put it out. By comparing these methods, we hope to find the best way to catch fires early and manage them better.

Overview of Fire Detection Systems

1. Human-Based Observation

This method is pretty basic: people watch for fires from towers or during patrols. They look for smoke or flames to spot a fire.

Human-based fire observation is cheap and easy to set up, requiring only people and simple tools. However, it is unreliable as fires can be missed, especially in dense forests. It also has a slow response time, and requires constant staffing, making it hard to maintain across large or difficult areas.

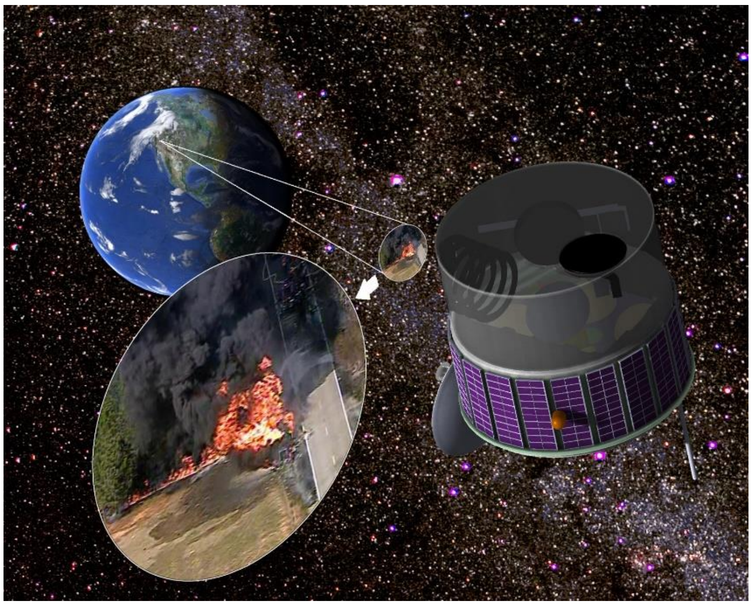


2. Satellite Systems

Satellites in space have sensors that take pictures of the Earth. They can spot heat or smoke, which helps detect fires over big areas.

Satellites are used to spot and detect forest fires. Two main satellites, AVHRR (launched in 1998) and MODIS (launched in 1999), were specifically built to help detect fires. However, these satellites can only take pictures of areas on Earth every two days, which is quite slow for catching fires early. Plus, the quality of the images can be affected by things like bad weather.

Satellites can cover large areas, including remote regions, and help track how fires spread. However, they are very expensive to launch and maintain, have delays in detection as they only pass over areas every 2 days, and their visibility is often hindered by weather conditions.



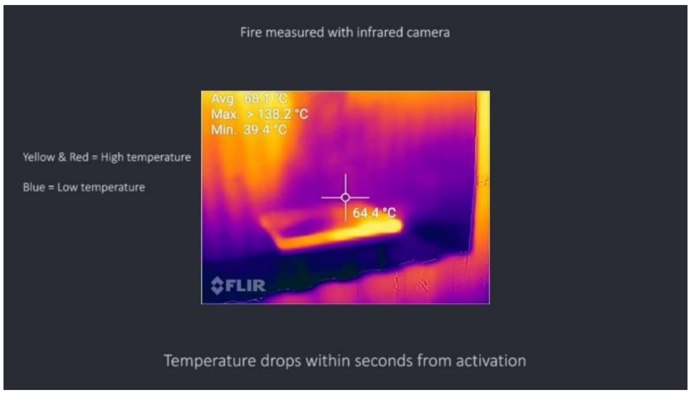
3. Optical Cameras

Today, two main types of sensor networks are used to detect fires: camera surveillance systems and wireless sensor networks. Advances in technology like sensors, digital cameras, image processing, and computers have led to the development of systems that can automatically recognize and warn about forest fires early.

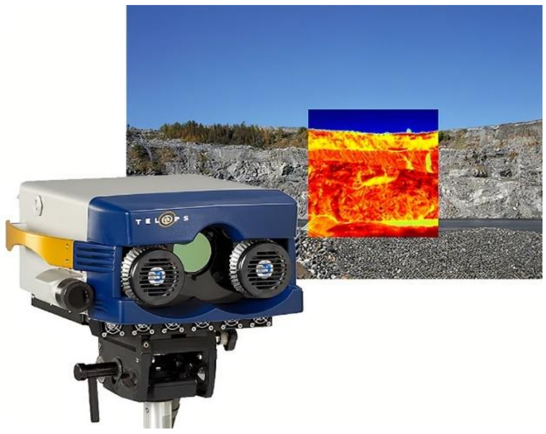
There are different types of sensors used for fire detection:

Video Cameras: These pick up visible light to detect smoke during the day and fire at night.

Infrared (IR) Cameras: These detect heat coming from a fire, which helps even if smoke or flames are not visible.


Fire measured with infrared camera
Yellow & Red = High temperature
Blue = Low temperature
Avg. 68.1 °C
Max. > 138.2 °C
Min. 39.4 °C
64.4 °C
FLIR
Temperature drops within seconds from activation

IR Spectrometers: They analyse the specific light spectrum to identify smoke particles.



LIDAR Systems: LIDAR uses laser beams to scan the environment, measuring how the light is reflected back after encountering smoke particles. By assessing the time taken and changes in the reflected light, LIDAR systems can accurately detect the presence of smoke, making them effective for real-time monitoring.

All of these optical systems follow a similar concept: they capture images over time. Each image is made of pixels, and the system analyses them to track movements or detect smoke or fire. If the system detects enough signs of smoke or flames, it uses algorithms to decide whether to trigger an alarm. Most of these systems are connected to geographical maps to locate the fire accurately.

Cameras provide continuous monitoring and are more affordable than satellites. However, they have a limited view that can be blocked by terrain or weather. They can also give false alarms by mistaking fog or reflections for smoke and may be slow to detect small fires.

4. Wireless Sensor Networks (WSNs)

Wireless sensor networks (WSNs) are getting popular for detecting forest fires early. These networks use small sensors that measure environmental factors like temperature, humidity, and gases (e.g., carbon monoxide) and send the data wirelessly to a base station. Each sensor runs on a battery and combines sensing, processing, and wireless communication on one small circuit board. Unlike cell phones, WSNs don't have regular recharging, so they’re designed to use minimal power, often supported by solar panels. One common technology used in WSNs is ZigBee, which focuses on low power use and short range communication.

Earlier fire detection methods like observation towers, cameras, and satellites were not always effective in detecting fires quickly or in difficult weather. WSNs solve this by providing real time data and can self organize to create a strong, efficient network.

Feature	WSNs	Camera Surveillance Systems	Satellite Systems
Cost	Medium - initial setup and maintenance	High - installation & maintenance of cameras	Very High - satellite launch & maintenance
Detection Speed	Fast - real-time data	Medium - relies on visual cues, can be delayed	Slow - typically every 1-2
Accuracy in Early Detection	High - detects temperature, humidity changes	Medium - depends on line of sight; weather can obstruct view	Low - can miss small or early fires due to coverage
Coverage Area	Localized, specific to sensor placement	Moderate - limited to camera range, line of sight	Large - global coverage but not continuous
Reliability in All Weather	High - detects physical changes in environment	Low - affected by rain, fog, and night conditions	Low - cloud cover and weather distort images
Maintenance Needs	Moderate - battery replacement, repairs	High - cleaning lenses, adjusting cameras	High - costly maintenance and updates

Power Source	Low power consumption; often solar-powered	Requires consistent power, can be high	Self-powered but requires significant energy
False Alarms	Low - measures actual changes in environment	Medium - can be triggered by visual obstructions	Low-medium - may trigger from non-fire heat sources
Location Precision	High - accurate data from sensors	Medium - relies on camera range & angle	Low - heat or smoke seen from afar, not precise
Ability to Self-Heal/Adapt	Yes - self-organizing network to reroute data	No - static camera positions	No - fixed orbit paths
Scalability	High - can add more sensors to expand coverage	Medium - adding cameras can be costly	Low - difficult to expand satellite coverage
Power Consumption	Low - designed for long-term battery life	High - continuous operation and transmission	High - significant power for long-distance transmission

Based on the comparison of different fire detection systems, **Wireless Sensor Networks (WSNs)** are the best option for helping drones find and respond to fires. Here's why:

Why WSNs Work Well with Drones

- **Fast and Real-Time Detection**: WSNs quickly pick up changes in temperature, humidity, and gases, so they can spot fires as soon as they start. This is super helpful for drones, which need accurate information quickly to respond before the fire spreads.
- **High Accuracy for Early Detection**: These sensors are very good at catching even small changes in the environment, making it easier to detect fires early on. This means drones can be sent out quickly, stopping the fire before it grows.
- **Precise Location**: WSNs give **exact locations** of where fires are happening. This is crucial for drones because they need to know exactly where to go and where to drop their fire-fighting materials.
- **Works in All Weather**: WSNs aren't easily affected by weather like fog, rain, or darkness. This means the data they send to drones is reliable anytime, day or night.
- **Low Power and Self-Healing**: WSNs use little power, often running on solar energy, so they're easy to maintain. Even if some sensors break or fail, the

network can still work by rerouting the data, so the monitoring remains consistent.

Comparison to Other Systems

- **Cameras**: Cameras can be blocked by obstacles and weather, which can cause delays or false alarms. Drones need clear and reliable data, and cameras might not always provide this.
- **Satellites**: Satellites cover large areas, but they don't provide real-time data. They only send images every 1–2 days, which is too slow for quick drone response.

Conclusion

Overall, WSNs are the best choice for helping drones detect fires. They provide real-time, accurate data in any weather, and they pinpoint the fire's exact location, making them ideal for quick drone action to control fires effectively.

Comparison of WSN-Based Fire Detection Systems

Several projects have used WSNs for forest fire detection:

Lloret et al. (Spain): A mesh network of sensors and IP cameras to detect fires early. When a fire is detected, nearby cameras turn on to confirm it. However, image transfer uses a lot of power, and cameras only work well in clear weather.

Son et al. (South Korea): A fire detection system using WSNs to measure temperature and humidity, sending data to a base station. This system is more about evaluating fire risks based on weather rather than network reliability.

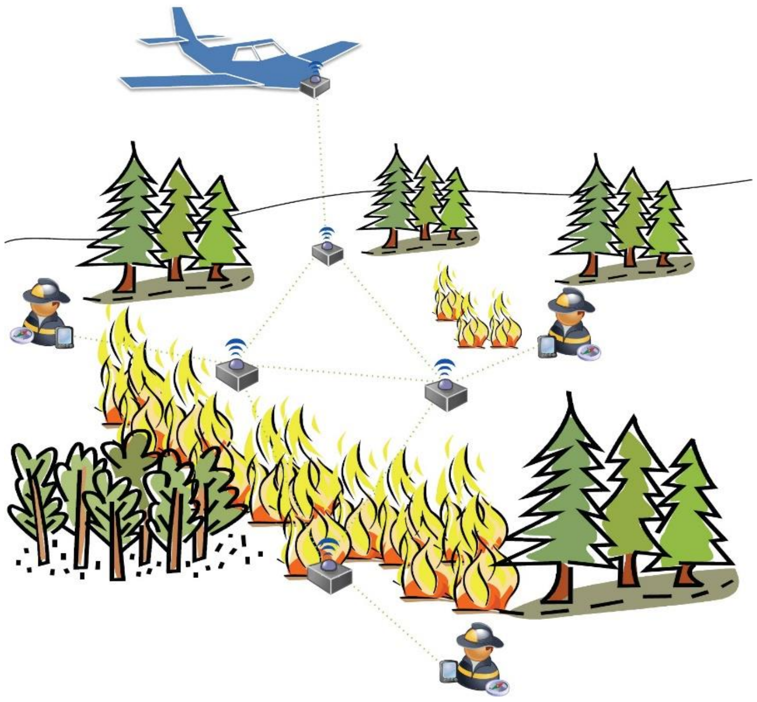
Hartung et al. (FireWxNet): A system for monitoring weather in forests, combining WSNs and web cameras. While it tracks fire behaviour and weather conditions, it doesn't focus on early fire detection.

