İZMİR
KÂTİP ÇELEBİ
ÜNİVERSİTESİ
2010
GRADUATE SCHOOL OF NATURAL
AND APPLIED SCIENCES

Machine Vision in Robotics

ROE510

Project

Onuralp Arslan

2025

Detection of Forest Fire Through Video Recording

To detect forest fires from videos many image processing methods can be used. In this assignment we will focus on different approaches to detect forest forest from a given image source.

For all the applications, complete source codes can be found on the shared github page, to enhance explanation, some snippets will be shared through this paper.

The Github link for all sources is https://github.com/onuralpArsln/droneImageFireDetection.

The three approaches that are tested in this study are training a model, using image processing methods and feature detection-matching. They are tested in their respective sections and the code for each approach is given as separate folders under the given github repo.

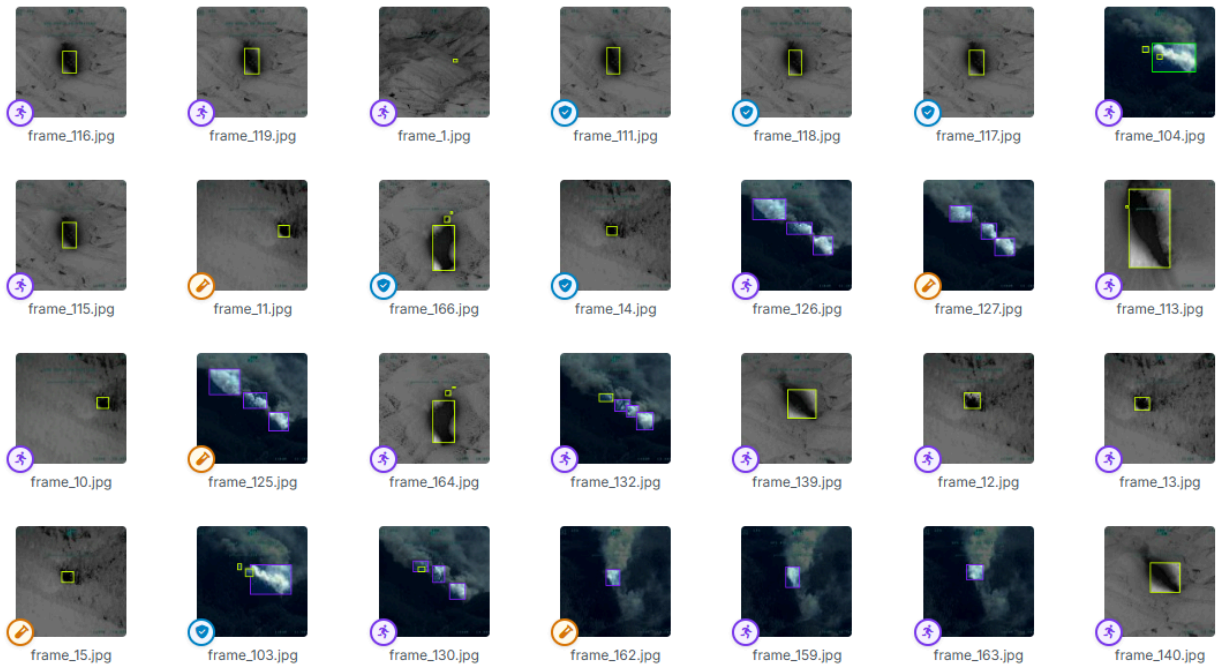
Approach One: Training an Object Detection Model

With developments in deep learning, there are many models available for users to adjust for their custom dataset. To obtain such a custom dataset we need an annotation tool and source images.

Images are quickly generated using given videos. A simple python script that employs opencv reads a frame of given videos, saves it as an image then fast forwards 1 second. So with that method around 200 frames are obtained from a single video.

Images are uploaded on roboflow, an easy to use platform to quickly annotate any images. This platform also gives valuable tools to preprocess and augment the dataset. To augment some new images created with gaussian blur, added noise, shear and rotation tripling the original data amount.

Shear and rotation is useful to mimic movements of a drone. Adding data with noise or gaussian blur can mimic any recording artifacts such as focusing and noise that is caused by fire.



Screen capture from roboflow UI, showing some of annotated frames.

Although now this data can easily be downloaded to device, since training will be performed on google colab to use better hardware than i own, to make process easier instead of downloading data and re-uploading to colab roboflow api is used to directly send data to colab.

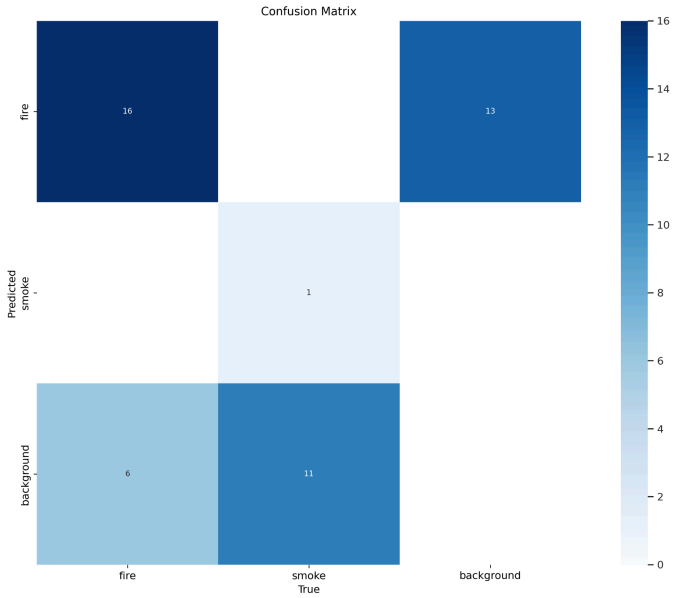
```
!pip install roboflow
from google.colab import userdata
from roboflow import Roboflow
ROBOFLOW_API_KEY = userdata.get('ROBOFLOW_API_KEY')
rf = Roboflow(api_key=ROBOFLOW_API_KEY)
project = rf.workspace("education-f32oy").project("forestfire-1x0i0")
version = project.version(3)
dataset = version.download("yolov11")
```

