Fire Classification

By India Ayo Godfrey

Abstract:

This project was an attempted to build a fire classifier specifically to help with Challenge 3 for the MBZIRC. This document serves as a tool to help evaluate my progress for my research done this semester.

Objectives:

- To learn the process of classification by way of Computer Vision
- To write and train a classifier for urban fires
- To attempt to use color segmentation and edging to train a KNN model

Accomplishments:

- Learned a process of classification by way of Computer Vision
- Wrote and trained a classifier for urban fires
- Learned more about various methods of image processing
- Learned how to install software using the terminal (Ubuntu 18.04 operating system) and a few debugging tricks
- Learned how to process images for classification with color segmentation and edging
- Learned how to mask a video with a specific RGB value

Classification Process:

The first objective was to learn the process of how to classify an object. The two ways of classification that were attempted were 1) training an image classifier and 2) using a combination of image processing and the KNN algorithm.

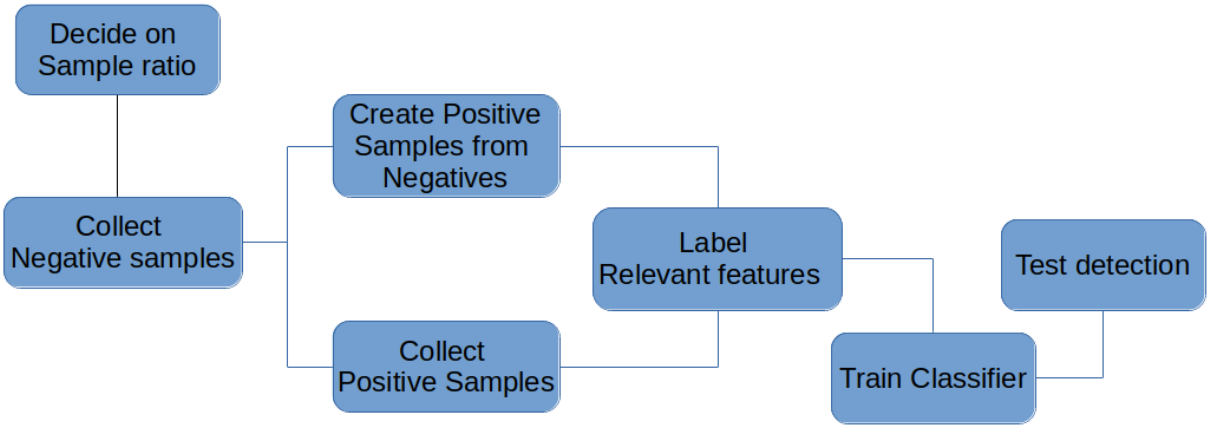

Figure 1: The Process of creating and testing a classifier

For the training set to be aligned with the environment of the competition, the images of fires for this project specifically needed to have images consisting of mostly small fires in urban settings. Because there was not already a database to fit this specific need, a database of 3500 positive sample and 2380 negative samples were created by collecting 2389 negative images. The negative images came from the Imagenet Databases (more specifically the sports and athletics, artifacts, people, and construction collection). The negative images were converted to grayscale images and resized to be 100 pixels X 100 pixels. The 3500 positive images were created by transposing seven pictures of fire that fit potential environmental components of the challenge

course (Figure 2 & Figure 3). For each of the seven images, there were 500 new positive images created in this manner.

Figure 2: Pictures used to create positive samples **Figure 3: An example of a positive sample**

After the positive images were created, OpenCV was used to train a classifier for 11 stages using 3000 positive samples and 2000 negative samples. A program was then written to train the classifier using OpenCV and numpy.