Alonso et al. (ZigBee Network for Fire Detection): This system uses a network of low-power sensors connected via **ZigBee** technology to detect changes in temperature, humidity, and gas concentration, which are key indicators of forest fires. The ZigBee protocol ensures efficient power usage, allowing sensors to run on batteries or solar panels for extended periods. Data is transmitted in real-time to a central base station for analysis. The system prioritizes early fire detection with fast alerts and is optimized for energy efficiency and reliability, making it suitable for continuous operation in remote forest areas.

In short, WSNs are an emerging, powerful tool for fire detection that is more efficient than older methods and can send real-time alerts for quick action.

Which WSN to use

Among the four systems, Alonso et al.'s ZigBee based fire detection network is the most efficient and effective for early fire detection. This is due to its use of ZigBee technology, which allows for low power consumption, real time data transfer, and is specifically designed for early alerts. In contrast, Lloret et al. (Spain) relies on IP cameras for fire confirmation, which consumes a lot of power and is unreliable in poor weather. Son et al. (South Korea) measures temperature and humidity but focuses more on assessing fire risk rather than providing immediate alerts, likely using less efficient wireless protocols. Hartung et al. (FireWxNet) monitors weather and fire behavior, but its use of web cameras faces similar power and reliability issues. Alonso et al.'s system is optimized for detecting fires quickly by monitoring key environmental factors like temperature, humidity, and gases, offering a fast, reliable, and energy-efficient solution for rapid fire response compared to the other approaches.so we can conclude that Alonso et al.'s is the best option.



Sensors are very accurate and efficient, quickly picking up environmental changes to detect fires early. They can cover large areas by being set up across vast forests and remote places and provide quick alerts, sending data in real time for a faster fire response. Additionally, they are versatile, as they can track other environmental data like weather and wildlife. However, they come at a medium cost, requiring some investment for setup, and need regular maintenance to ensure they are charged and functioning, which can be time-consuming.

Overview of Fire Suppression Techniques

Now that we’ve decided that **Wireless Sensor Networks (WSNs)** are the best for detecting fires, let’s find the best way to **put those fires out**. Fires need three things to burn: **oxygen, heat, and fuel** (like wood or leaves). This is called the **fire triangle**, and our goal is to break this triangle by removing at least one of these elements.

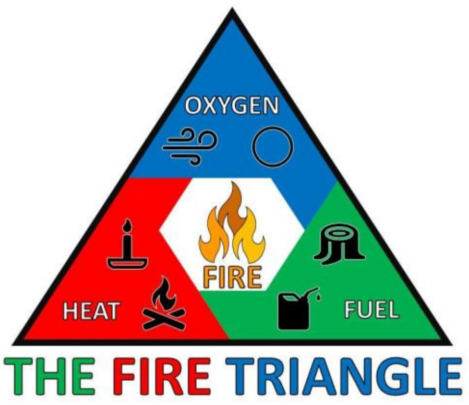
Understanding and Breaking the Fire Triangle

1. **Oxygen**: Fires need air to burn, and since the atmosphere has plenty of it, it’s usually not easy to cut off. But doing so can stop a fire.
2. **Heat**: Heat is what keeps the fire going. Taking away the heat, like cooling it with water, stops the fire.
3. **Fuel**: This is anything that can burn. If we remove or clear away fuel (like plants or trees), the fire has nothing to burn.

How Fire Suppression Works

- **Remove Fuel**: Clearing vegetation creates breaks in the fire's path.
- **Cut Off Oxygen**: Covering small fires with dirt or blankets can smother them.
- **Cool the Heat**: Using water or chemicals to lower the fire's temperature can put it out.

Fire suppression techniques aim to break one or more sides of the fire triangle to control and stop the fire effectively.



Fire Suppression Techniques Overview

There are several ways to control wildland fires, including direct attack, indirect attack, flank attack, and parallel attack. Each method has its advantages and disadvantages depending on the size, behavior, and intensity of the fire. Below is an overview of each approach, including its pros and cons:

1. Direct Attack

Firefighters work right at the fire's edge, using tools like hoses, shovels, or fire-retardant materials to put out the fire or slow it down. This method is best when flames are less than 4 feet tall, so firefighters can safely get close. If flames are 4 to 8 feet, heavier equipment like bulldozers or airdrops from planes might be needed. This method works well for smaller fires or less intense parts of a larger fire.

Direct fire attack allows for a fast response, quickly putting out small fires before they spread. It also provides precise control by directly targeting the flames. However, there are safety risks because firefighters have to be close to the fire, which can be dangerous if it spreads quickly. This method is also only good for less intense fires, so it's not suitable for large, strong fires, especially if flames are taller than 8 feet.

2. Indirect Attack

This technique involves building control lines (barriers) at a safe distance from the fire. Firefighters then burn the vegetation between the fire and control line, creating a buffer to stop the fire from spreading. This is used when flames are over 8 feet tall or when it's too dangerous to engage the fire directly.

An indirect fire attack puts safety first by keeping firefighters at a safe distance from intense flames. It is also effective for large fires, where directly engaging the fire is too dangerous. However, it takes time to set up, as control lines must be established and vegetation cleared or burned out. Additionally, this method can lead to potential damage to vegetation or structures, as some land may need to be sacrificed in controlled burns to stop the fire from spreading.

3. Flank Attack

In a flank attack, firefighters start from a safe area (an anchor point) on one or both sides (flanks) of the fire and work toward the head, which is the fastest moving part. The goal is to contain the fire from the sides as it moves forward. It works well when fires are spreading at a moderate rate.

A flank fire attack helps control the spread by stopping the fire's advance from the sides and is particularly effective for moderate fires where the flame spread isn't too rapid. However, it requires safe conditions, as burning out unburned areas between the fire and the control line can be risky. Additionally, this method may not quickly contain the fire, as it tends to be slower than a direct attack.

4. Parallel Attack

This method is similar to an indirect attack but with the control line built much closer to the fire's edge, often within 100 feet. Firefighters use mechanized equipment to build this line quickly. It's used when a direct attack isn't possible, but conditions allow for safe, close work near the flames.

A parallel fire attack allows for faster containment, as the control line is built closer to the fire than in an indirect attack. This approach also results in less land being sacrificed, as a smaller area is burned out. However, it requires proximity to the fire, meaning firefighters must work closer to the flames, which can be dangerous. Additionally, this method is labour and equipment intensive, needing machinery and skilled personnel to quickly establish effective control lines.

Conclusion

All of these methods aim to stop a fire by breaking its triangle by removing either the fuel, heat, or oxygen it needs to keep burning. The right method depends on the size of the fire, how intense it is, and the landscape. Direct attack is great for smaller, less intense fires because it can quickly put them out and save more land. But it can be risky for firefighters and isn't always possible for bigger, more dangerous fires.

If we could reduce the risks for firefighters and make direct attacks possible for larger fires, it would be the best approach. It would allow fires to be put out quickly at their source, with fewer dangers to people. Using drones could be the key to making this happen. they can tackle fires directly without putting human lives in harm's way.

Why Direct Attack Works Best with Suicidal Drones

Direct attack is all about fighting a fire right at its edge to stop it from spreading. It's very effective because it tackles the flames directly and quickly. However, sending firefighters close to the flames can be dangerous, especially if the fire is intense or in places that are hard to reach.

That's where **suicidal drones** make a difference. These drones can fly right up to the fire's edge and drop fire retardants exactly where they're needed, all without putting anyone at risk. Since they don't need roads or paths, they can get to spots that humans can't reach safely. This makes direct attack both faster and much safer.

Using these drones makes direct attack the best way to quickly and effectively control fires while keeping people out of harm's way.

Why Suicidal Drones Are Better Than Planes or Helicopters for Fighting Fires

Planes and helicopters usually drop water or fire retardant from high above the fire. While this helps, **suicidal drones** are much better in many ways.

1. **More Accurate**: Planes and helicopters release water and chemicals from way up in the sky, so the wind can blow it away from the fire. Suicidal drones can fly **right to the edge of the fire** and release the fire suppressant materials exactly where they're needed.

2. **Explode Inside the Fire**: Unlike planes that drop water from above, suicidal drones can actually **fly into the fire, explode, and spread fire fighting materials right in the flames**. This means they can attack the fire from the inside, hitting the hottest spots and stopping the fire more effectively.

3. **Reach Tough Areas**: Drones can easily get into places where planes or helicopters can’t safely fly, like dense forests, mountains, or valleys. They don’t need open space and can go where firefighters can't reach quickly.

4. **Less Waste of Materials**: Because drones drop the materials right into the fire, almost all of it hits the flames. Planes and helicopters often lose a lot of water or chemicals to wind and height, so drones use the materials much more efficiently.

5. **Keeps Firefighters Safe**: Pilots flying planes and helicopters near a fire can be in danger from smoke and flames. Since drones are unmanned, they can be sent in to do the job without risking any human lives.

6. **Quick to Launch and Use**: Drones can be set up and sent out fast, often from close to the fire. It’s much quicker than getting a plane or helicopter ready, which makes drones perfect for stopping a fire before it gets out of control.

7. **Cheaper to Run**: It costs a lot of money to fly planes and helicopters and keep them working. Drones are generally cheaper and easier to maintain, and they don’t need pilots, which makes them a more cost-effective option.

In short, suicidal drones are **more accurate, safer, and faster** than traditional firefighting aircraft. They fly directly into the fire, spreading materials where they’re needed most, making them an effective way to control and stop fires.

Selecting the Optimal Foam for Drone Fire Suppression

In this section, we compare different firefighting foams to find the best option for drone based fire suppression. The goal is to choose a foam that is quick to deploy, effective in extinguishing fires, and environmentally safe, considering the unique needs of drone operations.

1. Synthetic Foams (S1, S2, S3)

Synthetic foams are made using chemicals called surfactants, which help the foam expand and spread over a fire. These foams come in different concentrations (3% or 6%), which affects how well they work. They can be low, medium, or high expansion, meaning they spread quickly and cover large areas. However, their effectiveness can vary depending on their concentration and the type of fire they are used on.