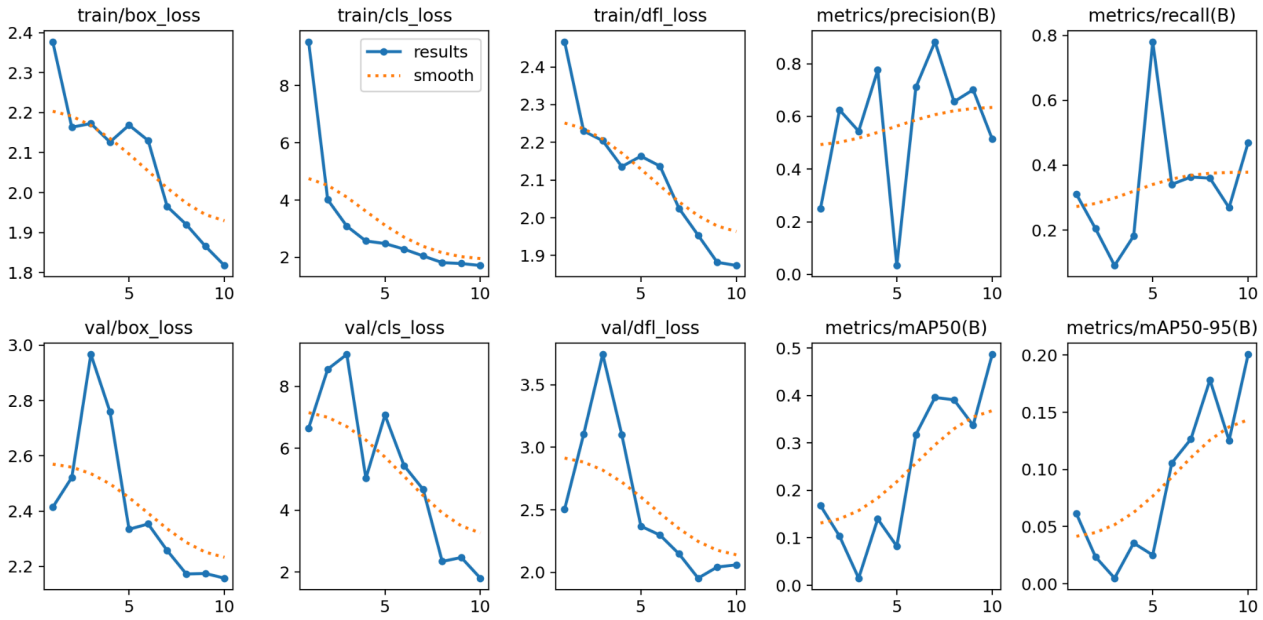
ROBOFLOW_API_KEY is a colab notebook variable where i add my api key. As the next step yolov11 must be clone to colab notebook.

```
%pip install "ultralytics<=8.3.40" supervision roboflow
import ultralytics
ultralytics.checks()
```

Yolo v11 is a general purpose object detection model with pre trained weight. It can be easily train on user data for special purpose applications. It is downloaded over ultralytics.

First training made with a dataset that shows both smoke and fire with batch size of 32 and 30 epoch. Resulting confusion matrix showed many false guesses by model.



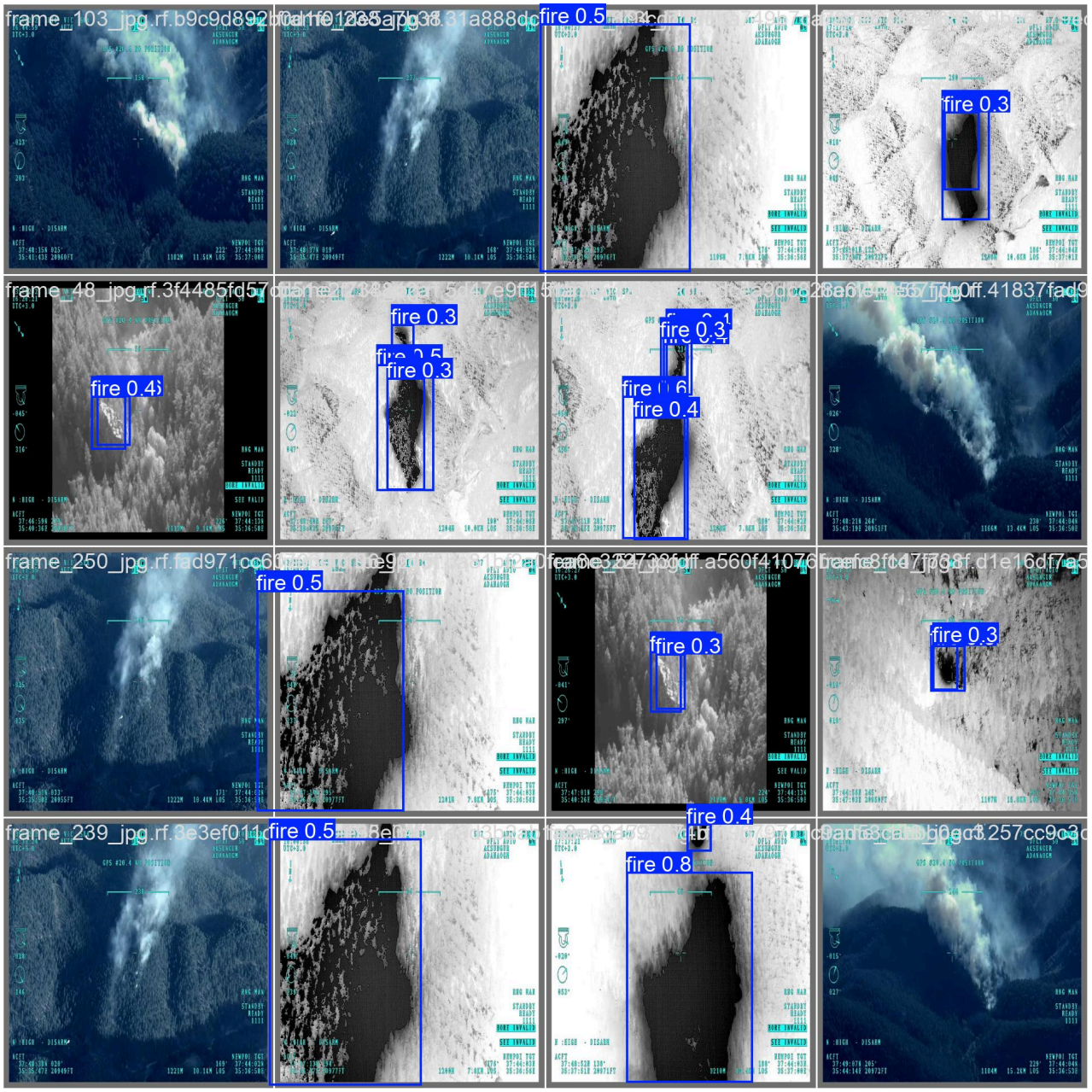


Loss graphs a showed results that are unstable for initial 10 epochs and guesses made were very poor on grayscale images and completely missed by model.