Results:

According to a test done during this project, the classifier has a detection rate of 80%, meaning that the classifier detected the fires in 80% of the images tested on. During the test, however, objects in the background that were not fire were in some cases labeled as fire. The classifier detects both pictures of fire from but nearby and distant locations (Figures 9 & 10). However, it also classifies glares (Figure 8), bright lights (Figure 5), or patterns that resemble the shape of fire (such as the pattern on the dress in Figure 7) as fire. It does not classify an object as fire

merely because the color of the object is in a similar color range as fire (Figure 4 & 6), which is an improvement from a mere color segmentation model.

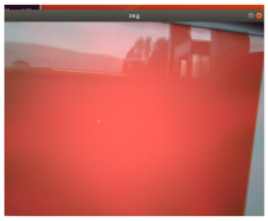

Figure 4: Testing within color range

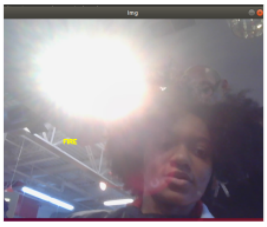

Figure 5: Testing with a bright light

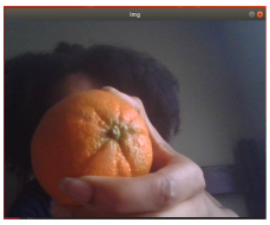

Figure 6: Testing with an actual object within color range

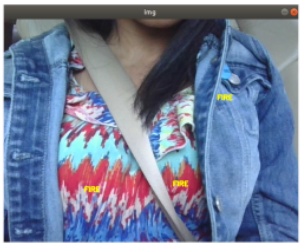

Figure 7: Testing with objects of fire-like shape

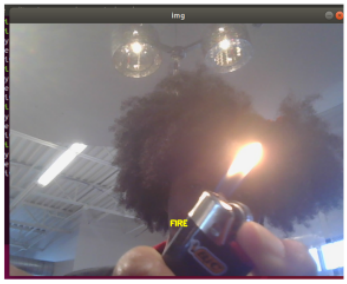

Figure 8: Testing with actual fire

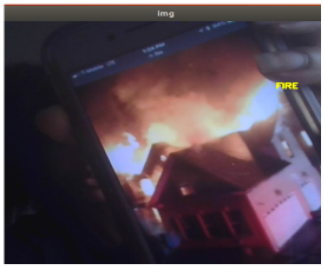

Figure 9: Testing with a photo of fire far away

Figure 10: photo of fire up close

Conclusion:

The images used to train the classifier were 100 pixel by 100 pixel in size. From the detector detecting glares and small patterns that resemble fire, it appears the algorithm is based off of images that are too small so instead of the detector looking at the video.Thus, a next step would be to retrain the classifier with images resized to a larger size to better match the video frame a suggested size is 300X300 pixels.

Image Work References

All of the negative images are from ImageNet, an online image library that can be found at http://image-net.org/explore under the “Person, individual, someone, somebody, mortal, soul”, “Sports and Athletics”, “Geographical Formations”, and “Artifacts” tabs.

All of the positive images were created within this project.

The urls for the images used for testing are listed below:

https://foxillinois.com/news/local/gallery/firefighters-battle-house-fire-on-west-decatur-street

https://www.google.com/amp/s/adventureasconstant.com/2017/01/31/small-fires/amp/

https://berkettinsurance.com/fires-home-insurance-los-angeles-ca/

https://www.google.com/amp/s/onthewight.com/ferry-fire-how-wightlink-define-a-small-fire/amp/

http://discovermagazine.com/2013/jan-feb/49-humans-had-mastered-fire-by-1000000-bc

http://www.artistacollection.com/index.php?dispatch=products.view&product_id=246

On top of building and testing the classifier, another model of fire detection was also attempted; however, the results of this model are currently unable to be tested due to the inefficient trainer (many pictures going through a series of for loops). Now, it is clear that

in order for this model to work, the trainer will have to use matrices instead of for loops but that was not realized until the end of the semester. However, the following segments of this report outline the work done towards this model of fire detection.