2. Aqueous Film-Forming Foams (AFFF)

AFFF foams are specially made to quickly form a thin film over flammable substance, cutting off oxygen and stopping the fire fast. They contain a mix of regular surfactants and fluorosurfactants, which help create that protective barrier over the flames. They're great for rapid action and can cover large areas, and they even work well with seawater. Their expansion is moderate, so they can spread over fires quickly but may drain within 10-20 minutes.

3. Fluoroprotein Foams (FP1, FP2, FP3, FP4, FP5 AR)

Fluoroprotein foams come from proteins with added chemicals that help them spread better over a fire. They stick to the fire and prevent it from reigniting, which makes them very stable. With an expansion ratio above 7, they can cover a lot of ground and stay effective for a long time. Some types can be mixed with seawater, making them versatile for different situations.

4. Film-Forming Fluoroprotein Foams (FFFP)

FFFP foams are a mix of protein-based foams and other agents that help them spread quickly and form a stable protective layer over a fire. While their expansion ratio is slightly lower (greater than 6), they drain slowly (taking over 9 minutes), making them good for lasting fire suppression. These foams are useful when you need quick coverage and a lasting effect to keep the fire from reigniting.

5. Protein-Based Foams (P)

Protein foams are made from natural proteins, making them more eco friendly. They create a dense blanket over fires and have a high expansion ratio (greater than 7). Used at a 6% concentration, they are good at forming a stable layer over a fire for long lasting suppression. These foams are great for extended firefighting and are a safer choice for the environment compared to some synthetic alternatives.

Firefighting Foam Parameters

Firefighting agent	**Viscosity at 20°C, mPa·s**	**Biodegradation, %/28 days**	**Compatibility with sea water**	**Expansion ratio**	**50% Drain time, min**
S1 3%	16±2	>90	-	>8	>15
S2 6%	4±1	88	-	>8	>15
S3 6%	10±2	NA	-	>8	>15
AFFF AR 3%	NA	NA	+	>7	10-20
FP1 6%	6±1	NA	-	>7	>15
FP2 6%	4±1	NA	-	>7	>10
FP3 6%	20±2	NA	+	>7	>9
FP4 6%	6±2	NA	+	>7	>9
FP5 AR 3%	>200	>92	-	>7	>12
FFFP AR 6%	<850	NA	-	>6	>9
P 6%	<25	NA	+	>7	>9

The best foams for drones are Aqueous Film-Forming Foam (AFFF) and Film-Forming Fluoroprotein Foam (FFFP). AFFF works really well because it spreads quickly over a fire, cutting off the oxygen and putting out the flames fast. It's easy to spray and can even be mixed with seawater, making it great for covering big areas quickly. FFFP, on the other hand, is more stable and stops the fire from starting up again. It's also better for the environment since it breaks down naturally. If we need to put out a fire quickly, we go with AFFF. But if we want longer lasting fire control and care more about the environment, then FFFP is the better choice. Since our main goal is to put out fires as fast as possible, AFFF is the foam we use.

METHODOLOGY

Role of AI in Enhancing Drone Firefighting Efficiency:

Adding AI to the drone is important, even though WSNs already find the fire’s location. Fires change quickly wind shifts, and the fire can spread fast. While WSNs point to where the fire is, by the time the drone arrives, the situation might be different. The AI helps the drone analyses real time images and choose the best spot to drop firefighting materials.

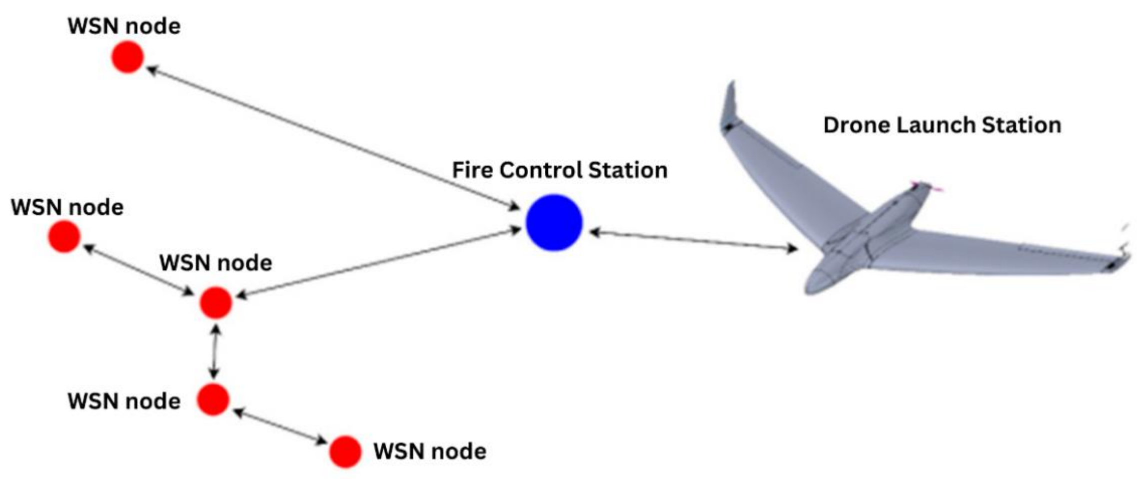
The WSN provides a general area, but AI allows the drone to target the exact spot needing attention. This is key since drones carry limited materials, and they need to be used effectively. AI also helps the drone navigate around obstacles like trees and rough terrain that WSNs might not fully map out.

In short, WSNs guide the drone to the fire’s general area, but AI ensures it reacts to the current situation quickly and accurately. This makes firefighting faster and more efficient.

Data Collection and Response Methodology:

The system works by gathering input from a network of **Wireless Sensor Network (WSN) nodes** spread across the forest. These sensors constantly monitor environmental conditions such as temperature, humidity, and gas levels to detect any signs of a fire. When they pick up any abnormal changes, the data is sent from node to node until it reaches a central point: the **Fire Control Station**. Here, the information is processed to pinpoint the fire's exact location and assess its severity.

Once the Fire Control Station identifies and confirms a fire, the alert is immediately passed on to the **Drone Launch Station**. At this point, a drone is quickly deployed to the fire's location. The drone’s job is twofold: it can either get a closer look to verify the fire or directly start fighting it by self destruction and spread fire retardant materials. This system is designed for speed and efficiency, making sure that fires are detected, located, and suppressed as quickly as possible to reduce damage and prevent the fire from spreading.



Code Explanation

The goal is to develop an AI model that can accurately detect fire in images, which will help guide a drone to identify and respond to fires in real time. The model will be trained to differentiate between images containing fire and those without it, allowing the drone to make precise decisions on where to deploy fire suppression materials effectively.

To achieve this, we use a dataset that is organized into "Training" and "Testing" folders, with a total of 1,900 labelled images. The "Training" data is split into two categories: "fire" and "nofire," each containing 760 images, providing a balanced set for learning. The "Testing" set contains 380 images that are a mix of both "fire" and "nofire" categories, and will be used to evaluate the model's performance. This dataset provides a solid foundation for training the AI, as it covers different visual scenarios of fire and non fire situations, helping the drone to make accurate real time decisions when detecting and responding to fires.

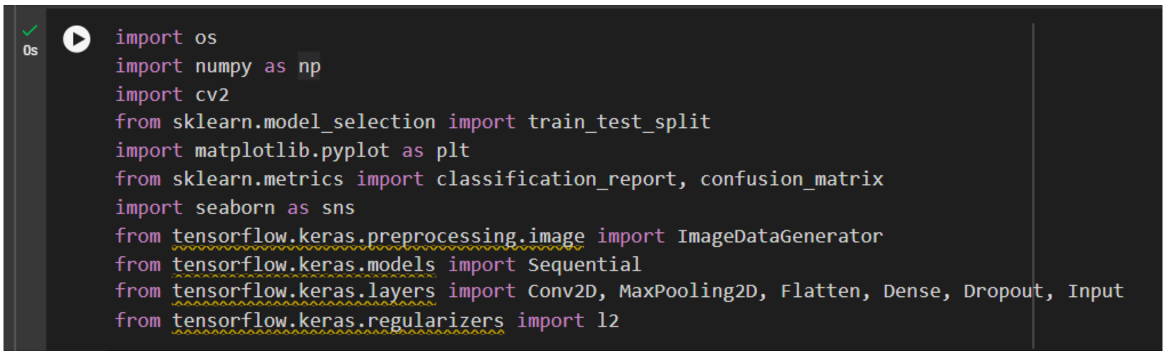
1. Data collection and Processing

The project used a set of images labeled as “fire”or “no fire”. Before using them, we changed the size of all images to 224x224 pixels. This made sure all the pictures were the same size, so it was easier to train the model.

Next, we split the data into two parts: one part to train the model and the other part to test it. The training data helped the model learn to see if an image showed fire or not. Later, we used the testing data to check how well the model learned.

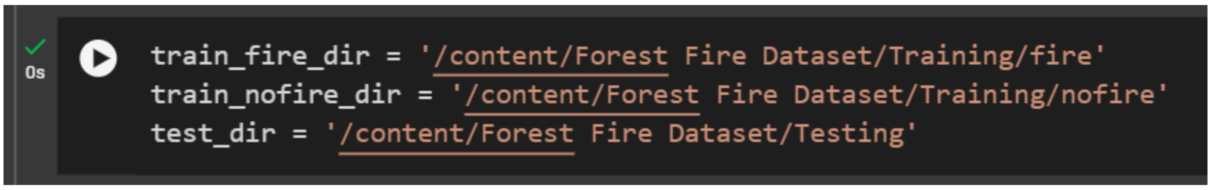
1. Importing Required Libraries

We started by importing some important libraries:



```python
import os
import numpy as np
import cv2
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
from tensorflow.keras.regularizers import l2
```

2. Defining Paths to the Directories



```python
train_fire_dir = '/content/Forest Fire Dataset/Training/fire'
train_nofire_dir = '/content/Forest Fire Dataset/Training/nofire'
test_dir = '/content/Forest Fire Dataset/Testing'
```

train_fire_dir:

This variable points to the folder where images of fire are stored.

These images are used to teach the model what fire looks like.

train_nofire_dir:

This variable points to the folder where images without fire are stored.

These images help the model learn what it looks like when there is no fire.

test_dir:

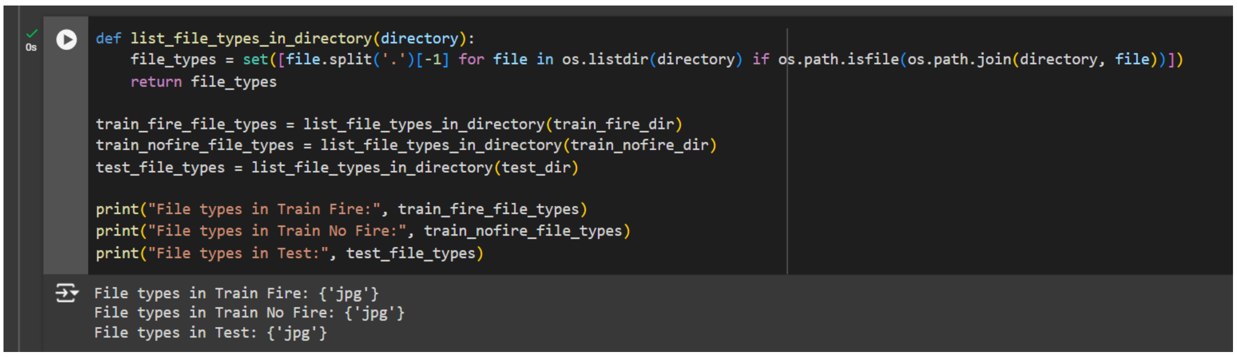
This variable points to the folder with images for testing the model after it's trained.