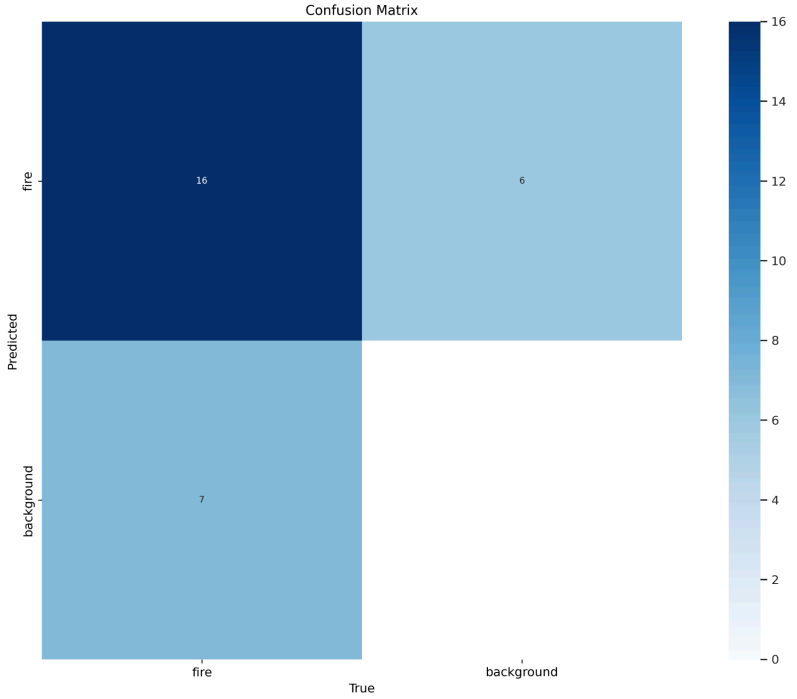
To overcome this problem the dataset is revised to show only fire in all images and if fire is not directly seen by image location is guessed according to smoke shape. This is expected to also help with guessing the original fire location by fire. Since on colored images fire is rarely seen directly it is highly possible that using a dataset containing an unbalanced amount of smoke and fire tags caused a bias in the model. This guess is can be also supported by looking at results that never made a guess about smoke.

When working for classification, using an unbalanced amount of classes in a dataset can cause a bias and the model tends to ignore classes with less data. Random augmentation is highly likely to further strengthen this imbalance.

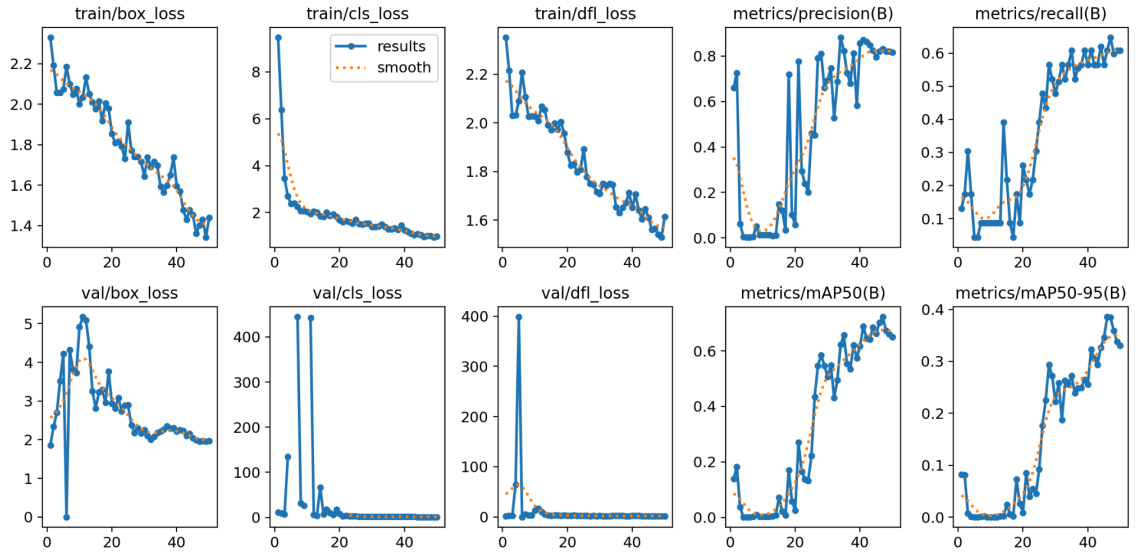
To resolve the given issue, data annotation is revised.


fire 0.5
fire 0.3
fire 0.3
fire 0.3
fire 0.4
fire 0.3
fire 0.4
fire 0.5
fire 0.3
fire 0.3
fire 0.5
fire 0.4
fire 0.8

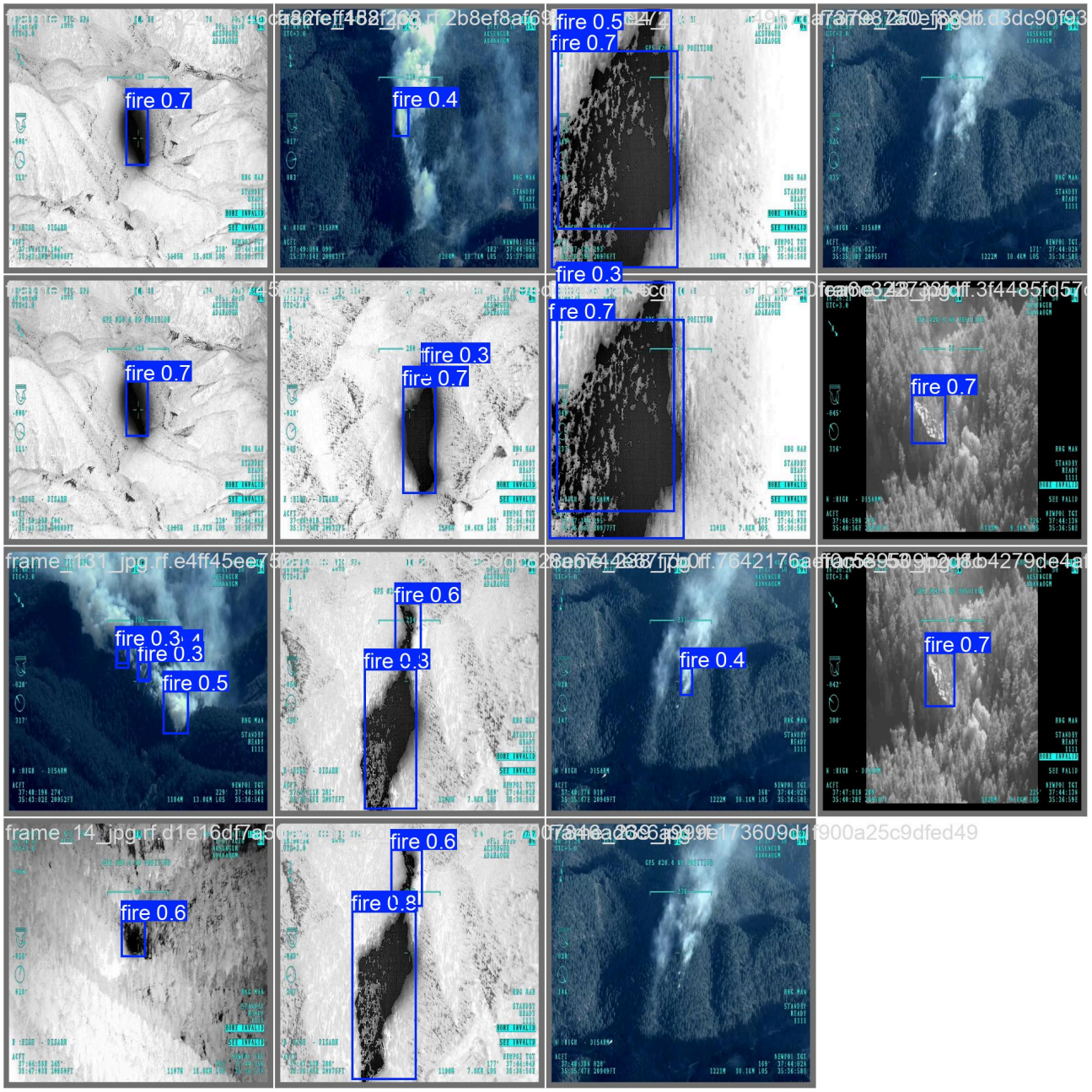
With revised annotations new training performed with batch size of 32 over 50 epochs.