Two features often used to classify an object are the object's shape and it's color. In this classification model the general idea is that the training set is trained on a set of features derived from a method of image processing and then fed into a KNN algorithm.

Color Segmentation: can be used as a method of image processing and is the idea of finding a range of colors that can be considered as a feature of an object. These colors can then be masked using true or false values to indicate where in a photo those colors can be found. Color segmentation can provide specific HSV or RGB of an object and help an algorithm recognize the object from those values. An example of HSV thresholding can be seen in Figure 11.

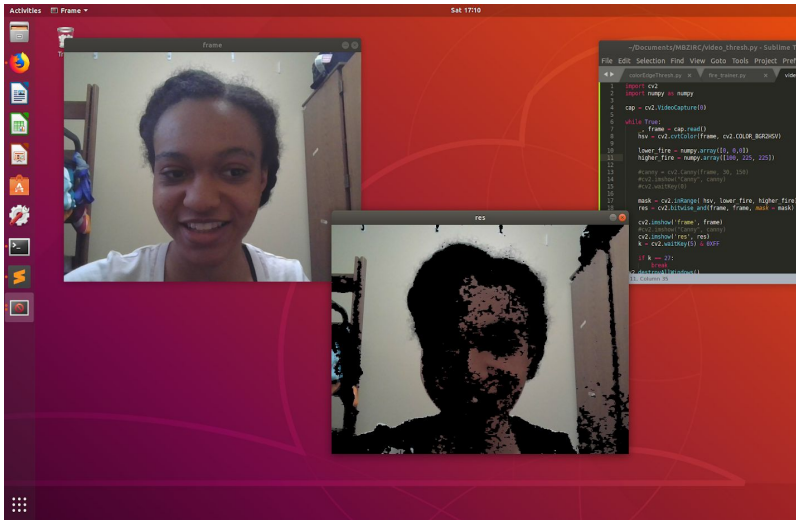

Figure 11: color thresholding used to display every color except dark brown, black and white

Edging: Using the same masking idea, edging allows one to look at the edges of an object and detect a shape.

Image processing can be used to train a data set for machine learning by helping to indicated distinct patterns. Based off of *Practical Python and OpenCV*, the adaptive color thresholding was the superior thresholding method. This was decided due to the fact that setting a threshold for fire manually could be extremely inaccurate due to the many variations of color patterns that can be found in fire and even the lighting used in the picture of the fire. Due to this fact Otsu a method of Adaptive Thresholding was used.

Canny edge detection was used for its simplicity in comparison to the sobel edge detection methods. Part of the Canny edging process also includes the Sobel filtering so Canny uses other types of edge filtering and outputs the most simplistic edging filter. As shown in Figure 12, the Canny edge detection creates the least amount of lines and gives a simplified version of the edges in the image in comparison to the and Sobel method.