It has both fire and no fire images, which helps us check how well the model works.

3. Dataset Distribution Analysis:

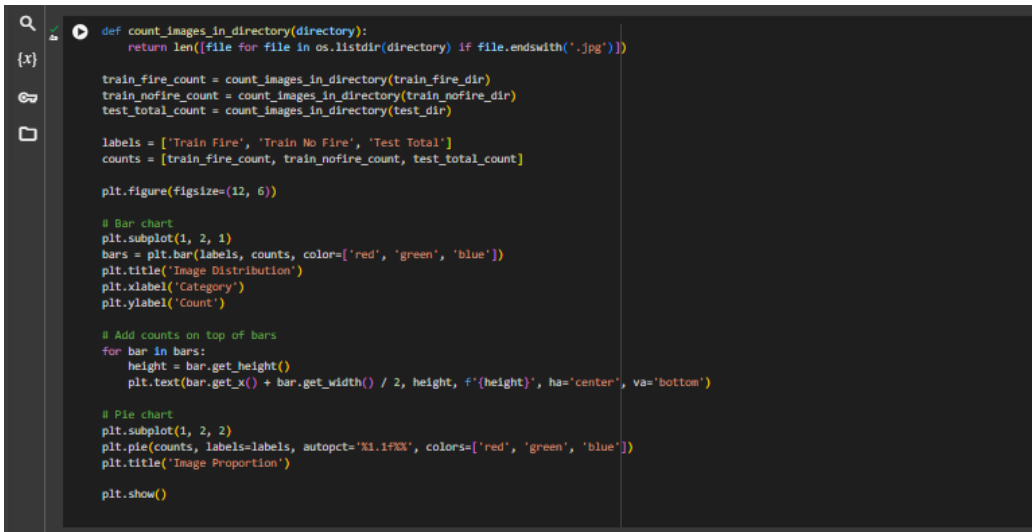
I started by trying to understand what kind of data I am working with. To do this, I created a function to count the images in each category: fire, no fire, and test images. Then i visualized the data distribution using both a bar chart and a pie chart.

- The reason for visualizing the data was to check if the dataset is balanced between fire and no fire images, which is important for training the model effectively.
- The visuals helped me see that the fire and no-fire images in the training set are balanced (40% each), while the test set makes up 20% of the total dataset. This balance is good because it ensures that the model won't be biased toward one class over the other.

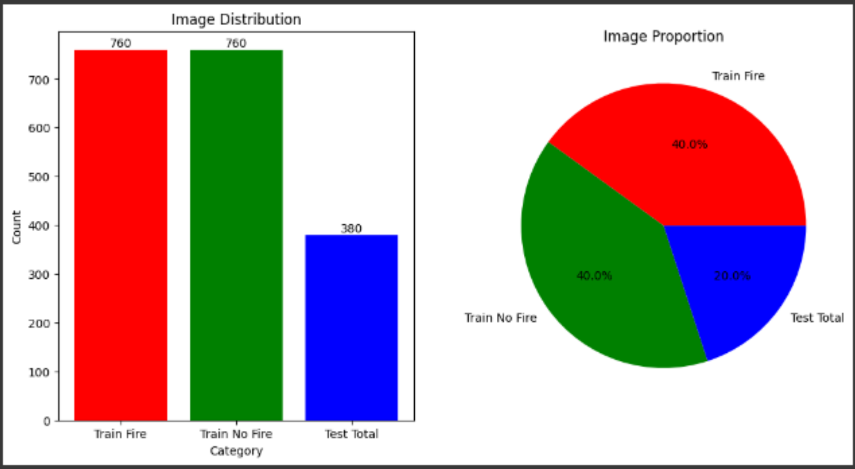


```python
def list_file_types_in_directory(directory):
    file_types = set([file.split('.')[-1] for file in os.listdir(directory) if os.path.isfile(os.path.join(directory, file))])
    return file_types

train_fire_file_types = list_file_types_in_directory(train_fire_dir)
train_nofire_file_types = list_file_types_in_directory(train_nofire_dir)
test_file_types = list_file_types_in_directory(test_dir)

print("File types in Train Fire:", train_fire_file_types)
print("File types in Train No Fire:", train_nofire_file_types)
print("File types in Test:", test_file_types)
```

```
File types in Train Fire: {'jpg'}
File types in Train No Fire: {'jpg'}
File types in Test: {'jpg'}
```

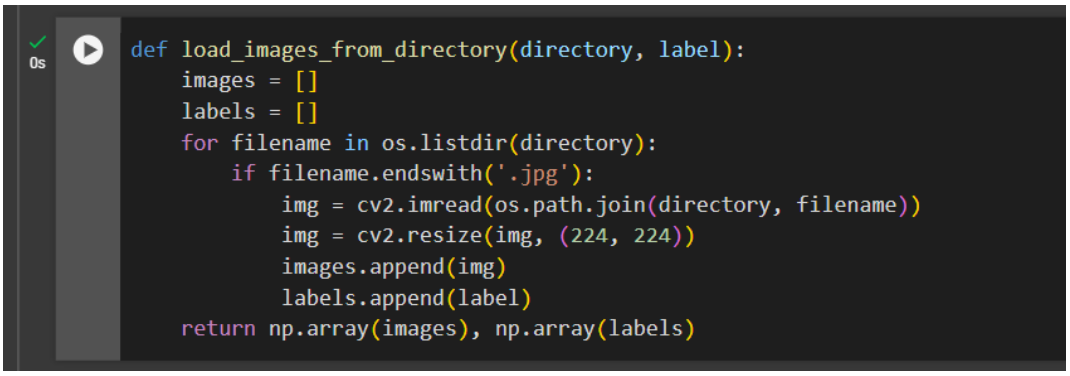


```python
def count_images_in_directory(directory):
    return len([file for file in os.listdir(directory) if file.endswith('.jpg')])

train_fire_count = count_images_in_directory(train_fire_dir)
train_nofire_count = count_images_in_directory(train_nofire_dir)
test_total_count = count_images_in_directory(test_dir)

labels = ['Train Fire', 'Train No Fire', 'Test Total']
counts = [train_fire_count, train_nofire_count, test_total_count]

plt.figure(figsize=(12, 6))

# Bar chart
plt.subplot(1, 2, 1)
bars = plt.bar(labels, counts, color=['red', 'green', 'blue'])
plt.title('Image Distribution')
plt.xlabel('Category')
plt.ylabel('Count')

# Add counts on top of bars
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height, f'{height}', ha='center', va='bottom')

# Pie chart
plt.subplot(1, 2, 2)
plt.pie(counts, labels=labels, autopct='%1.1f%%', colors=['red', 'green', 'blue'])
plt.title('Image Proportion')

plt.show()
```



4. Function to Load Images from a Directory

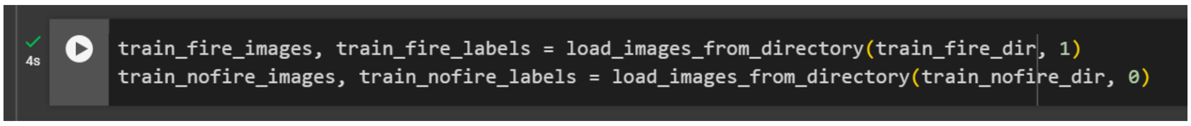
This function takes images from a specified directory, changes their size to make them the same, and gives them back along with their labels. The images and labels are turned into a numpy array that's easy for models to work with. So the images get ready before using them in a machine learning model.



```python
def load_images_from_directory(directory, label):
    images = []
    labels = []
    for filename in os.listdir(directory):
        if filename.endswith('.jpg'):
            img = cv2.imread(os.path.join(directory, filename))
            img = cv2.resize(img, (224, 224))
            images.append(img)
            labels.append(label)
    return np.array(images), np.array(labels)
```

5. Loading Training Data

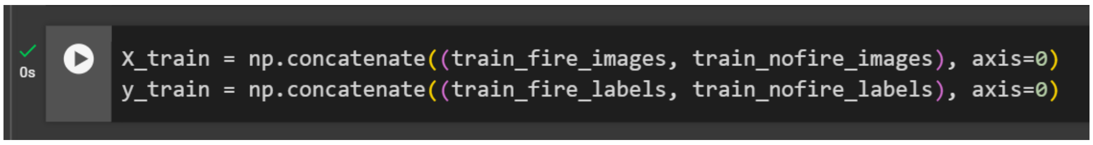
Now we load and label the training images from the directories specified earlier. The fire images are assigned a label of 1, and the no fire images are assigned a label of 0. So in training part the machine learning model to differentiate between fire and no fire scenarios.



```
train_fire_images, train_fire_labels = load_images_from_directory(train_fire_dir, 1)
train_nofire_images, train_nofire_labels = load_images_from_directory(train_nofire_dir, 0)
```

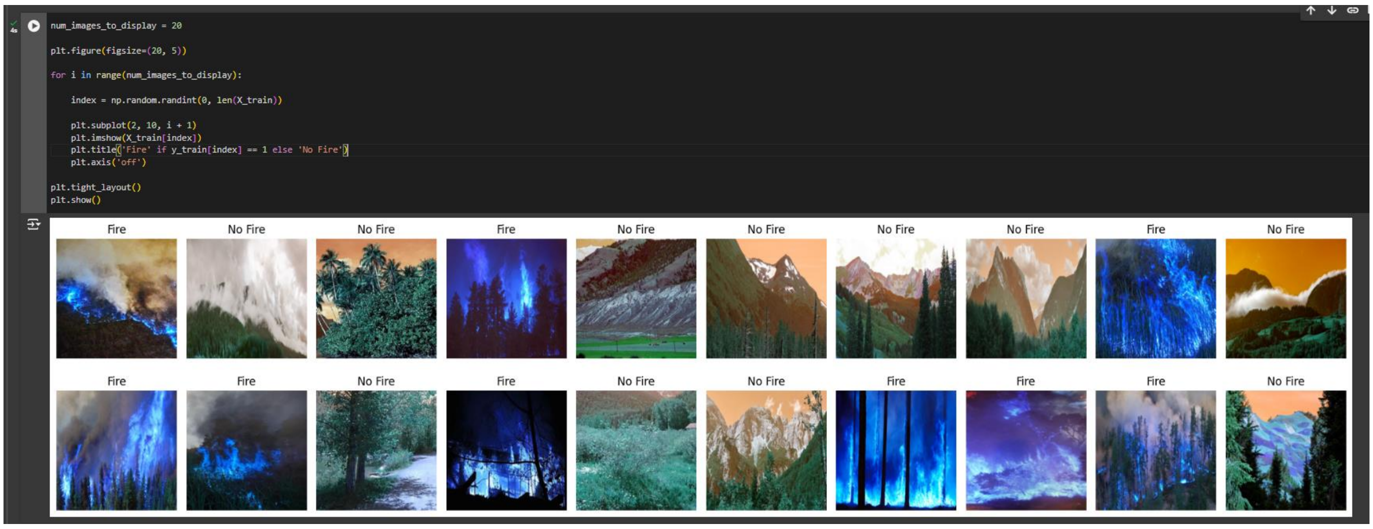
6. Combining the Fire and No Fire Data

These lines combine the fire and no fire images into one large NumPy array, and they also combine their labels into one large NumPy array. This makes it easier to use all the data together when training the machine learning model.



```
X_train = np.concatenate((train_fire_images, train_nofire_images), axis=0)
y_train = np.concatenate((train_fire_labels, train_nofire_labels), axis=0)
```