Resulting confusion matrix showed improvement on fire detection with a reduced amount of false positives. Loss graphs are again unstable at the beginning yet they quickly started to follow a much more smoother profile with revised annotation.

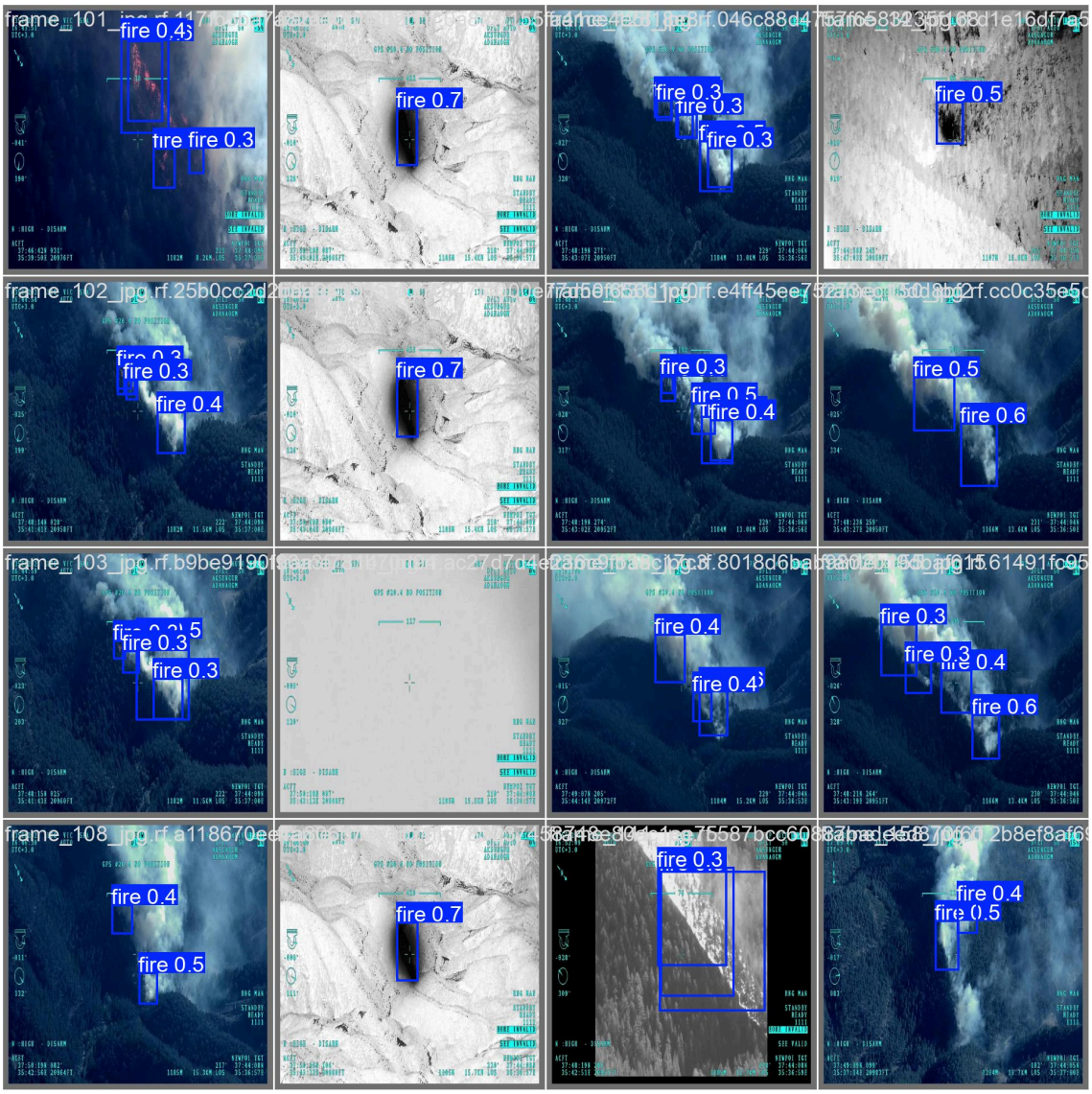


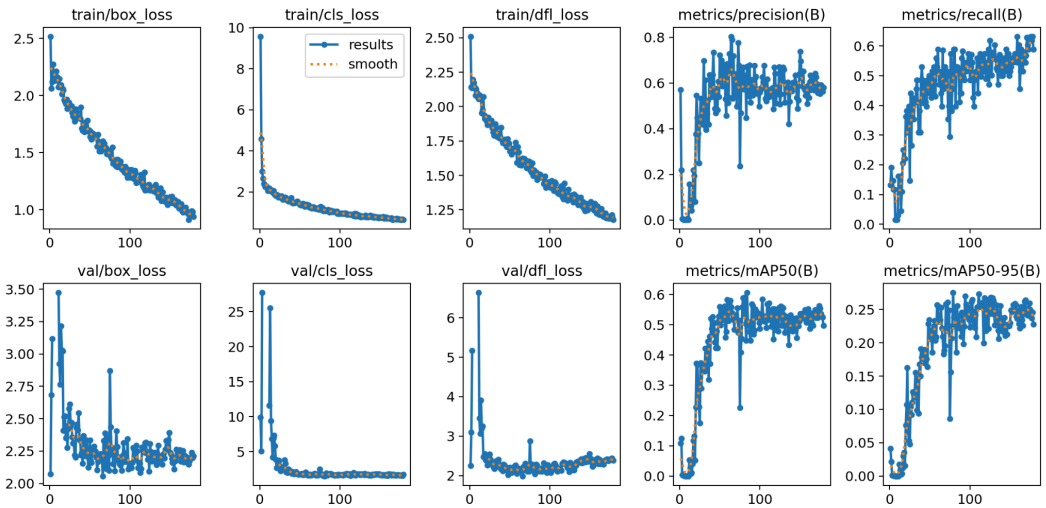
With revised annotation results are much better on RGB images yet there is massive room to improvements.



To further improve more frames from videos are added, mostly RGB ones, since models can't efficiently perform on those. With added new data performance increases on detection yet los function still show an unstable profile.

With newly added data a training with batch size of 64 over 200 epochs is started yet due to no improvements over last epochs, it stopped early on epoch 179. With a much more successful result. It shows that it needs more data to properly detect fire.

