A Comparison of Automated Bolide Detection Methods

Maxine Thembi Khumalo

Follow this and additional works at: https://repository.fit.edu/etd

Part of the Astrophysics and Astronomy Commons
We the undersigned committee hereby approve the attached thesis

“A Comparison of Automated Bolide Detection Methods”

by
Maxine Thembi Khumalo

______________________________
Csaba Palotai, Ph.D.
Associate Professor
Aerospace, Physics and Space Sciences
Committee Chair

______________________________
Xianqi Li, Ph.D
Assistant Professor
Mathematical Sciences

______________________________
Hamid Rassoul, Ph.D
Distinguished University Professor
Aerospace, Physics and Space Sciences

______________________________
David Fleming, Ph.D.
Associate Professor and Department Head
Aerospace, Physics and Space Sciences
Abstract

Title:
A Comparison of Automated Bolide Detection Methods

Author:
Maxine Thembi Khumalo

Major Advisor:
Csaba Palotai, Ph.D.

Bolide recording and analysis are crucial so meteoroid fragments can be found, a lightcurve analyzed, and its trajectory calculated. The Spalding Allsky Camera Network (SACN) generates videos and composite images of the night sky that are potential meteors based on changes in brightness. The best way to ensure quick identification is to automate the detection of bolides (and all meteors) using computational techniques. This project tested three algorithms to sort events between those with and without meteors - a Traditional Hough Detection Method, Convolutional Neural Network (CNN), and YOLOv5 against the previous technique from 2018 by Elena Botella. All the methods improved from the Optimized Botella method, but none performed well enough to automate detection fully. The best-performing method on all the metrics is the naive approach to the CNN, but it misses big, bright meteors and bolides. The three new methods can be combined to leverage their strengths to automate meteor detection fully.
# Table of Contents

Abstract ................................................................. iii

List of Figures .......................................................... vii

List of Tables .............................................................. xi

Abbreviations ............................................................... xii

Acknowledgments ........................................................... xiii

1 Introduction ............................................................... 1
   1.1 Aims ................................................................. 1
   1.2 Background ........................................................ 2
      1.2.1 Bolides ......................................................... 2
      1.2.2 History of bolide observation ............................... 3
      1.2.3 Computer vision techniques ................................. 5
      1.2.4 Machine learning techniques ............................... 8
   1.3 Limitations and Assumptions .................................... 9

2 Research Methodology .................................................. 11
   2.1 Data Preprocessing ................................................. 12
   2.2 Metrics ............................................................. 13
   2.3 Algorithm description ............................................ 15
2.3.1 Optimized Botella Automated Detection ........................................ 15
2.3.2 Traditional Hough Detection Method ............................................. 20
2.3.3 Convolutional Neural Network ..................................................... 29
2.3.4 YOLOv5 ................................................................. 31

3 Datasets and Data Collection ......................................................... 35
3.1 Hardware and Software Specifications ........................................... 35
3.2 Data Sources ............................................................................. 36
  3.2.1 Spalding Allsky Camera Network ............................................ 36
  3.2.2 SkySentinel Instrument ........................................................ 37
  3.2.3 PiSentinel Instrument ............................................................ 38
  3.2.4 How Automated Detection Fits In ............................................ 40
3.3 Data Classification and Segmentation .............................................. 40

4 Results ......................................................................................... 43
4.1 Optimized Botella Automated Detection ........................................ 43
4.2 Traditional Hough Detection Method ............................................. 46
  4.2.1 Training .............................................................................. 47
  4.2.2 Testing ............................................................................... 49
4.3 Convolutional Neural Network ...................................................... 52
4.4 YOLOv5 ................................................................................ 55
  4.4.1 Training .............................................................................. 55
  4.4.2 Testing ............................................................................... 59

5 Conclusions and Future Work ......................................................... 65
5.1 Compare Performance ................................................................. 65
5.2 Future Work ............................................................................. 68
List of Figures

1.1 A labeled composite image of a meteor on August 8, 2020, from Florida Tech roof. .................................................. 3

1.2 This shows how the Hough transform identifies lines. First, the algorithm detects the edges of the meteor path in b, then the two longest lines are identified in green in c and d. The meteor shown occurred on August 5, 2017. .................................................. 6

2.1 This is the flowchart for Optimized Botella Automated Detection. From left to right, it displays the different steps in the code that each video goes through. The method discretizes the video into consecutive frames that are each masked and runs pixel and coordinate analyses. ..................... 17

2.2 This is the flowchart of the pixel analysis Optimized Botella Automated Detection. The analysis subtracts the binary versions of two consecutive frames from one another and then adds together all the pixels to test for the brightest moving light. .................................................. 17

2.3 This is the flowchart of the coordinate analysis Optimized Botella Automated Detection. The analysis subtracts the current frame from the first frame and calculates the position of the brightest pixel. The center of mass calculation compares the positions of previous bright spots. . . . 18

2.4 This is a binary edge detection image of a meteor path with Hough line calculations drawn to compare $\rho$ values for a set $\theta$. ......................... 22
2.5 This is the flowchart for Traditional Hough Detection. After finding
the background image, the composite image analysis looks for meteors
before the video analysis checks events that test positive. . . . . . . . . 23

2.6 This is the flowchart for Video Analysis of Traditional Hough Detection.
The analysis discretizes the video then the procedure loops over each
consecutive frame. The procedure formats the frames so the Otsu test
can decide which circle test the procedure performs before calculating
the frame’s speed. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

