Tensor Forest

Arnnav Kudale, Shayaan Azeem, Rohanth Marem

About the team

We're a team from Oakville, Ontario, Canada. Our team consists of three International Baccalaureate students. We met through our common interest in robotics and engineering at our school's engineering club. Shayaan (center) is in grade 11, whilst Rohanth (left) and Arnnav (right) are in grades 10 and 9 respectively. We are all passionate about all kinds of technology and the value it can provide us. In our spare time, we enjoy watching Formula 1, 3D printing, and playing board games.

We divided the work for this project based on our areas of expertise. Arnnav focused on the hardware aspect of the project, including flight systems, drone navigation, and automation. Shayaan researched the forest fire detection mitigation models and developed our flammability level detection model. Rohanth focused on running the model on a local TPU and creating a map output to provide the user with the final information. We integrated our systems together to create “TensorForest”.

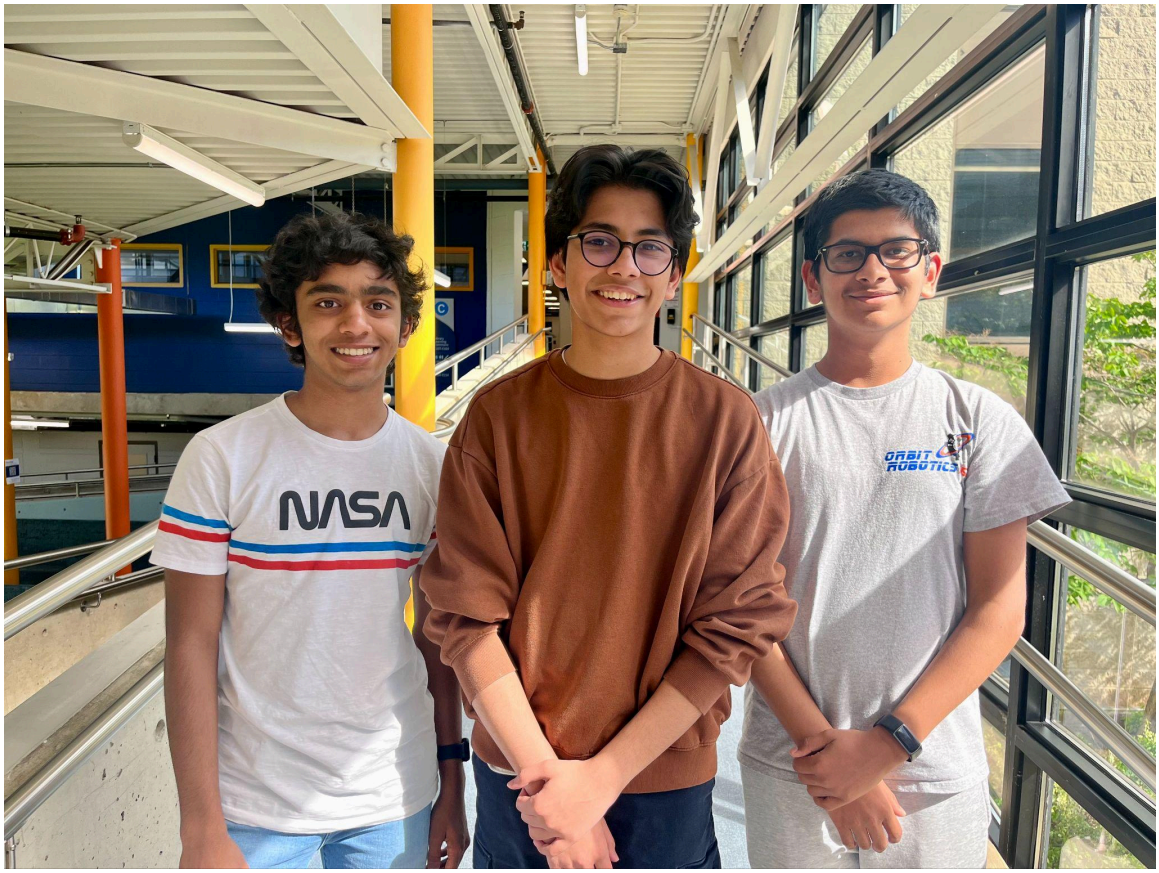


Figure #1: Rohanth (Left), Shayaan (Center), Arnnav (Right)

Executive Summary

Over the past years, wildfires have become more frequent and severe. They destroy precious forest land, ecosystems, and contribute to a significant amount of greenhouse gases, causing additional global warming and respiratory illness for those in nearby communities. The UN Environment Programme (UNEP) predicts a global increase of extreme fires of up to 14 percent by 2030, 30 percent by the end of 2050, and 50 percent by the end of 2100. We chose this problem due to its increasing threat to environmental sustainability and human health. We felt our expertise and passion for innovation in this field positioned us to tackle this challenge.