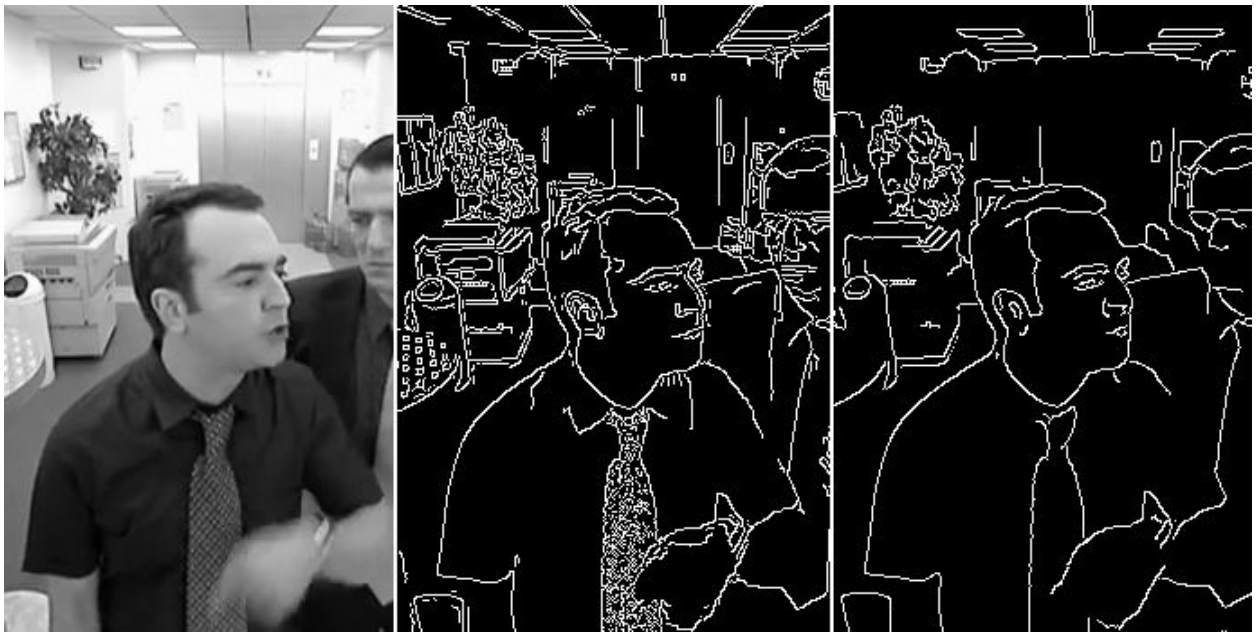

Figure 12: from left to right: the original image, the Sobel filter, the Canny filter (this picture was taken from https://blog.sicara.com/opencv-edge-detection-tutorial-7c3303f10788)

Machine Learning Models Considered:

The KNN algorithm is based off of ranges of data points. One can imaging that there three data points on a graph, if a new data point is presented, KNN says that whichever data point the new data point is nearest to based off of similarities is of the same classification (Figure 2). The Decision Tree is a series of decisions, it poses multiple question and based off of the boolean values of the answers, the classification of the object is determined (Figure 3). The advantages of KNN are that it handles noisy training data well, and it works well for large data sets. However, the drawbacks of using KNN is potentially slower computation time and that a developer may have to manually tweak the K- values, which are the number of data points that the algorithm takes into account when deciding what classification the new data point fits into. Advantages of the Decision tree are that its relatively intuitive for simple classification (detection of one or two features). Due the there only being one or two decisions made. However, the major disadvantage of it is that it is difficult to tweak for more robust classifications. There are many feature to fire such as color shape, and motion patterns, and it would be hard to write a simple decision tree to identify the intricacies of all of those features. Tweaking a more complicated model could be difficult as well considering that changing one node also overrides the nodes below it. Because if these facts and the fact that there were a lot more resources available to learn how to implement the KNN, the KNN model was chosen. However, with more time to experiment, the binary tree algorithm could also be implemented to compare the two.