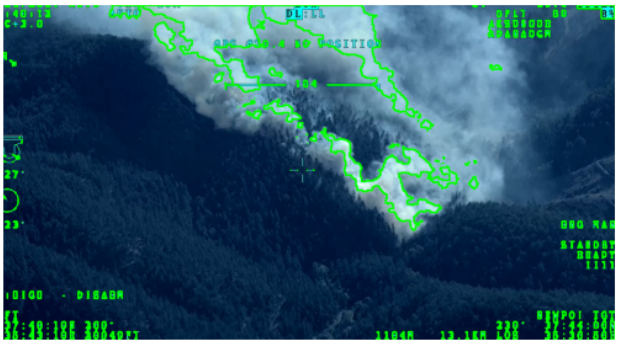

fire 0.4
fire 0.3
fire 0.7
fire 0.5
fire 0.6

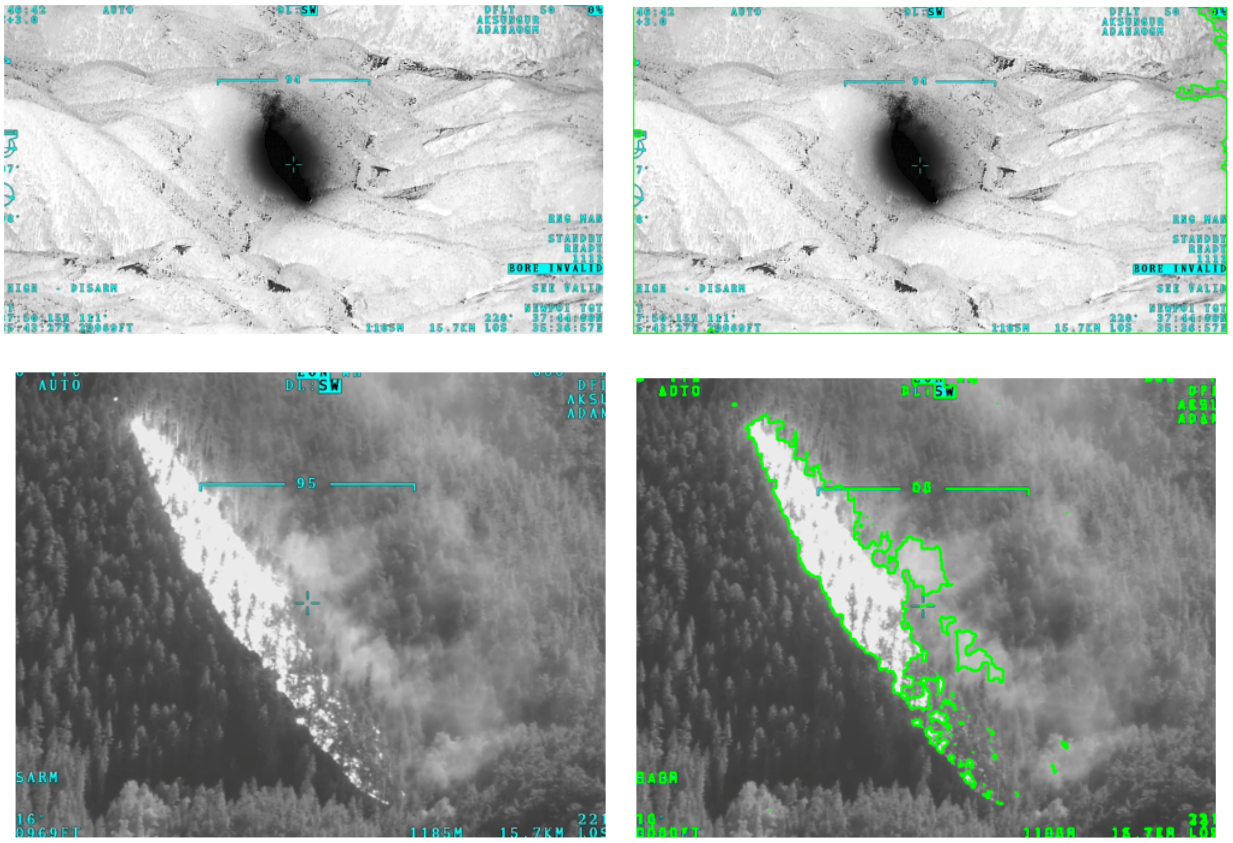

train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)

Approach Two: Using Image Processing Methods

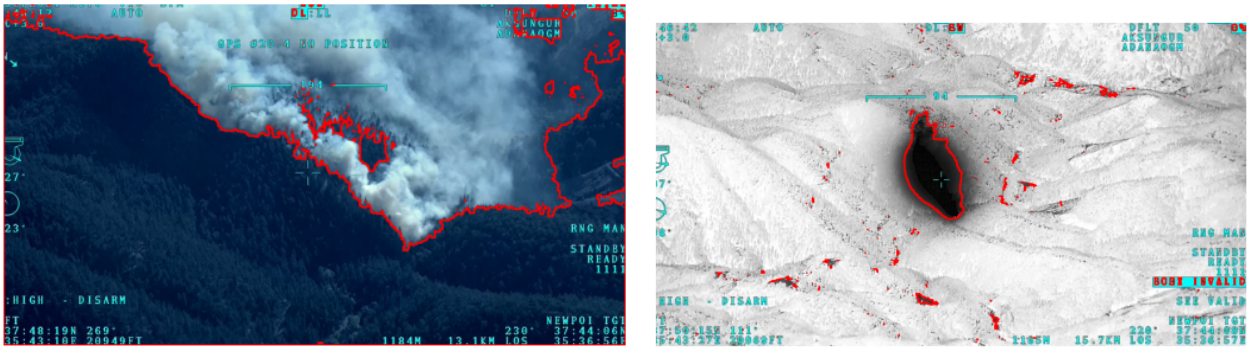
When videos are evaluated visually it is seen that places with fire and/or smokes are seen like clumps. This can be an easy target to detect using thresholding applications.

Initial approach is to use thresholding to make mostly white or mostly black clumps more prominent. Then a mask generated with thresholded images both for white and black mask and they are applied with the ”findContours” method from opencv.

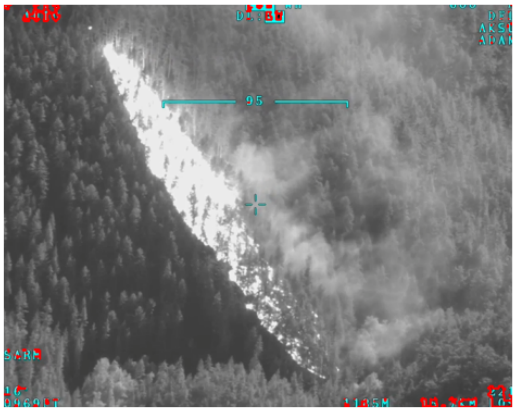


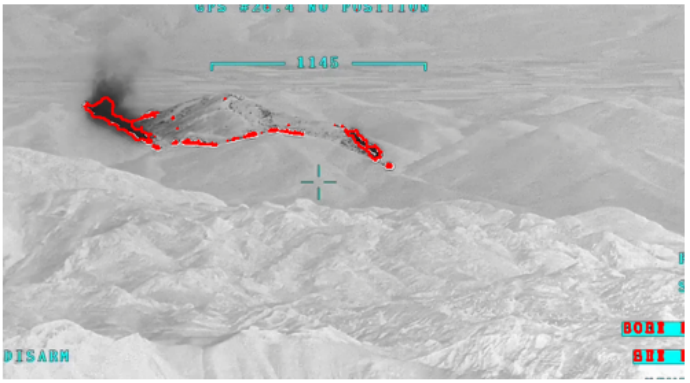


Results from masking white clumps are not successful in RGB images and completely unsuccessful when flames seen as black. When flames are seen as white it is much more successful.