While it's not possible to completely eliminate the risk of wildfires, much can still be done to manage and reduce risks. The most effective method to mitigate fire risk is active and aware management of the available vegetation/fuel on a given site before a wildfire breaks out. However, it is extremely labor, cost, and time-intensive to conduct frequent manual on-site surveys and measure the amount and types of fuel available on a site, especially since it changes over a season with precipitation and weather changes.

Our solution, TensorForest, is an automated drone designed to capture aerial photographs of forests and compile high-resolution imagery. This imagery is used to detect various types of vegetation and assess their flammability levels. The resulting flammability map identifies areas of continuous flammable vegetation and pinpoints the locations most at risk of fire, potentially indicating where a fire is most likely to start. The drone's ability to create detailed flammability maps is crucial for disrupting patterns of vegetation continuity that could allow the spread of fires. It also supports fire management planning, such as optimizing the placement of firefighting resources (i.e., water reserves). The use of drones reduces the need for costly, labor-intensive manual surveys, enabling more frequent and expansive monitoring at a lower cost. This automation allows for more accurate and up-to-date data collection, which allows for better forest fire prevention.

If further developed, TensorForest can be the cost-effective starting point for all property owners starting their wildfire management workflow, as given a route, it can fly automatically regularly over at-risk areas to update flammability maps in real-time. This data would be accessed by property owners to make informed decisions on vegetation management and firefighting logistics.

TensorForest is crucial not just for mitigating the immediate risks of wildfires but also for its broader impact as it reduces carbon emissions and protects ecosystems. It also offers a scalable solution that can be adapted for different regions and types of vegetation, potentially serving as a global standard for wildfire risk assessment and management.

Presentation of the Robotic Solution

Technical Approach and Methodology

After identifying the issue we wanted to tackle—which was better and more accessible data collection for forest fire mitigation—we began examining current technology and solutions. We discovered that the primary methods for creating forest vegetation maps and flammability heat maps involved: firstly, manual data collection, which is extremely time and labor intensive and becomes outdated very quickly; and secondly, the use of satellite imagery by various organizations and fire departments. However, this method only offers a maximum resolution of 1 kilometer, which means the smallest detail it can resolve is about 10 meters, rendering the imagery less effective due to its limited accuracy.

However, using drones could dramatically improve this situation by providing imagery with a resolution of 1 meter, which would be a huge increase in the accuracy of such vegetation maps, especially beneficial for smaller property owners, like a farmer in rural Ontario. Initially, we also explored using the Canadian Forest Fire Weather Index, part of the Canadian Forest Fire Danger Rating System (CFFDRS), which assesses the risk of a forest fire in a specific location but, we quickly realized that this system, originally developed in 1984, is outdated and lacks the responsiveness required by modern science, which could accelerate the process of mitigating fires. The system, while useful for forest rangers in Alberta, fails to adapt to meet the needs of property owners and farmers who need to prevent wildfires on their lands.

For these reasons, our solution proved much more helpful. It does not require daily manual input of weather, precipitation, or moisture data. We developed a more simplified and adaptable system. Our high-resolution drone imagery processed through an AI-driven object identification model, classifies vegetation types and generates detailed flammability heat maps. This automation enhances both the speed and accuracy of the data analysis. Our maps clearly identify when and where there is a need to prune specific trees, thanks to the continuity in the presence of flammable tree species, making our system much more primal and easy to use.

Species	Flammability	Fuel Type
Black spruce	EXTREME	C2
Cured/dead grass and slash	EXTREME	01,S1, S2, S3
Lodgepole or jack pine	HIGH	C3
White spruce	HIGH	M1, M2
Western larch	LOW	C1
Young and mature aspen (has clean forest floor present)	VERY LOW	D1

Figure #2: Relationship between common north american forest fuel types and flammability levels

Construction of Solution

We began with the assembly of the chassis of our drone, which consists of a prebuilt carbon fiber Turnigy Talon Tricopter V1 kit. We chose to build a tricopter solution because of its dynamic yaw control, which is controlled by a servo that tilts one of the motors left and right. This allows for easy and precise adjustment against different wind conditions. The drone is run off three DJI Brushless motors. Brushless motors were chosen due to their efficiency, precision, and superior higher speeds compared to brushed motors, and also because they were easily accessible.