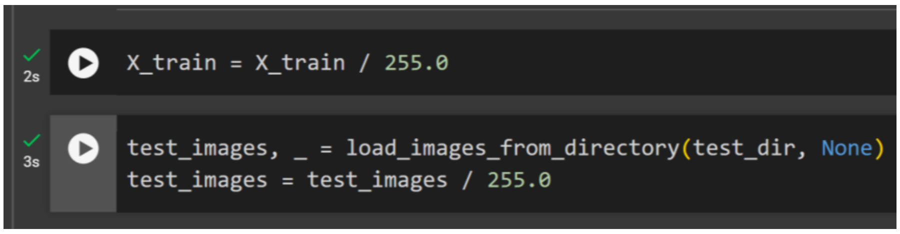
7. Visualizing the Training Data



```
num_images_to_display = 20

plt.figure(figsize=(20, 5))

for i in range(num_images_to_display):

    index = np.random.randint(0, len(X_train))

    plt.subplot(2, 10, i + 1)
    plt.imshow(X_train[index])
    plt.title('Fire' if y_train[index] == 1 else 'No Fire')
    plt.axis('off')

plt.tight_layout()
plt.show()
```

8. Normalizing the Images

Now I normalize both the training and test images by dividing their pixel values by 255, so the values range from 0 to 1 instead of 0 to 255. This helps the model work better with the images.

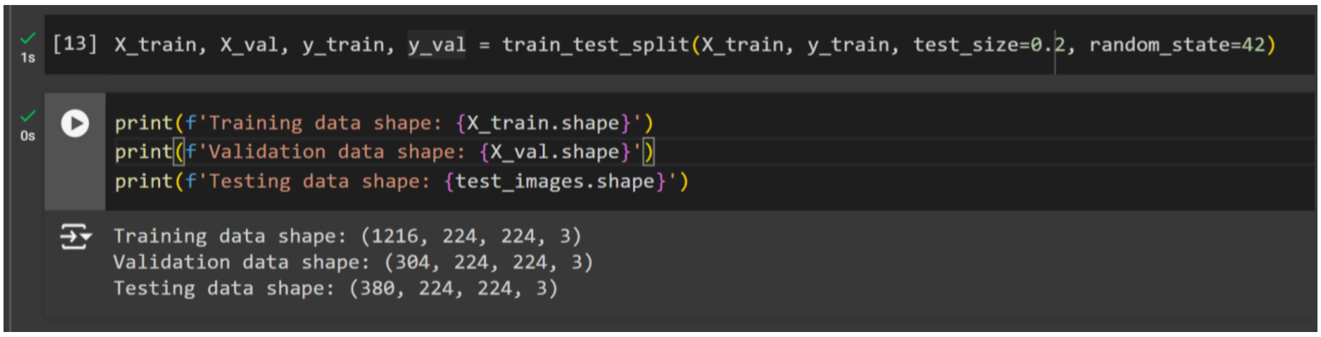


```
X_train = X_train / 255.0
```

```
test_images, _ = load_images_from_directory(test_dir, None)
test_images = test_images / 255.0
```

9. Split train data into training and validation sets

In this code, I split the data into training and validation sets, with 80% of the data used for training and 20% for validation. This helps train the model on one part of the data while checking how well it performs on the other part to avoid overfitting. After splitting, I printed out the sizes of the training, validation, and test sets to make sure everything looks right. The training set is used to teach the model, and the validation set is used to check its progress as it learns.



```
[13] X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

```
print(f'Training data shape: {X_train.shape}')
print(f'Validation data shape: {X_val.shape}')
print(f'Testing data shape: {test_images.shape}')
```

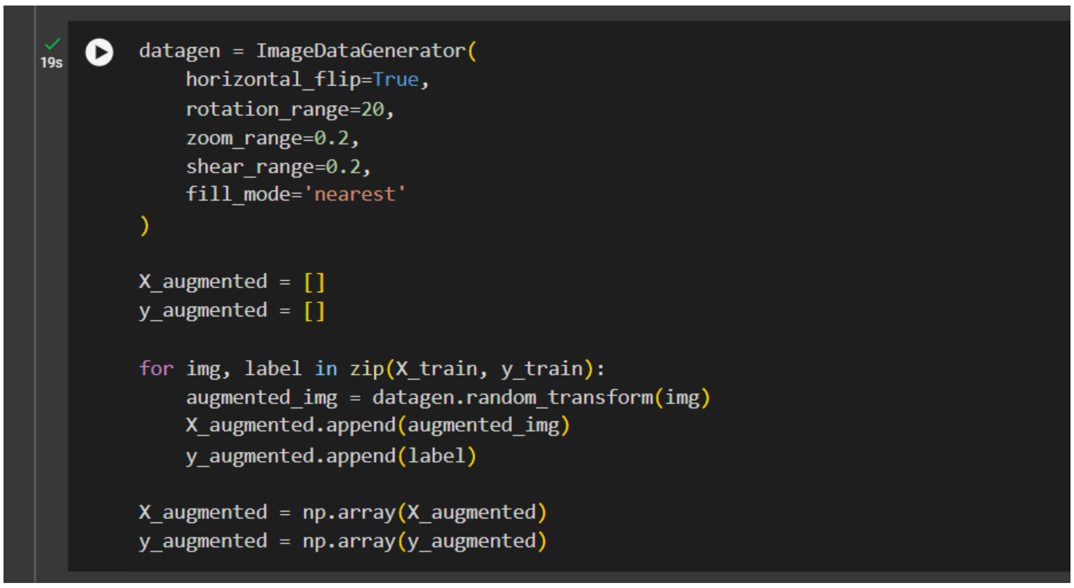
```
Training data shape: (1216, 224, 224, 3)
Validation data shape: (304, 224, 224, 3)
Testing data shape: (380, 224, 224, 3)
```

2. Building and Training the Fire Detection Model

I built a model using a Convolutional Neural Network to tell the difference between fire and no fire images. I used data augmentation techniques like rotating and flipping the images to make the model better. The model was trained over a few rounds to improve how well it can learn from the data.

1. augmenting the training data

This code creates new training images by randomly flipping, rotating, zooming, and shearing the original ones. We do this to make the model better at recognizing fire in different conditions by giving it more varied examples to learn. The augmented images help the model generalize better and not just memorize the original training images. This improves its performance on new, unseen data and reduces overfitting.

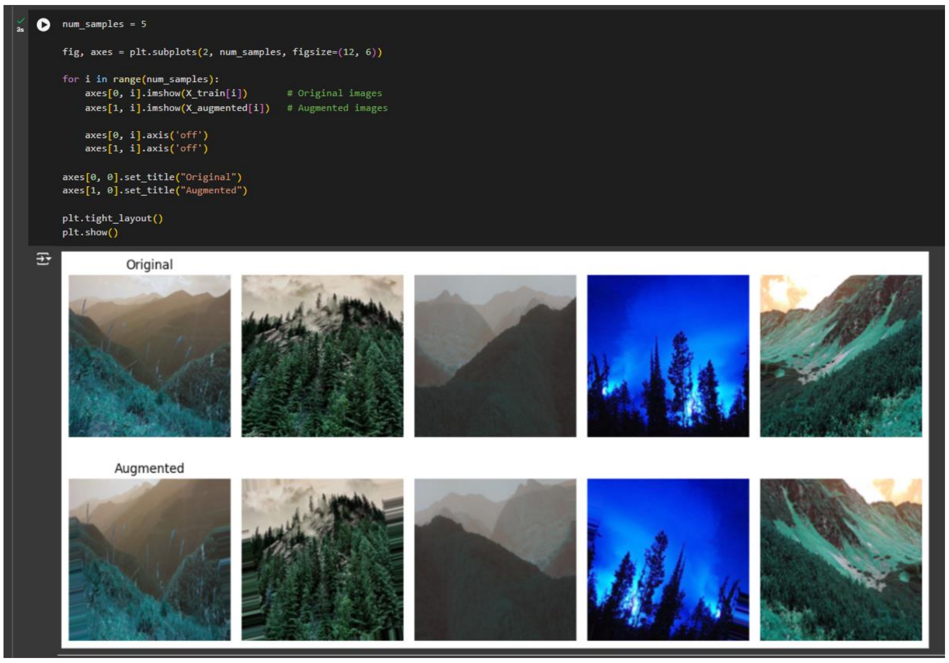


19s

```python
datagen = ImageDataGenerator(
    horizontal_flip=True,
    rotation_range=20,
    zoom_range=0.2,
    shear_range=0.2,
    fill_mode='nearest'
)

X_augmented = []
y_augmented = []

for img, label in zip(X_train, y_train):
    augmented_img = datagen.random_transform(img)
    X_augmented.append(augmented_img)
    y_augmented.append(label)

X_augmented = np.array(X_augmented)
y_augmented = np.array(y_augmented)
```

2.Visualizing Data Augmentation

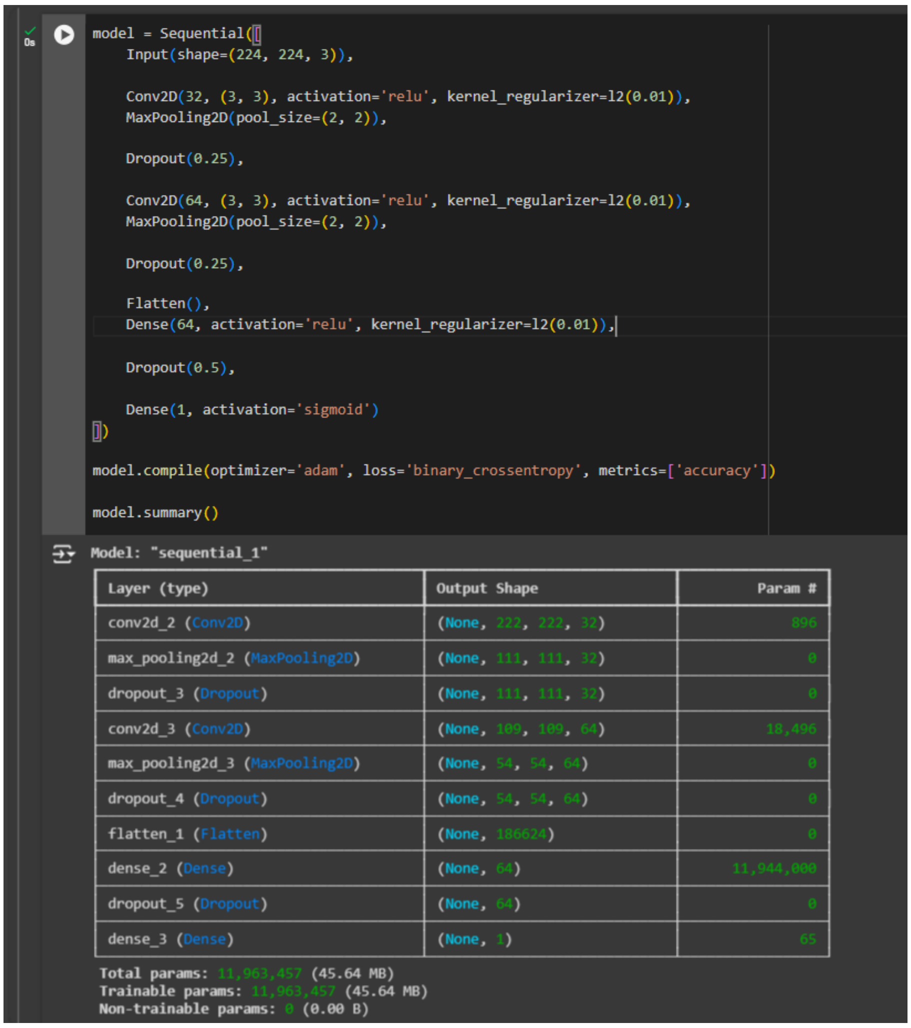
Then I visualized original images and the ones produced by data augmentation to verify the success of the process.