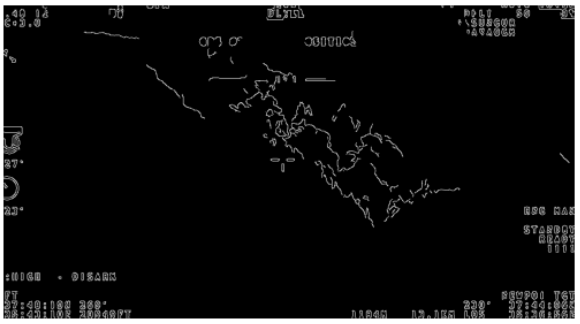
With thresholding the black pixels it yields better results at detecting fire when it is seen as black but RGB images with smoke detect areas that are not smoke. Yet it was a better smoke detection compared to white mask. In examples with fire seen as black it had many false detections as well as detecting the actual target.

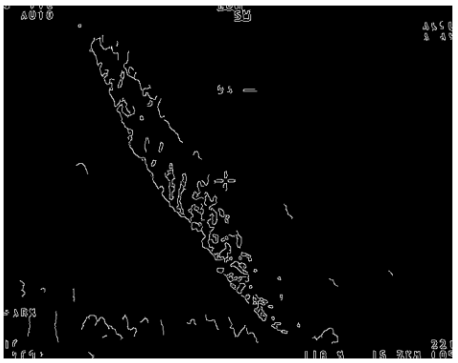


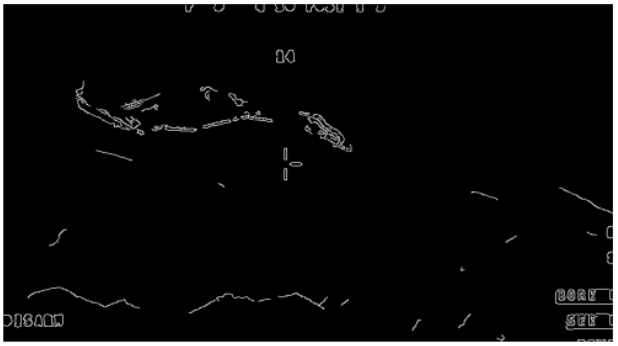


Since this approach requires fine tuning for each occurrence and risk of false positives even finely tuned this is not an applicable approach.

Another approach is to use edge detection expecting to detect areas of smoke and fire. Using canny edge detection with different values showed that high frequency images that are taken from forests have many features that are detected by Canny. To successfully detect smoke and fire alone it required tuning that lessens the performance of edge detection.

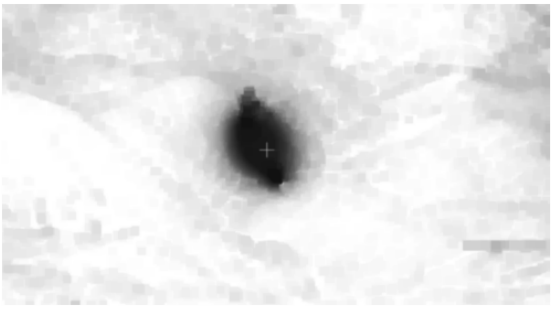




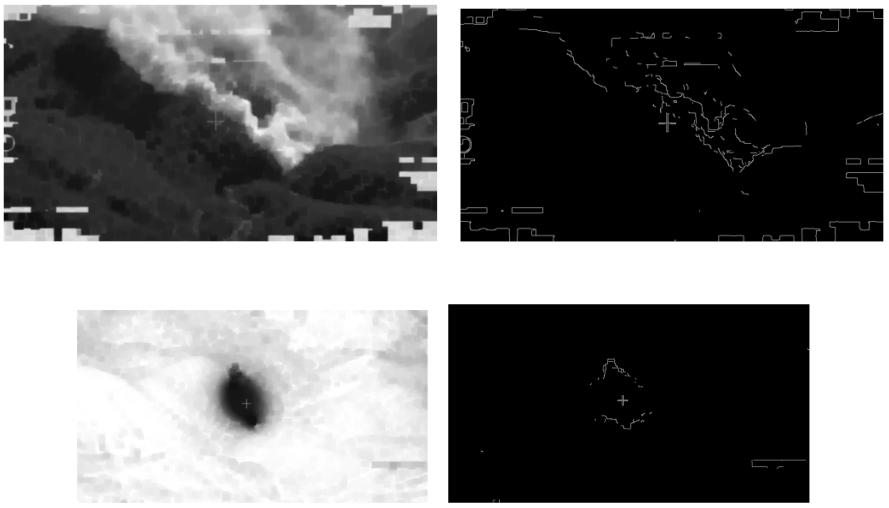


Although given examples are sherry picked to be on the successful side, they are not reliable to detect only fire areas. It also detected features like mountains and trees. Using different thresholding values that wont detect such features also started not to detect smoke and fire.

So to solve problems encountered in previous trials a more complex method is offered. Performing morphological closing operations to join areas together so areas of forest, fire or any other feature becomes differently colored areas. It uses a kernel 13 by 13 and is performed in a loop of 50 iterations. Adding a gaussian blur with kernel of 15 by 15 after process to smooth any noise or artifacts.

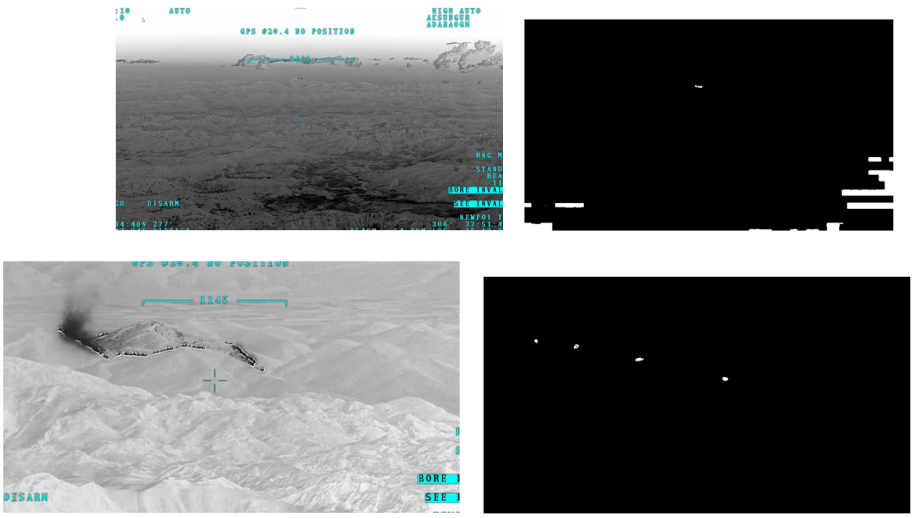


Edge detection is again applied to images that undertook described operations.