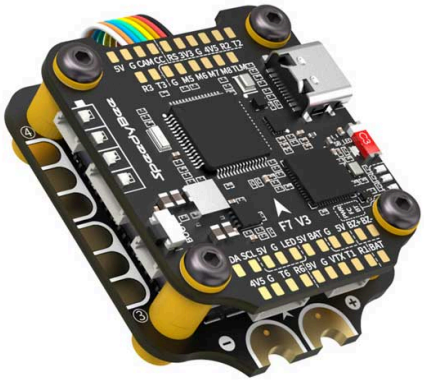
Figure #3: Completed Assembly of TensorForest

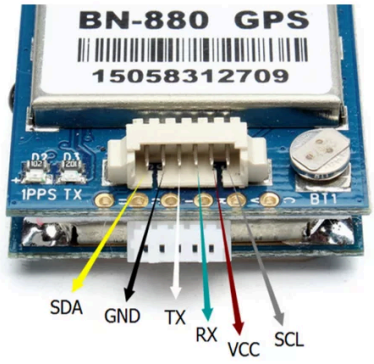
We opted for the SpeedyBee F405 FC as our flight controller, recognizing its high processing power and compatibility with Betaflight software. The F405 stands out for tricopters, particularly because of its precision in yaw adjustments. With its array of UART ports, the F405 not only provides extensive telemetry but also enables real-time flight data monitoring through its OSD capabilities, which is crucial for adapting quickly under varying conditions. Its GPS functionality further adds advanced navigational and safety features to our setup.

Integrating the radio controller with the current firmware and receiver proved to be a challenging task, as the software we initially used did not support the connection. To overcome this hurdle, we employed an Arduino to facilitate the conversion from CPPM/PPM to SBUS. This workaround not only solved the compatibility issue but also ensured a reliable communication link between our radio controller and the drone, which is critical for maintaining control and responsiveness during flight.

With the FC finally in place and the radio controller connection stabilized, we proceeded to install the BN-880 GPS Compass module. This component significantly enhances our navigational accuracy, providing dependable GPS and compass data that support advanced autonomous flight capabilities, such as waypoint navigation, return-to-home (RTH), and position hold. These features are crucial for improving both the precision and safety of the drone, enabling it to autonomously navigate back to a home position in case of any signal loss or emergency scenarios.

Linking the BN-880 with the SpeedyBee F405 FC via a UART port streamlines the setup and simplifies the configuration process in Betaflight. The integration enriches the OSD with essential flight metrics such as altitude, speed, and location, vital for the autopilot functionalities we aim to develop further.



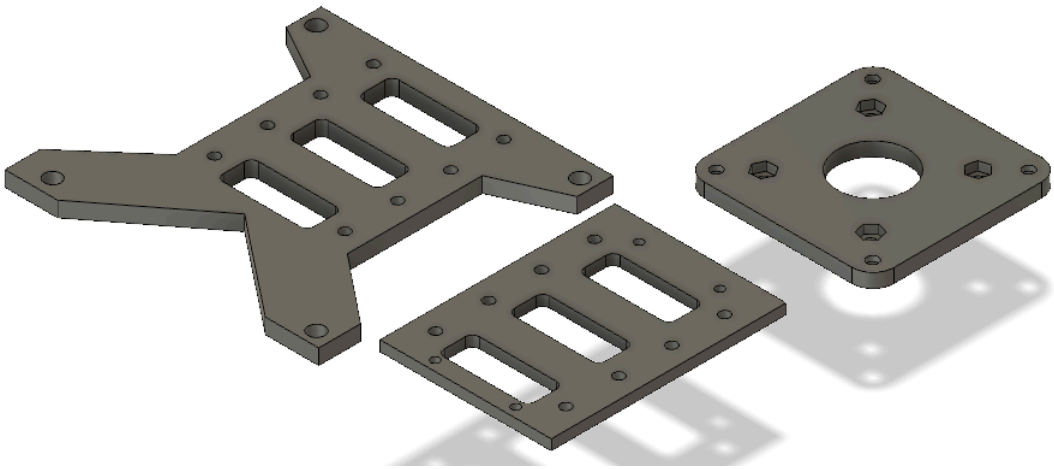


Figue #4: Speedybee F405 FC

Figure #5: BN880 GPS

We further enhanced our drone's capabilities by incorporating a Raspberry Pi Zero W into the configuration to execute autopilot software. The Raspberry Pi Zero W, notable for its compact form and Wi-Fi capability, serves as a companion computer. This allows us to extend the functionalities of our drone well beyond basic flight control. By integrating the Raspberry Pi Zero W with the SpeedyBee F405 FC, the Raspberry Pi handles complex navigational and automated flight tasks, freeing the F405 to focus on maintaining stable, real-time flight control. This division of labor between the Raspberry Pi and the F405 allows for the handling of advanced features without burdening the flight controller, ensuring both efficient processing and stable flight.