```
num_samples = 5

fig, axes = plt.subplots(2, num_samples, figsize=(12, 6))

for i in range(num_samples):
    axes[0, i].imshow(X_train[i])       # Original images
    axes[1, i].imshow(X_augmented[i])   # Augmented images

    axes[0, i].axis('off')
    axes[1, i].axis('off')

axes[0, 0].set_title("Original")
axes[1, 0].set_title("Augmented")

plt.tight_layout()
plt.show()
```

3. Building the CNN Model

This code builds a Convolutional Neural Network (CNN) to classify images as either fire or no fire. It starts by taking 224x224 color images as input. The model has two layers to detect features, each using filters to find patterns in the image, with 32 filters in the first layer and 64 in the second. Each of these layers is followed by a pooling layer to reduce the size and Dropout layers (25%) to prevent overfitting. Then, a Flatten layer turns the 2D data into a 1D vector to prepare for classification. A dense layer with 64 units follows, using ReLU activation, and has a 50% dropout rate to prevent overfitting. The last layer uses sigmoid activation to make a final decision between the two classes (fire or no fire). The model uses the Adam optimizer with binary cross-entropy loss and tracks accuracy. In total, it has about 11.96 million trainable parameters, which means it's a pretty complex model that can learn a lot from the image data.

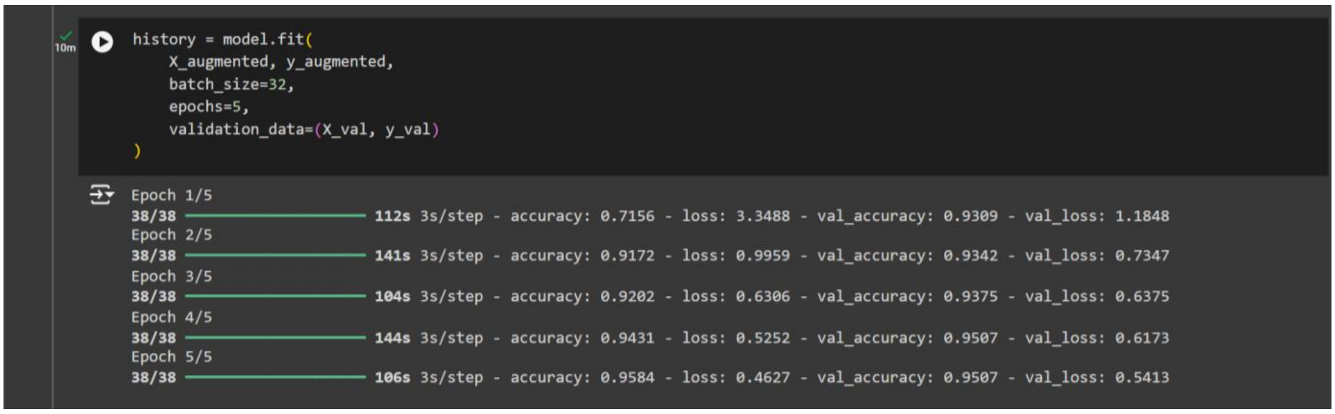


```
model = Sequential([
    Input(shape=(224, 224, 3)),

    Conv2D(32, (3, 3), activation='relu', kernel_regularizer=l2(0.01)),
    MaxPooling2D(pool_size=(2, 2)),

    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.01)),
    MaxPooling2D(pool_size=(2, 2)),

    Dropout(0.25),

    Flatten(),
    Dense(64, activation='relu', kernel_regularizer=l2(0.01)),

    Dropout(0.5),

    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_1"
┌──────────────────────────────┬──────────────────────────┬────────────┐
│ Layer (type)                 │ Output Shape             │    Param # │
├──────────────────────────────┼──────────────────────────┼────────────┤
│ conv2d_2 (Conv2D)            │ (None, 222, 222, 32)     │        896 │
├──────────────────────────────┼──────────────────────────┼────────────┤
│ max_pooling2d_2 (MaxPooling2D)│ (None, 111, 111, 32)    │          0 │
├──────────────────────────────┼──────────────────────────┼────────────┤
│ dropout_3 (Dropout)          │ (None, 111, 111, 32)     │          0 │
├──────────────────────────────┼──────────────────────────┼────────────┤
│ conv2d_3 (Conv2D)            │ (None, 109, 109, 64)     │     18,496 │
├──────────────────────────────┼──────────────────────────┼────────────┤
│ max_pooling2d_3 (MaxPooling2D)│ (None, 54, 54, 64)      │          0 │
├──────────────────────────────┼──────────────────────────┼────────────┤
│ dropout_4 (Dropout)          │ (None, 54, 54, 64)       │          0 │
├──────────────────────────────┼──────────────────────────┼────────────┤
│ flatten_1 (Flatten)          │ (None, 186624)           │          0 │
├──────────────────────────────┼──────────────────────────┼────────────┤
│ dense_2 (Dense)              │ (None, 64)               │ 11,944,000 │
├──────────────────────────────┼──────────────────────────┼────────────┤
│ dropout_5 (Dropout)          │ (None, 64)               │          0 │
├──────────────────────────────┼──────────────────────────┼────────────┤
│ dense_3 (Dense)              │ (None, 1)                │         65 │
└──────────────────────────────┴──────────────────────────┴────────────┘
 Total params: 11,963,457 (45.64 MB)
 Trainable params: 11,963,457 (45.64 MB)
 Non-trainable params: 0 (0.00 B)
```

4.Train the model

The model was trained for 5 epochs with bathes 32 images using augmented training data. The results show that the model was learning effectively and improving its ability to detect fire as the training progressed.

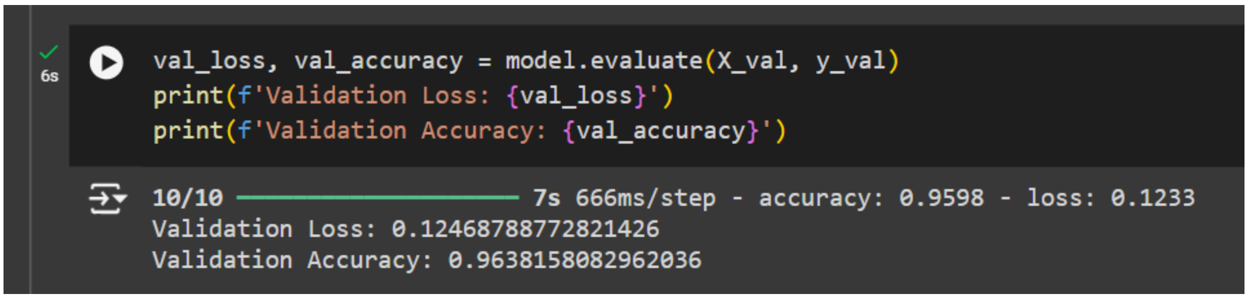


```
history = model.fit(
    X_augmented, y_augmented,
    batch_size=32,
    epochs=5,
    validation_data=(X_val, y_val)
)
```