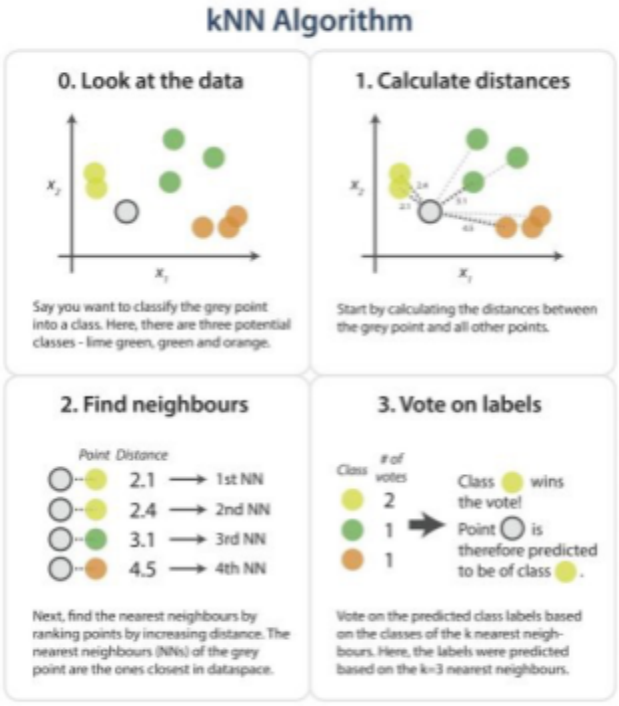

Figure 2. Visual representation of KNN algorithm

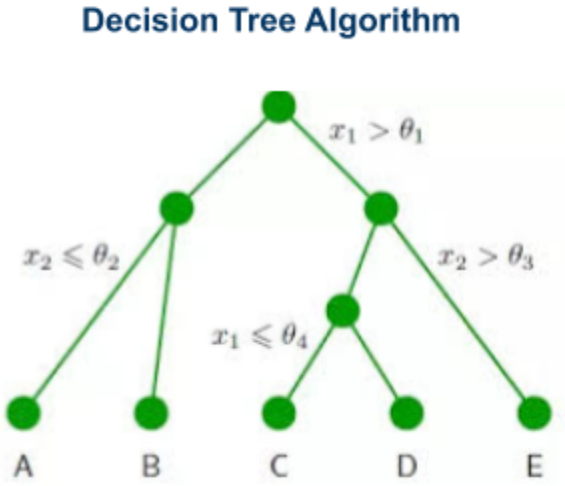

Figure 3. Visual Representation of DT

This section is attached to the report to address my process and next steps

<u>Challenges/ Solutions:</u>

- There was difficulty configuring the environment for the majority of the semester (this issue set the project back about two weeks). Every time I logged into the dual booted Ubuntu server, something disabled the driver so that it could not connect to the internet. Even when the drivers were reset, the connection was not restored for long. The issue was resolved when the Ubuntu version was updated to version 18.04.
- There were two different approaches to the project presented by two different mentors, The first approach was based on the idea of KNN with the expectation that it would

create a more robust system than image processing. The second approach was to just do a very specific series of image processing and see if an image could be detected from that before going into machine learning. Bits and pieces from each approach were taken to attempt to make the most robust classifier possible.

- There was no data sets already made with fires specific to our needs. Most of the photos were forest fires but the most accurate data set for our purposes would be pictures of small, urban fires. There were not that many pictures of small fires, thus acceptable positive pictures seven pictures were selected and the opencv_createsample command was used to transpose those photos onto the negative photos to expand the repertoire of positive photos.

Current Deliverables:

- A program that uses Canny Edging and Otsu Adaptive Thresholding to process an image, and KNN to train a data set (OpenCVWorkspace > fire_trainer.py)
- A program that can load a classifier and test it on a video feed (OpenCVWorkspace > fireClassifier.py)
- A positive (3500 images, these images can be found under OpenCVWorkspace > info) and a negative (2389 images, these images can be found under OpenCVWorkspace > neg).
- A classifier that has been trained for 11 stages (the classifier made can be found in the file path OpenCVWorkspace > fire-cascade-11xml.
- A program that can detect a specific color threshold in a video and mask out the rest of the video. The limitations of this program are vast, however, if the detection was based off of it's thresholding, anything of similar color would be classified as fire because it can

only look at a range of HSV. It does not utilize any edge detection. This program can be found in the path Documents > MBZIRC > video_thresh.py file, to use it, on just needs to enter an HSV range to mask the video input.

Next Steps:

- Experiment with the ratio of positive and negative images to see which ratio yields the best results for training a classifier
- Try training the classifier with larger images
- Look into classification by detection patterns in motion
- Look into adding a thermal thresholding component to which ever algorithm used
- Look into CUDA software as a way to spend up the trainer process

*All documents discussed in this report have been uploaded to the GTAR Team Dropbox.