We also designed and created custom 3D-printed mounts to securely hold both the battery and our camera module in place. These mounts were engineered for optimal stability and streamlined integration with the drone's overall design.



The camera module is built around a Raspberry Pi 4 Model B with 4GB RAM and features a Google Coral Edge TPU, which is connected to a Raspberry Pi Camera Module V2. This setup enables the execution of a sophisticated tree class identification model in real-time. The model is designed to analyze live video feeds to detect and classify different types of vegetation captured by the camera, assessing their flammability levels based on predefined criteria. This functionality is crucial for environmental monitoring and managing fire risks in natural settings.

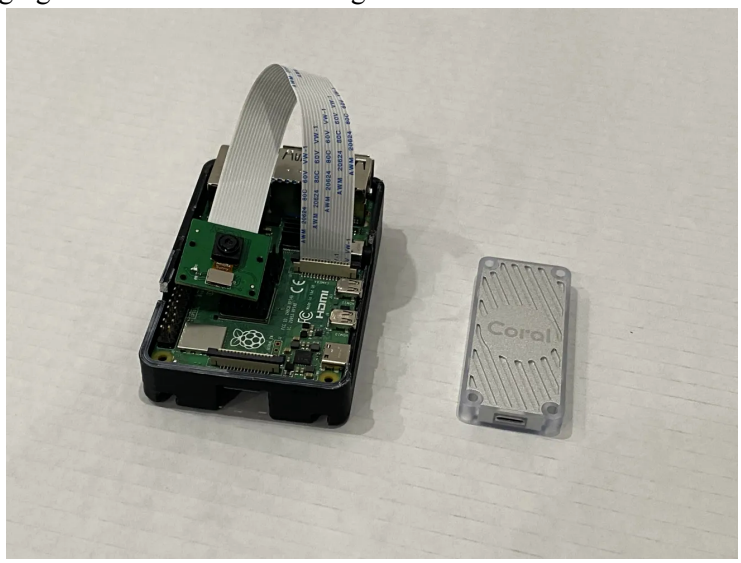


Figure 6: Raspberry Pi Integrated Unit with Coral Edge TPU

Additionally, this same Raspberry Pi 4 serves as an interactive hub for users. Through a local host connection, users can access a detailed map generated from the collected data, allowing for deeper analysis and insights. This dual-purpose use of the Raspberry Pi 4 not only maximizes efficiency but also enhances the drone's utility for field operations.

Software Component

To make the drone autonomous, we used ArduPilot. ArduPilot is an open-source project maintained by FPV and autonomous drone enthusiasts. It is a firmware that allows autonomous flight on any drone and enables you to add a Raspberry Pi for faster flight data readings. ArduPilot was used in this case as it is heavily documented, well-updated, and has a huge community around it, so finding help or solving issues wouldn't be a problem.

For flight planning, we used Mission Planner, one of the world's most used Flight Planning software used by professionals and DIY enthusiasts alike. It allows us to map out areas, route them automatically, and send the routes to our Flight Controller instantly. ArduPilot and Mission Planner are built for each other and work flawlessly together.

Figure 7: Mission Planner Software

Machine Learning Model

The machine learning model's software portion has four distinct segments that work in tandem to create a processed output heatmap of the fuel in a given forested region. The first script is used to capture images from the drone when flying using a Raspberry Pi Cam V2. The duration of the image capture, along with the interval at which the images are taken, can be adjusted to the user's liking or use the predetermined values.

These images are then stitched together using the OpenCV library to create one large image of the forested region. This image is then put through the inference/classification model run through the Coral Edge TPU, and the output is processed to create a heatmap.