```
Epoch 1/5
38/38 ━━━━━━━━━━━━━━━━━━━━ 112s 3s/step - accuracy: 0.7156 - loss: 3.3488 - val_accuracy: 0.9309 - val_loss: 1.1848
Epoch 2/5
38/38 ━━━━━━━━━━━━━━━━━━━━ 141s 3s/step - accuracy: 0.9172 - loss: 0.9959 - val_accuracy: 0.9342 - val_loss: 0.7347
Epoch 3/5
38/38 ━━━━━━━━━━━━━━━━━━━━ 104s 3s/step - accuracy: 0.9202 - loss: 0.6306 - val_accuracy: 0.9375 - val_loss: 0.6375
Epoch 4/5
38/38 ━━━━━━━━━━━━━━━━━━━━ 144s 3s/step - accuracy: 0.9431 - loss: 0.5252 - val_accuracy: 0.9507 - val_loss: 0.6173
Epoch 5/5
38/38 ━━━━━━━━━━━━━━━━━━━━ 106s 3s/step - accuracy: 0.9584 - loss: 0.4627 - val_accuracy: 0.9507 - val_loss: 0.5413
```

5. Evaluate the model on validation data

I used the validation data to evaluate how well the model performs on unseen data that was not part of the training process. This helps us understand how the model generalizes to new data, ensuring that it doesn't just memorize the training examples and we can have a final accuracy rate.



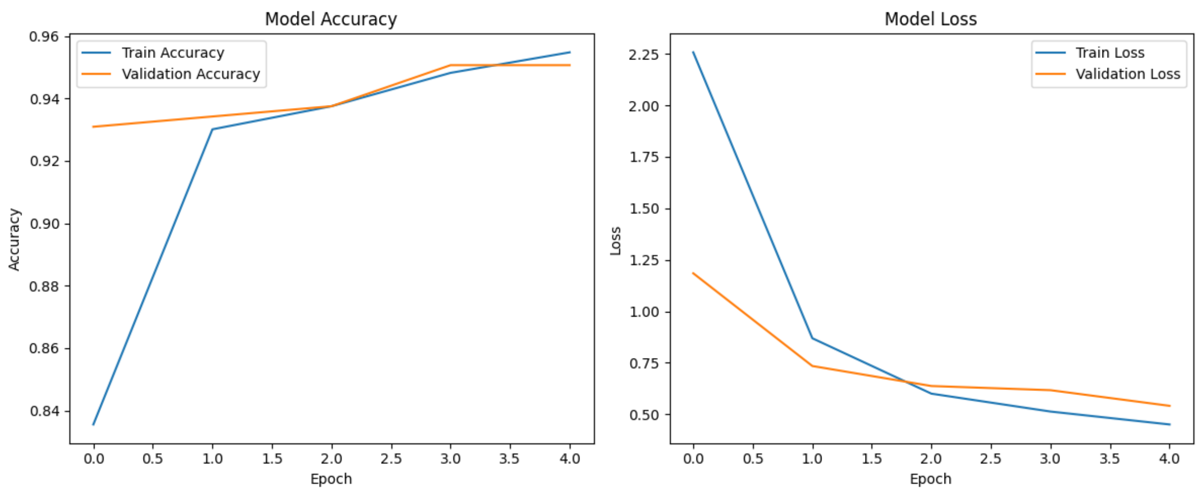
```
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f'Validation Loss: {val_loss}')
print(f'Validation Accuracy: {val_accuracy}')
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 7s 666ms/step - accuracy: 0.9598 - loss: 0.1233
Validation Loss: 0.12468788772821426
Validation Accuracy: 0.9638158082962036
```

6. Training Progress Visualization

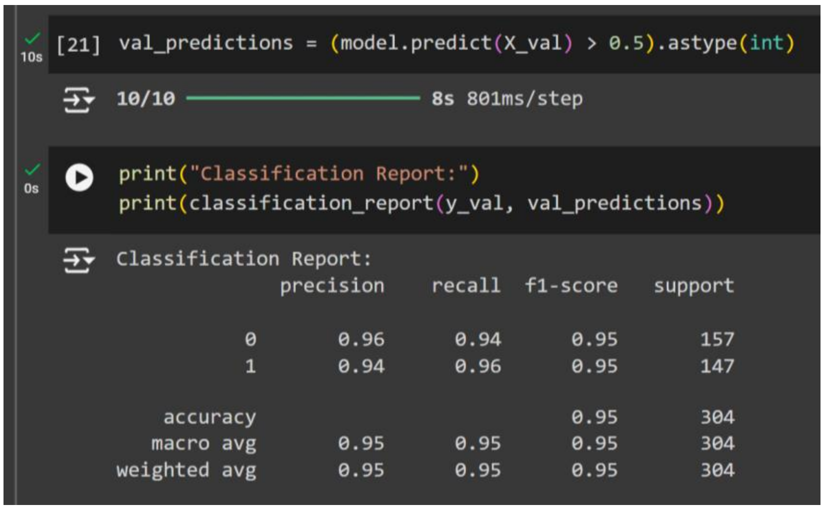
I created these plots to show how my model’s accuracy and loss changed during training.

- On the left, the "Model Accuracy" plot shows a steady improvement in training accuracy, with validation accuracy staying consistently high throughout. The close alignment of both lines indicates that the model is performing well on both the training and validation data.
- On the right, the "Model Loss" plot shows a strong decrease in loss for both training and validation over time. This suggests that the model is learning effectively and improving its performance with each epoch. Overall, the plots indicate that the model is training successfully and generalizing well.



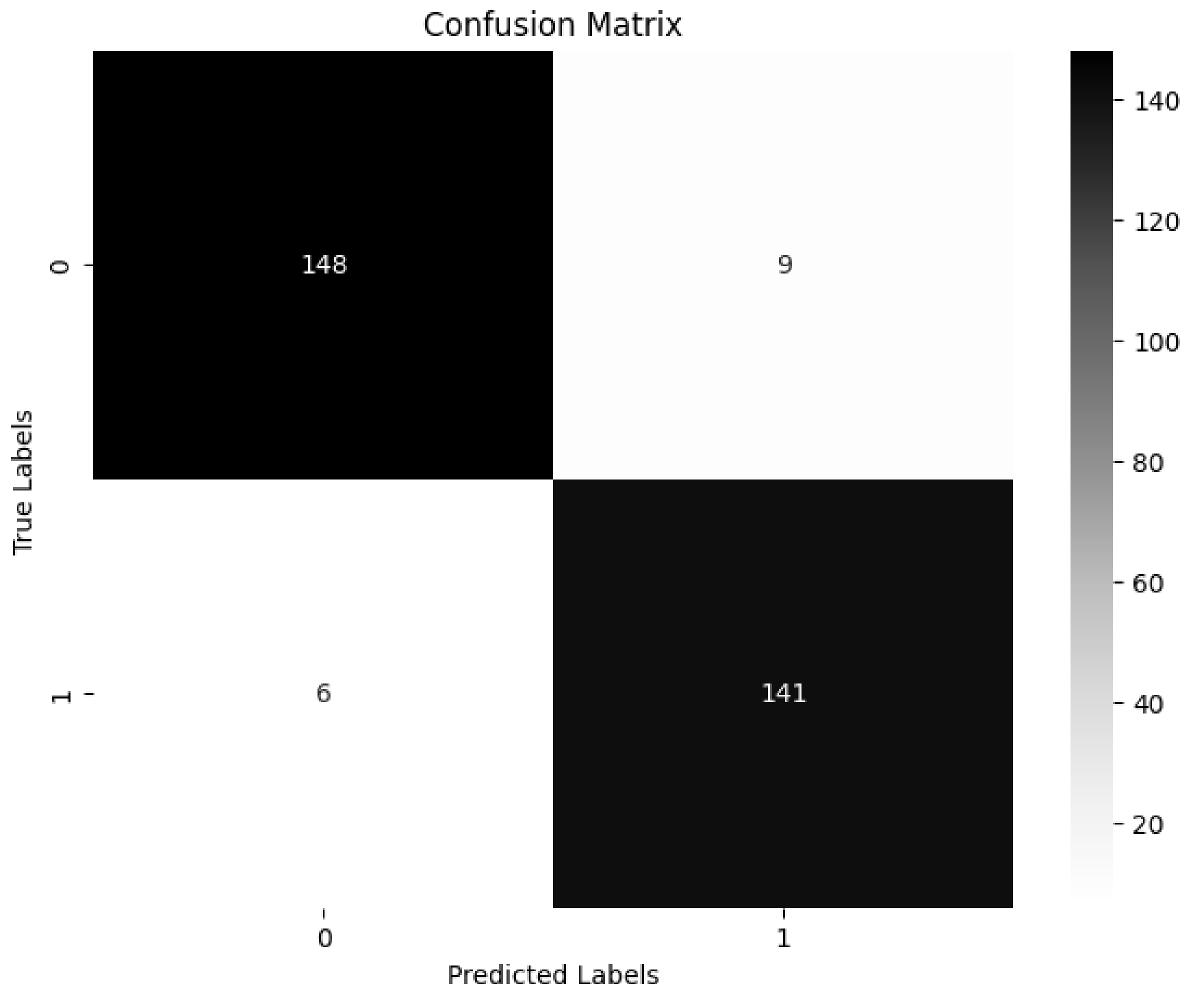
3. Evaluate Model Performance

In this step, I checked how well the model works by using validation data and predicted labels. The classification report gave me important metrics like precision, recall, f1-score, and accuracy, which helped me see how well the model identifies fire and no fire images. The confusion matrix also showed where the model made right or wrong guesses. These tools helped me understand how good the model is at making predictions.



```
[21] val_predictions = (model.predict(X_val) > 0.5).astype(int)

10/10 ━━━━━━━━━━━━━━━━━━━━ 8s 801ms/step

print("Classification Report:")
print(classification_report(y_val, val_predictions))

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.94      0.95       157
           1       0.94      0.96      0.95       147

    accuracy                           0.95       304
   macro avg       0.95      0.95      0.95       304
weighted avg       0.95      0.95      0.95       304
```

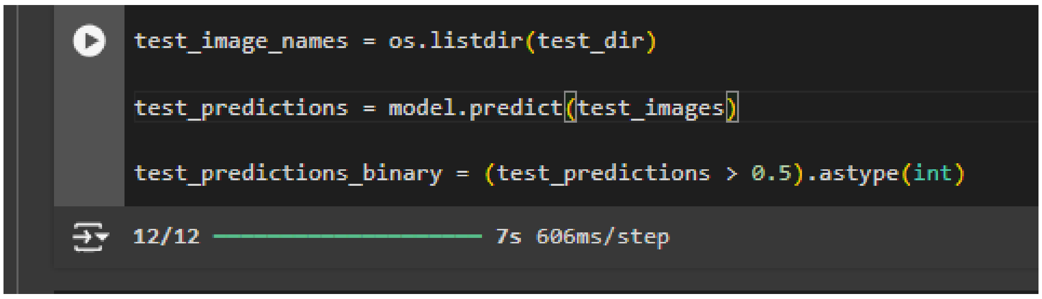
The confusion matrix shows that the model's predictions were mostly accurate. Out of 157 "no fire" images, the model correctly predicted 148 and missed 9. Similarly, for 147 "fire" images, the model correctly identified 141 and made only 6 mistakes. This means the model did well in distinguishing between fire and no fire, with very few errors.



5. Drone Deployment and Fire Suppression

1. Predicting Fire Risk in Test Images

This code gets the names of new test images that haven't been used before to act like real-world situations. The model predicts if there's a fire in these images. The predictions are turned into simple "yes" or "no" (fire or no fire). I did this to find the places with the highest fire risk and use the results in the next steps, like sending a drone there.


```
test_image_names = os.listdir(test_dir)

test_predictions = model.predict(test_images)

test_predictions_binary = (test_predictions > 0.5).astype(int)
```
```
12/12 ━━━━━━━━━━━━━━━━━━━━ 7s 606ms/step
```

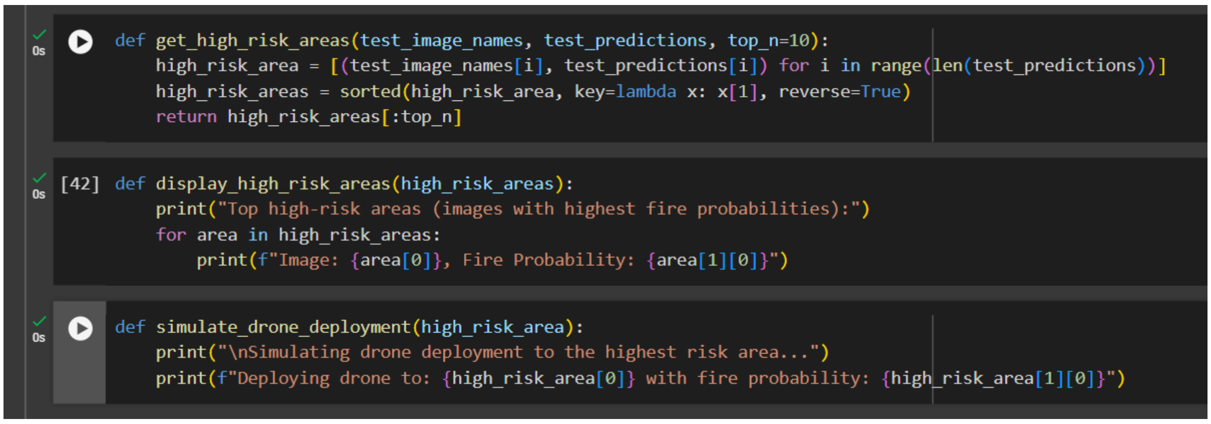
2. Defining Fire Response Functions

Then I made three functions to find, show, and act on high-risk fire areas.

1. The first function, get_high_risk_areas, finds the areas with the highest chance of fire by matching each test image with its fire prediction. It gives back the top areas.

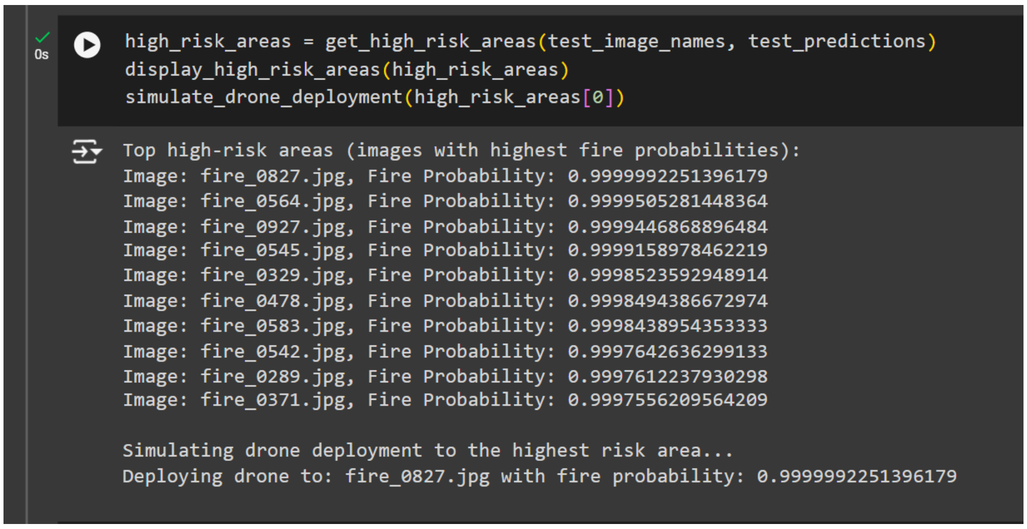
2. The second function, display_high_risk_areas, shows the names and fire probabilities of these high-risk areas, so I can see which places need attention.

3. The third function, simulate_drone_deployment, simulates sending a drone to the area with the highest fire risk, showing how the system can act automatically