Morphologically processed image Edge detected image

Since the success of this method is not sufficiently successful enough, a flood filling is applied. Which is a method to find between detected edges and areas even if they don't possess closed loops flood filling can perform better compared to methods like contour filling.

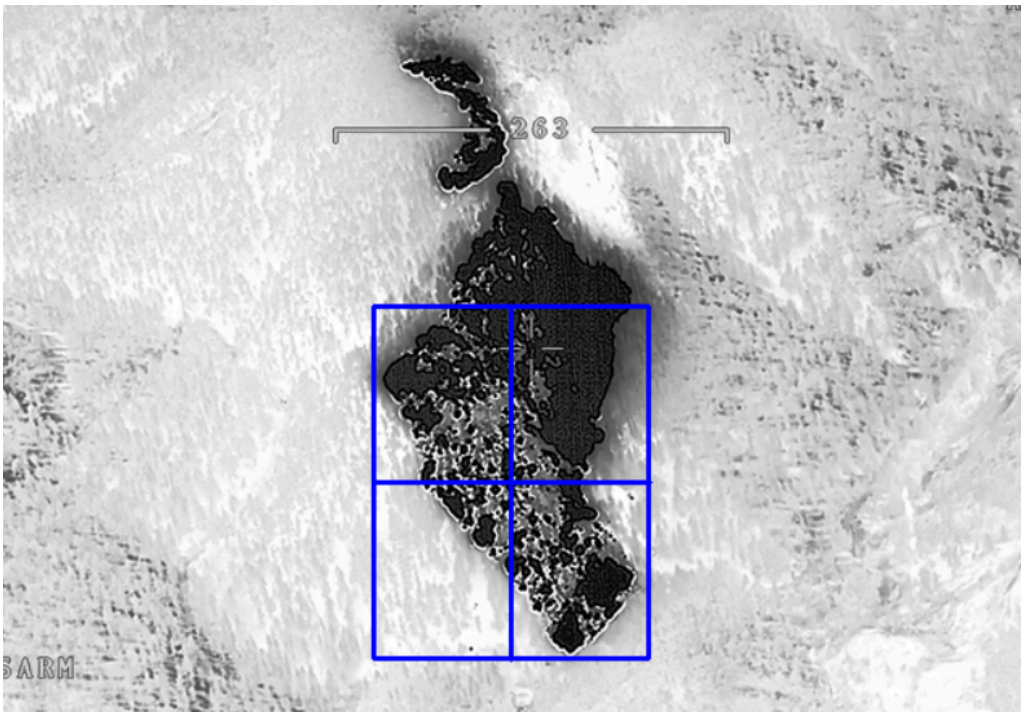


Yet this addition was not successful in detecting fire. It managed to detect some points but it was not reliable.

Approach Three: Using Image Processing Methods

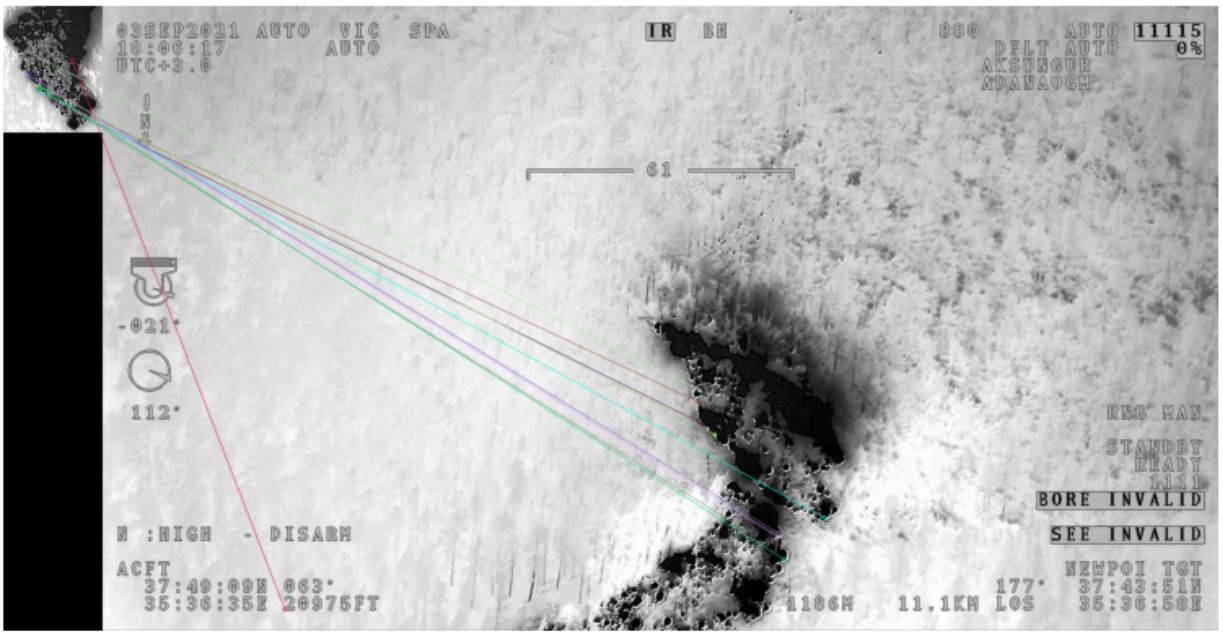
Feature extraction techniques were tested using OpenCV's ORB feature extractor and BFMatcher. This method demonstrated the potential to detect certain flame features in images. However, it struggled with inconsistent matches due to the varying colors and patterns of flames. Although this approach could be meaningful with a large dataset of diverse fire features, it is not reliable as a standalone method.

To easily select features we need to select the ROI that includes our features. Using selectROI tool within open cv, a GUI of ROI selection can be used.