Figure 8: Sample Stitched Image Output

Image Capture:

The image_capture.py script creates a time lapse image series on the Raspberry Pi. It interacts with the Raspberry Pi's camera module and captures images at user-specified intervals. The captured images are then used to create a timelapse image series which can be used to stitch together the image of the whole forest.

```
# Import necessary libraries
from picamera2 import Picamera2, Preview
from libcamera import controls
import time
from time import sleep
import os

# Initialize the camera
picam2 = Picamera2()

# Create a configuration for the camera preview
camera_config = picam2.create_preview_configuration()

# Ask the user for the duration of the timelapse in minutes
duration = input("Enter the duration of the timelapse in minutes: ")
# Ask the user for the interval between each image capture in seconds
interval = input("Enter the interval in which images are taken in seconds: ")
# Ask the user if they want to enable live preview
preview = input("Enable live preview? (0 for no, 1 for yes): ")
# Ask the user for the path to the folder where the images will be saved
path = input("Enter path to save folder: ")

# If the provided duration is not a valid number, set it to a default value of 10
minutes
if duration.isdigit() is False:
    duration = 10 * 60
else:
    duration = int(duration) * 60

# If the provided interval is not a valid number, set it to a default value of 5
seconds
if interval.isdigit() is False:
    interval = 5
else:
    interval = int(interval)

# If the provided preview option is not a valid number, set it to a default value of
0 (no preview)
if preview.isdigit() is False:
    preview = 0
else:
    preview = int(preview)
```

```
# If the provided path does not exist, ask the user if they want to use a default
path
if os.path.exists(path) is False:
    error = int(input("Path does not exist, do you want to use default path? 0 for
no, 1 for yes: "))
    if error == 1:
        path = "/home/pi/Documents/image_capture/images"
    else:
        quit()

# Configure the camera with the created configuration
picam2.configure(camera_config)
# If the user chose to enable live preview, start the preview
if preview == 1:
    picam2.start_preview(Preview.QTGL)
# Start the camera
picam2.start()

# Calculate the end time of the timelapse
end = time.time() + duration
# Calculate the total number of images that will be captured
total_image_count = duration // interval
# Initialize a counter for the number of images taken
images_taken = 0
# Record the start time of the timelapse
start_time = time.time()

# Loop until all images have been captured
while images_taken < total_image_count:
    # Create a label for the current date and time
    date_label = (time.strftime("%y-%m-%d_%H:%M:%S"))
    # Capture an image
    r = picam2.capture_request()
    # Save the image with the date label in the filename
    r.save("main", f"{path}/{date_label}.png")
    # Release the capture request
    r.release()
    # Increment the counter for the number of images taken
    images_taken += 1
    # Print the number of images taken and the time elapsed
    print(f"Captured image {images_taken} of {total_image_count} at {time.time() -
start_time:.2f}s")
    # Wait for the specified interval before capturing the next image
    time.sleep(interval - 0.0255)

# Print a message indicating that all images have been captured
print("Captured all images.")

# Stop the camera
picam2.stop()
```

Figure 9: Image Capture Script

Image Stitching:

The image_stitch.py script stitches multiple images together to create a single, larger image. All the captured images from the drone flight are put in a directory in which this script runs. It looks for common features between each image and combines them on that basis. This script works effectively when there is a significant overlap between the images, and this occurs when the robot takes its image timelapse of the forest region. It does this through various methods, such as feature detection and description, feature matching, homography estimation and image manipulation. These methods allow OpenCV to find regions with an image such as corners and edges which bleed into other images, allowing it to match them up. If two images are matched closely, Homography determines how to warp and manipulate an image to best align with the overlapping regions. By relying on these methods across images, OpenCV can create a cohesive stitched image even if the original images have slightly different viewpoints or lighting conditions. The stitching in the script is done using the Stitcher class from the OpenCV library. The stitched image is then saved to the disk. The output of this file is then given to the inference model to detect all the classes of vegetation fuel that exist within the given forest region.

```python
import cv2
import numpy as np
import os

# Load the images
image_folder = 'unannotated/'
image_files = sorted([f for f in os.listdir(image_folder) if
os.path.isfile(os.path.join(image_folder, f))])  # Sort the filenames alphabetically

images = [cv2.imread(os.path.join(image_folder, f)) for f in image_files]

# Initialize the stitcher
stitcher = cv2.Stitcher_create()

# Stitch the images together
status, stitched_image = stitcher.stitch(images)

# Check if the stitching was successful
if status == cv2.Stitcher_OK:
    # Save the stitched image
    cv2.imwrite('stitched_image.jpg', stitched_image)
else:
    print('Error during stitching, status code =', status)
```

Figure 10: Image Stitching Script

Inference Model:

One of the important features of the TensorForest model lies in the inference model script which automates object detection on a series of images using pre-trained models. It leverages TensorFlow Lite for efficient on-device processing. The script begins by initializing two object detection models. These models are pre-trained convolutional neural networks (CNNs) capable of identifying specific objects within images, trained on a publicly available dataset on Roboflow. The dataset was then trained on the YOLOv8 (You Only Look Once version 8) model to be custom tailored for forest fuel vegetation detection. The model is then loaded from optimized .tflite files. The script iterates through each image in a given folder and runs it through the mode using the .predict() method .The results of this are then stored in a separate directory, with bounding boxes and associated class labels for each identified object, are then saved to disk for further analysis.

```
# Import the necessary libraries
from ultralytics import YOLO
import os
import time

# Load the YOLO model from the specified .tflite file
model = YOLO('/home/pi/Documents/tensor_forest/best_full_integer_quant_edgetpu.tflite')

# Specify the directory where the images are stored
directory = '/home/pi/Documents/tensor_forest/testv2'

# Iterate over the files in the specified directory
for filename in os.listdir(directory):
    # Record the start time of the image processing
    image_time = time.time()

    # Construct the full file path
    f = os.path.join(directory, filename)

    # Check if the file is an image file (.jpg or .png)
    if os.path.isfile(f) and filename.find("jpg" or "png") > 0:

        # Run the object detection model on the image
        result = model.predict(f"{directory}/{filename}", save=True, save_conf=True,
save_txt=True, conf=0.2, iou=0.6,
                               project="results 1", name="results", exist_ok=True)

        # Get the class names from the results
        names = result[0].names

        # Initialize a list to store the number of objects detected per class
        class_detections_values = []

        # Count the number of objects detected for each class and add it to the list
        for k, v in names.items():
            class_detections_values.append(result[0].boxes.cls.tolist().count(k))

        # Create a dictionary mapping class names to the number of objects detected for that
class
        classes_detected = dict(zip(names.values(), class_detections_values))

        # Print the time it took to process the image
        print(f"Image processed in {time.time() - image_time:.2f}s")

# Print the total time it took to process all images
print(f"All images processed in {time.time() - start_time:.2f}s")
```

Figure 11: Model Prediction Script

Heatmap Generation

The density_heatmap.py script reads bounding box data from text files, which are associated with specific images. Each bounding box is represented by a class and coordinates. The class corresponds to a specific color, defined in the class_values dictionary. The script then creates a heatmap by drawing these colored bounding boxes on the corresponding image. The color of each box represents the class of the object within the box, with different colors indicating different classes. The script also creates a legend that maps these colors to their respective classes. This script represents the final output of the robotic solution which provides the end user with a detailed breakdown of the fuel ratings of vegetation in a given forested area.

```python
import os
import cv2
import numpy as np

# Define the class values and corresponding colors
class_values = {
    0: (0, 255, 255),  # aspen - VERY LOW (Yellow)
    1: (0, 165, 255),  # larch - LOW (Orange)
    2: (0, 0, 255),  # pine - HIGH (Red)
    3: (0, 0, 128),  # white_spruce - EXTREME (Maroon Red)
    4: (0, 0, 128),  # black_spruce - EXTREME (Maroon Red)
    5: (0, 0, 255),  # coniferous - HIGH (Red)
    6: (0, 165, 255),  # deciduous - LOW (Orange)
}

# Define the directory where the images are stored
image_folder = 'LOL'

# Function to read bounding boxes from a file
def read_bounding_boxes(file_path):
    if not os.path.exists(file_path):
        return []
    with open(file_path, 'r') as file:
        lines = file.readlines()
    boxes = [line.strip().split() for line in lines]
    return boxes
```