```
def get_high_risk_areas(test_image_names, test_predictions, top_n=10):
    high_risk_area = [(test_image_names[i], test_predictions[i]) for i in range(len(test_predictions))]
    high_risk_areas = sorted(high_risk_area, key=lambda x: x[1], reverse=True)
    return high_risk_areas[:top_n]
```
```
[42] def display_high_risk_areas(high_risk_areas):
    print("Top high-risk areas (images with highest fire probabilities):")
    for area in high_risk_areas:
        print(f"Image: {area[0]}, Fire Probability: {area[1][0]}")
```
```
def simulate_drone_deployment(high_risk_area):
    print("\nSimulating drone deployment to the highest risk area...")
    print(f"Deploying drone to: {high_risk_area[0]} with fire probability: {high_risk_area[1][0]}")
```

3. Identifying High-Risk Areas and Drone Deployment Simulation

I used the functions I created to find and show the areas most at risk of fire. The output gave me a list of images with the highest chances of having a fire. The simulation then sent a drone to the area with the highest risk, where the fire probability was 0.99999. This result helps to know which areas need quick action.

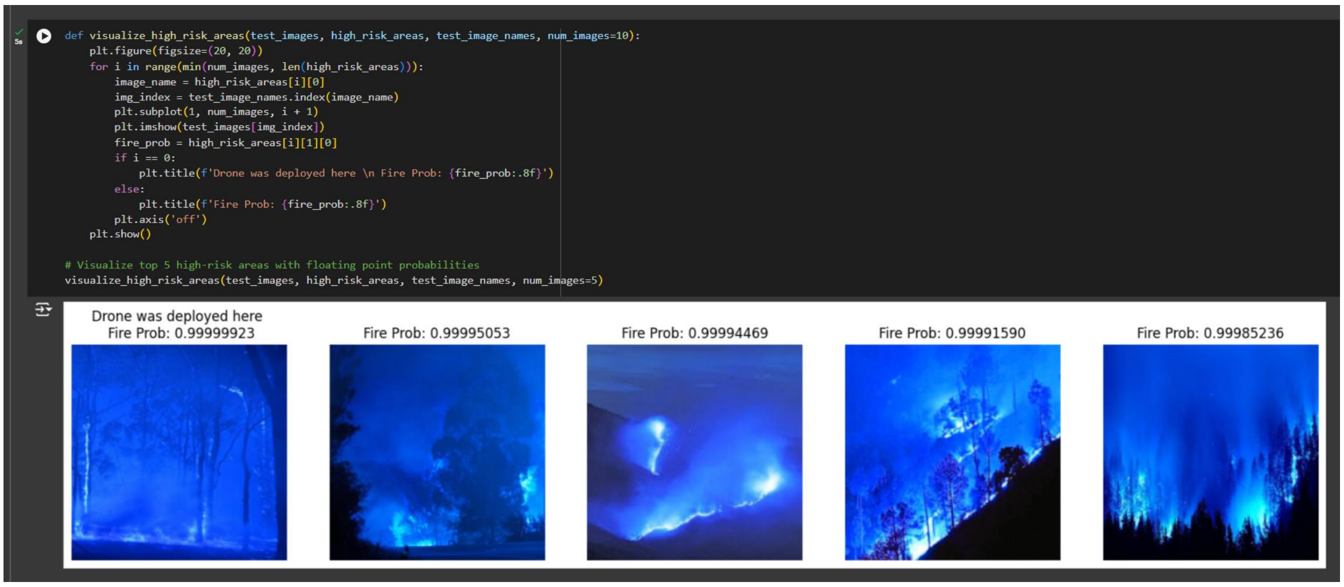


```
high_risk_areas = get_high_risk_areas(test_image_names, test_predictions)
display_high_risk_areas(high_risk_areas)
simulate_drone_deployment(high_risk_areas[0])
```

```
Top high-risk areas (images with highest fire probabilities):
Image: fire_0827.jpg, Fire Probability: 0.9999992251396179
Image: fire_0564.jpg, Fire Probability: 0.9999505281448364
Image: fire_0927.jpg, Fire Probability: 0.9999446868896484
Image: fire_0545.jpg, Fire Probability: 0.9999158978462219
Image: fire_0329.jpg, Fire Probability: 0.9998523592948914
Image: fire_0478.jpg, Fire Probability: 0.9998494386672974
Image: fire_0583.jpg, Fire Probability: 0.9998438954353333
Image: fire_0542.jpg, Fire Probability: 0.9997642636299133
Image: fire_0289.jpg, Fire Probability: 0.9997612237930298
Image: fire_0371.jpg, Fire Probability: 0.9997556209564209

Simulating drone deployment to the highest risk area...
Deploying drone to: fire_0827.jpg with fire probability: 0.9999992251396179
```

4. Visualizing High-Risk Fire Areas

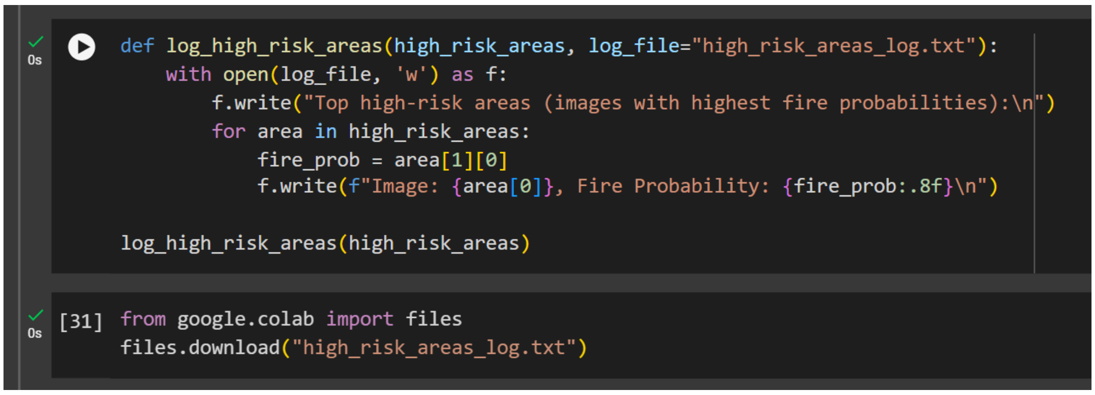
Here i displayed the areas most likely to have fires and the area which the drone was deployed, showing images with their fire probabilities. This allows me to understand how confident the model is in its predictions. It helps to verify how well the model identifies high-risk fire areas for drone deployment.



```
def visualize_high_risk_areas(test_images, high_risk_areas, test_image_names, num_images=10):
    plt.figure(figsize=(20, 20))
    for i in range(min(num_images, len(high_risk_areas))):
        image_name = high_risk_areas[i][0]
        img_index = test_image_names.index(image_name)
        plt.subplot(1, num_images, i + 1)
        plt.imshow(test_images[img_index])
        fire_prob = high_risk_areas[i][1][0]
        if i == 0:
            plt.title(f'Drone was deployed here \n Fire Prob: {fire_prob:.8f}')
        else:
            plt.title(f'Fire Prob: {fire_prob:.8f}')
        plt.axis('off')
    plt.show()

# Visualize top 5 high-risk areas with floating point probabilities
visualize_high_risk_areas(test_images, high_risk_areas, test_image_names, num_images=5)
```

5. Logging and Downloading High-Risk Fire Areas

By logging the high risk areas with the highest fire probabilities into a text file, I can create a clear record of where the most dangerous areas are. In real life, this can help guide drones or firefighters to focus on these critical locations. The logged information can be shared with a team on the ground, ensuring they know where to deploy resources to prevent or minimize fire damage effectively. The ability to download and share this data makes it easier to coordinate efforts in real-time during fire emergencies.



```
def log_high_risk_areas(high_risk_areas, log_file="high_risk_areas_log.txt"):
    with open(log_file, 'w') as f:
        f.write("Top high-risk areas (images with highest fire probabilities):\n")
        for area in high_risk_areas:
            fire_prob = area[1][0]
            f.write(f"Image: {area[0]}, Fire Probability: {fire_prob:.8f}\n")

log_high_risk_areas(high_risk_areas)
```

```
[31] from google.colab import files
     files.download("high_risk_areas_log.txt")
```

5. Results

The model I built did a good job of detecting fires in images. After training with fire and no fire pictures, the model was able to correctly identify most of the images. It showed strong results in accuracy, precision, and recall, meaning it can tell the difference between fire and no fire well. I also showed the top high risk areas with their fire probabilities, which helped me see how confident the model was in its predictions.

The drone deployment simulation worked well too. The model's predictions helped decide which high risk areas the drone should be sent to, showing how this system can be used in real life to quickly find and stop fires.

6. Conclusion

This project successfully used machine learning and drones to find and manage forest fires more quickly. The model did a good job at spotting fires, and the drone simulation showed how it could respond fast to dangerous areas. This system can help improve how we deal with fires, protecting nature, animals, and people.
The system could get better by looking at things like wind, how many trees are in an area, what kind of trees (if they catch fire easily), and how close the fire is to people. These improvements would make the system smarter and help protect the environment and people even more.

7. References

- Google scholar
- Google Colab
- Python
- NumPy
- OpenCV (cv2)
- Scikit-learn
- TensorFlow/Keras
- Matplotlib
- Kaggle