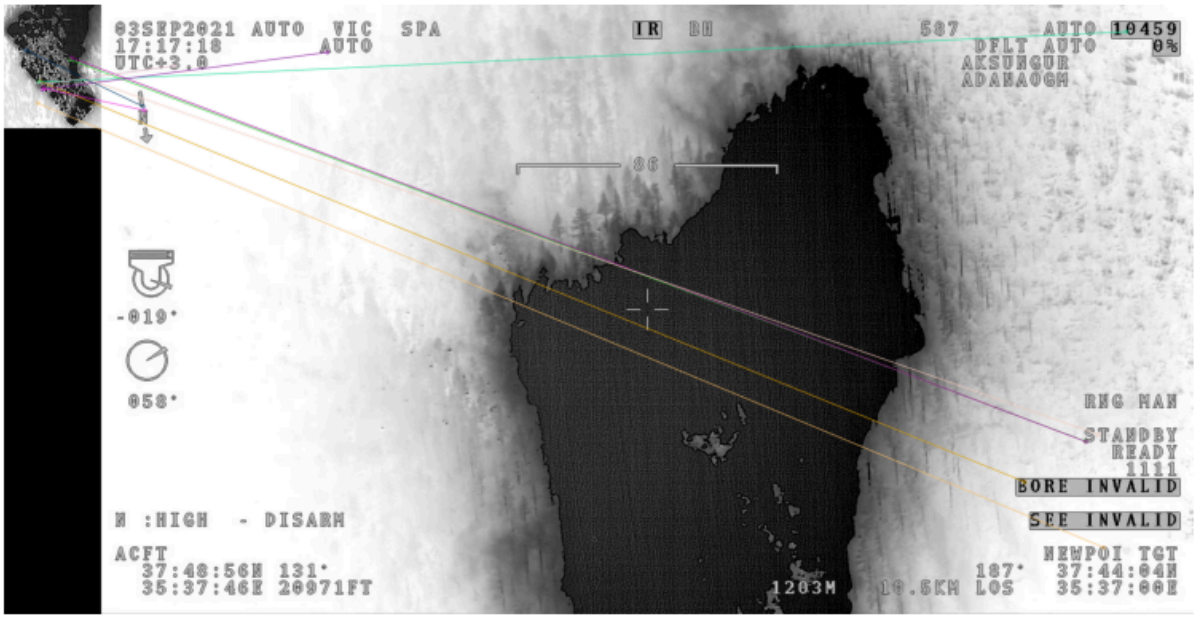


After selecting a ROI, ORB_create can be used to extract features. Extracted features can be compared with other images finding the same features might indicate fire. The first image that tried to match shows hopeful matching but on others it can mistook GUI elements as features. Results with successful matches are shown to be promising yet it missed on some trials completely not being able to detect images. To succeed repeatedly images need to be similar with source yielding the result of needing many predicted features as a database and continuously looking for matches over each image.

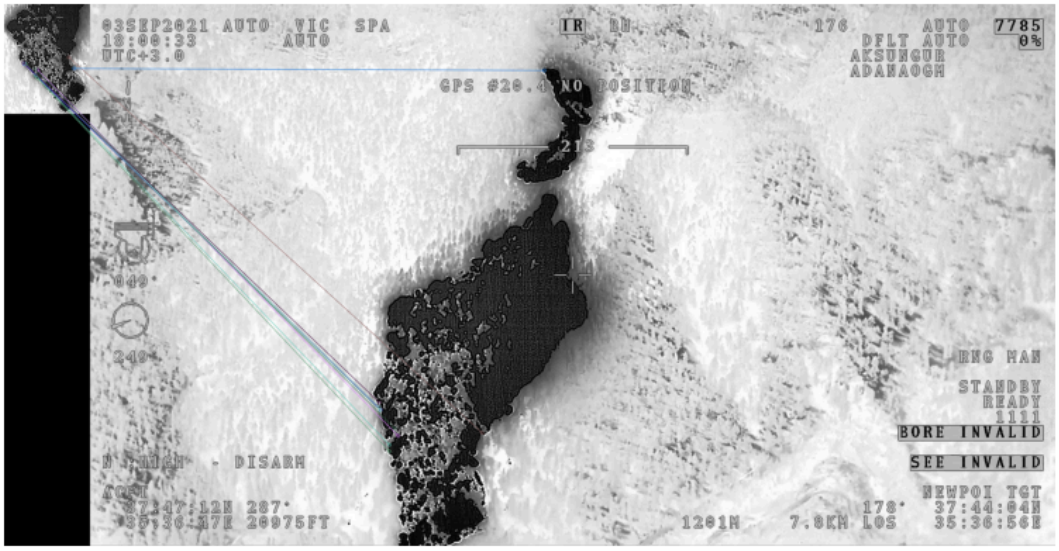
Matches in Image 0



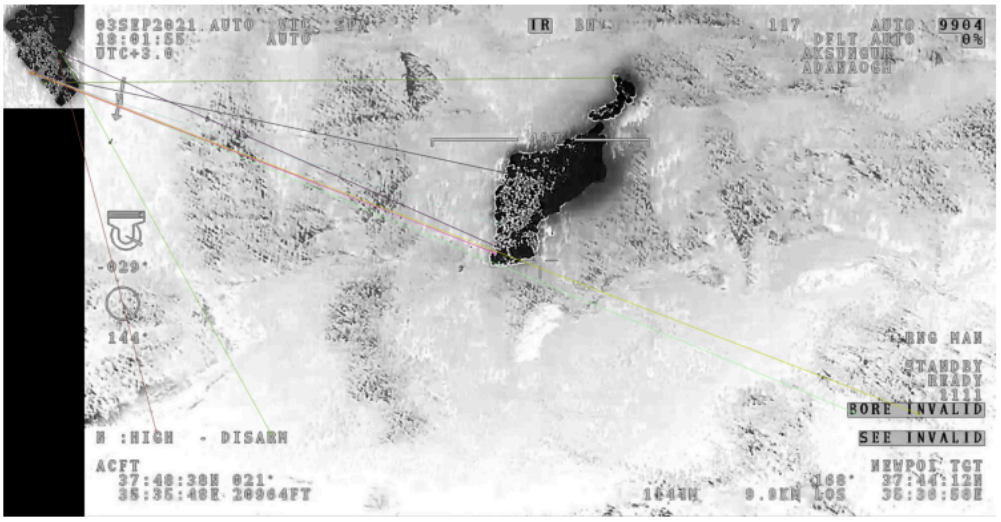
Matches in Image 1



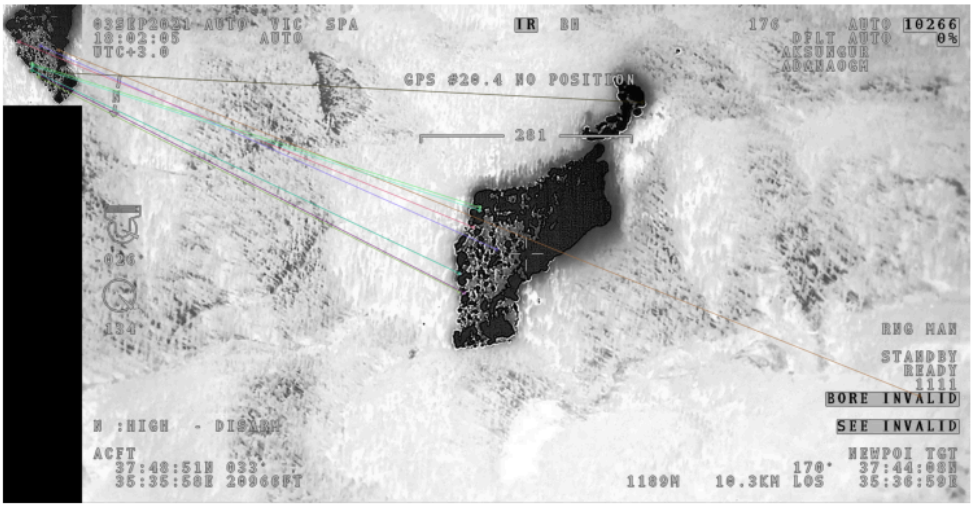
Matches in Image 3



Matches in Image 4



Matches in Image 5



Conclusion

As the result of comparing three methods, training a model was the easiest and most successful method. During the training process it is shown that adding more data and adding it correctly drastically increased performance of the model. Unlike using computer vision methods it performed without human interference during usage. It is more costly to train feature detectors but it can also outperform limitations of feature detection where it often struggles with GUI elements while trained model can easily overcome such obstacles.