Figure 12: Heatmap Creation Script

```
# Function to create a heatmap from the bounding boxes
def create_heatmap(image, boxes, class_values):
    for box in boxes:
        class_name, x_center, y_center, width, height, _ = box
        color = class_values.get(int(class_name), (0, 0, 0))
        x = int(float(x_center) * image.shape[1])
        y = int(float(y_center) * image.shape[0])
        w = int(float(width) * image.shape[1])
        h = int(float(height) * image.shape[0])
        cv2.rectangle(image, (x-w//2, y-h//2), (x+w//2, y+h//2), color, -1)
    return image

# Function to create a legend for the heatmap
def create_legend(class_values):
    labels = ["VERY LOW", "LOW", "HIGH", "EXTREME"]
    colors = [class_values[i] for i in range(len(class_values))]  # Get colors from
class_values

    legend = np.zeros((100, 500, 3), dtype='uint8')
    for i, (label, color) in enumerate(zip(labels, colors)):
        cv2.rectangle(legend, (i*70, 0), ((i+1)*70-10, 50), color, -1)
        cv2.putText(legend, label, (i*70+5, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255,
255), 2)
    return legend

# Create the legend
legend = create_legend(class_values)

# Iterate over the files in the image folder
for file_name in os.listdir(image_folder):
    if not file_name.endswith('.jpg' or '.jpeg' or '.png'):
        continue  # Skip non-image files
    image_path = os.path.join(image_folder, file_name)
    image = cv2.imread(image_path)

    label_folder = os.path.join(image_folder, 'labels')
    label_file_path = os.path.join(label_folder, file_name.replace('.jpg', '.txt'))

    boxes = read_bounding_boxes(label_file_path)
    if not boxes:
        print(f"No detections for {file_name}, skipping...")
        continue

    heatmap = create_heatmap(image, boxes, class_values)

    # Save the heatmap image
    output_folder = os.path.join(image_folder, 'heatmap_results')
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    cv2.imwrite(os.path.join(output_folder, file_name), heatmap)
    cv2.imwrite(os.path.join(output_folder, 'legend.jpg'), legend)
```

Figure 12 cont.: Heatmap Creation Script

Social Impact & Innovation

TensorForest is designed to significantly enhance fire prevention capabilities across diverse geographical and economic landscapes. It will primarily benefit municipalities, regional governments, and private property owners who lack the resources to regularly employ professional auditors to assess vegetation fuel and flammability risks. This tool is particularly vital for rural regions in countries such as Pakistan, Indonesia, and Vietnam. These areas, despite not historically being prone to forest fires, are now facing increasing incidents each year. The lack of infrastructure, training, and resources for fire mitigation in these countries means that fires can escalate rapidly, causing substantial damage.

The importance of TensorForest cannot be overstated. It democratizes access to critical fire prevention technology, allowing areas with limited resources to actively manage and mitigate fire risks. In many developing countries, the lack of infrastructure, training, and access to fire mitigation resources means that forest fires often escalate far more than necessary. This not only results in larger fires but also severely impacts nearby communities who suffer from the health effects of PM2.5 pollutants and bear the brunt of economic losses. By providing tools to create accurate and accessible vegetation fuel and flammability maps, TensorForest addresses these challenges at the root, potentially preventing devastating outcomes and promoting sustainable environmental management.

For example, if a fire department on the island of Sumatra in Indonesia is given access to TensorForest, they can generate a vegetation or flammability map for their given forest. This flammability map allows them to then identify the lines of continuity where trees need to be cut or pruned, as well as the high-risk/most flammable zones which need to be fire safety target areas during extreme hot weather conditions, etc. This would allow them to prevent the spread of the fire to a point where such devastating outcomes occur. If the fire was stopped earlier, there would be fewer pollutants consumed by the residents, less economic loss, fewer greenhouse gases emitted, and so on.

The latter was an extremely niche example; however, it is meant to represent the versatility and insightfulness of a tool like TensorForest. TensorForest, in its current stage, is only on its first iteration and can only generate a vegetation/flammability map. However, given a bit more resources and access to a better LiDAR/photogrammetry sensor, the drone can have an exponentially greater amount of applications.

Figure #13: Forest Fire in Rural Indonesia, Source: NYT

TensorForest as a Start-Up

As we move from discussing the social impact of TensorForest, it is crucial to envision how this solution could function as a startup. By incorporating advanced drone technology, TensorForest provides a revolutionary approach to assessing and mapping forest flammability, representing a significant breakthrough in wildfire risk management. The core value proposition is its capability to deliver high-resolution aerial imagery that identifies various vegetation types and their flammability levels. This technology not only advances traditional methods but also significantly lowers the risks associated with wildfires by offering automation and convenience, replacing labor-intensive manual surveys.

TensorForest primarily targets fire departments and municipalities in developing nations, aligning its product with the needs of under-resourced areas that face severe wildfire threats. Additional customer segments include property owners in fire-prone areas and international environmental organizations, thereby enhancing market diversity and business resilience. The product reaches its customers through a direct-to-consumer e-commerce website, supplemented by partnerships with environmental NGOs and a dedicated sales team focused on governmental engagement, which facilitates personalized interaction and tailored solutions.

Customer relationships are managed through an automated system that provides real-time updates and monitoring, complemented by a responsive customer support team. The revenue model includes direct sales of drones and an initial image detection model, along with a subscription service for ongoing access to newer and updated image detection algorithms, promoting long-term customer engagement.

Overall, TensorForest can leverage corporate social responsibility in its marketing efforts by providing services to countries lacking fire safety resources. This initiative can serve as a starting point for promoting the product in new markets by demonstrating its effectiveness to communities in need.

Sources:

Canadian Wildland Fire Information System. (2024, April 16). Canadian Wildland Fire Information System Background Maps. https://cwfis.cfs.nrcan.gc.ca/background/maps/fbpft

Southern Fire Exchange. (2024, March 9). FIRE LINES A Bimonthly Newsletter of the Southern Fire Exchange. https://southernfireexchange.org/wp-content/uploads/v6-5.pdf

Canadian Wildland Fire Information System. (2024, April 24). Canadian Wildland Fire Information System Background Maps. https://cwfis.cfs.nrcan.gc.ca/background/maps/elevation

Innovative Solutions Canada. (n.d.). High Resolution Forest Mapping. https://ised-isde.canada.ca/site/innovative-solutions-canada/en/high-resolution-forest-mapping

Natural Resources Canada. (n.d.). Forest Classification. https://natural-resources.canada.ca/our-natural-resources/forests/sustainable-forest-management/measuring-and-reporting/forest-classification/13179

Alberta Agriculture and Forestry. (2012, August). Tree Species Impact on Wildfire. https://www1.agric.gov.ab.ca/$department/deptdocs.nsf/all/formain15744/$FILE/tree-species-impact-wildfire-aug03-2012.pdf

Canadian Wildland Fire Information System. (n.d.). Canadian Wildland Fire Information System Background Maps. https://cwfis.cfs.nrcan.gc.ca/background/fueltypes/d1

FireSmart Canada. (2008, April). FireSmart Guidebook for the Wildland/Urban Interface. https://firesmartcanada.ca/wp-content/uploads/2022/01/Firesmart_Guidebook_Final_April2008.pdf

Mitsopoulos, I., Mallinis, G., Arianoutsou, M., Dimitrakopoulos, A. P., & Ghosn, D. (2022). Wildfire risk assessment using machine learning: A case study in Greece. Fire Ecology, 18(1), 1-18. https://fireecology.springeropen.com/counter/pdf/10.1186/s42408-022-00132-9.pdf

van Geffen, L., Miesner, J., Kruse, S., Buras, A., Rammig, A., Rammig, A., ... & Bohn, F. (2022). SiDroForest: a comprehensive forest inventory of Siberian boreal forest investigations including drone-based point clouds, individually labeled trees, synthetically generated tree crowns, and Sentinel-2 labeled image patches. Earth System Science Data, 14(11), 4967-5005. https://essd.copernicus.org/articles/14/4967/2022/

Enayetullah, H., Hopkinson, C., Chasmer, L., & Mahoney, C. (2022). Identifying Conifer Tree vs. Deciduous Shrub and Tree Regeneration Trajectories in a Space-for-Time Boreal Peatland Fire Chronosequence Using Multispectral Lidar. Atmosphere, 13(1), 112. https://www.mdpi.com/2073-4433/13/1/112

Crowley, M. (n.d.). Forest Fire Prediction and Control. University of Waterloo. https://uwaterloo.ca/scholar/mcrowley/projects/forest-fire-prediction-and-control

Jain, P., Coogan, S. C., Subramanian, S. G., Crowley, M., Taylor, S., & Flannigan, M. D. (2020). A review of machine learning applications in wildfire science and management. Environmental Reviews, 28(4), 478-505. https://www.mdpi.com/2571-6255/7/1/13

Cortés, A., Margalef, T., Luque, S., & Malik, A. M. (2023). Machine Learning for Wildfire Prediction: A Systematic Review. Forests, 14(2), 319. https://www.mdpi.com/2571-6255/6/8/319#:~:text=The%20ML%20model%20obtained%20a,that%20determine%20wildfire%20growth%20rate

Mitsopoulos, I., Mallinis, G., & Arianoutsou, M. (2017). Wildfire risk assessment in a typical Mediterranean wildland–urban interface of Greece. Environmental Management, 60(5), 890-900. https://www.sciencedirect.com/science/article/pii/S0378112717300610

U.S. Forest Service. (n.d.). Geodata Mapping. https://data.fs.usda.gov/geodata/edw/datasets.php?xmlKeyword=humidity