2.7 This is a binary image of a meteor frame with approximate circle calcu-
lations drawn to compare the bright red zone to the dark green border.
The image passes the approximate circle test if most of the pixels in the
red square are white and most of the pixels in the green region are black. 27

2.8 This is the architecture of the Convolutional Neural Network. The pooling
layers are smaller than the convolutional ones because their job is to
reduce the dimensions of the feature maps. The fully connected output
layer condenses the feature map into a classification prediction. Stutz . 31

2.9 The anatomy of an object detector like YOLOv5. The EfficientNet CNN
into the BiFPN layer nodes until the class prediction Tan et al. (2020) 32

2.10 These are some example results to YOLOv5 for composite images from
the SACN data. The color-coordinated labels show how images are
output by YOLOv5. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

3.1 The map of SkySentinel cameras with the legacy instruments in red and
the new nodes in black . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

3.2 The SkySentinel instrument schematics (SkySentinel, 2013) . . . . 38

3.3 This is a labeled photo of the inside of the PiSentinel instrument. . . 39
4.1 These are a selection of composites of true positives from the Optimal Botella Method. The bottom right image is a bolide, while the other three are meteors. .................................................. 45

4.2 These are all the images of the composites of false negatives from training in the Traditional Hough Detection Method. All of these have very small meteors - some of which are hard to see. ................................. 48

4.3 These are a selection of images of the composites of false negatives from testing the Traditional Hough Detection Method. It includes mostly short, with the largest meteor in the bottom right image. .................. 51

4.4 Some false negatives from the test set of the best CNN model. These composite images include very short meteors and the large meteor in the top left corner. ......................................................... 54

4.5 Labels used in the training and testing set. The YOLOv5 software generates these plots to show patterns within the dataset. They show the number of labels per class and summarize the size. ..................... 56

4.6 YOLOv5 training confusion matrix. This training performed very well with true positives near 100 % for all the classes. ......................... 58

4.7 The precision and recall values over the epochs for the training set. These plots show how the training improves the metrics with each epoch with a near 100 % value for both after 200 epochs. ....................... 59

4.8 The precision and recall values over the epochs for the testing set. These plots show that introducing a testing dataset adds more noise to the metrics with larger amplitudes between epochs. However, the precision is not increasing while the recall is. ........................................ 60
4.9 Precision vs. Recall over the epochs for each label. The goal is high precision and recall, so the bolide label is the least successful while the lights label is the most.

4.10 YOLOv5 testing confusion matrix. These plots show the lights and the moon have a high percentage of true positives.

4.11 This shows the difference between the labeled test images and their results. The lights label performs well, while there are bolides misidentified most often by YOLOv5.

5.1 Metrics for each method. All three new methods improved the recall and F-score.

5.2 The computational time shows with methods is most efficient. YOLOv5 is the fastest, while all new methods outperform Optimized Botella.
List of Tables

2.1 Labeled confusion matrix example .................................. 15

3.1 Hardware and software specifications ................................. 36

3.2 Number of events used .............................................. 41

4.1 The confusion matrix for the Optimized Botella Method. .......... 44

4.2 The metrics from the Optimized Botella Method. ................. 44

4.3 The confusion matrix for the training data for the Traditional Hough Detection Method. .................................................. 49

4.4 The confusion matrix for the testing data for the Traditional Hough Detection Method. .................................................. 49

4.5 The results from the Traditional Hough Detection Method .......... 50

4.6 The top four best methods .......................................... 53

4.7 The results from the training set on the best model. ............... 59

4.8 The results from the testing set on the best model. ............... 61

A.1 The Python libraries used ........................................... 83
List of Symbols, Nomenclature or Abbreviations

BiFPN  Bidirectional Feature Pyramid Network
CNN  Convolutional Neural Network
CPU  Central Processing Unit
EN  European Fireball Network
GLM  Geostationary Lightning Mapper
GPU  Graphics Processing Unit
PCA  Principal Component Analysis
ReLU  Rectified Linear Units
SACN  Spalding Allsky Camera Network
SVM  Support Vector Machine
YOLOv5  You Only Look Once version 5
I would like to thank the following for their help and support:

Prof Csaba Palotai, for advising, guiding and inspiring. Thank you for making this project possible.

The Fulbright Program for sponsoring my travel and studies and providing help with being prepared for studying in a new country.

The Spalding Allsky Camera Network and their team provided data and resources on the hardware that is the backbone of this research.

To my peer graduate students for advise, data and editing help.

To my friends and family for their support.
Chapter 1

Introduction

This chapter covers the aims of this project. The background details previous work as a literature review. The limitations and assumptions section gives the boundaries and shortcomings of the project’s process.

1.1 Aims

This project aims to compare methods for automatically detecting meteors and especially bolides from all-sky camera recordings of the night sky. The detection devices include software that records when the sky’s brightness changes, containing events that are not meteors. Each day, users manually sort the previous night’s recordings (in the form of videos and images), which is time-consuming. This project adds to the work begun in Botella (2018), where brightness tests find the flashes of light in the recording. The aim is to build three new methods that improve on Botella (2018) by improving the computational time and reliability using traditional data metrics.

The detection process (identifying if there is a meteor) is the beginning of the process. Once detected, its velocity, trajectory, lightcurve and mass can be calculated.
from the all-sky camera recordings. Then the user runs a fragmentation model, which estimates how the bolide broke up. This data can be analyzed to find the meteor’s source to understand better the asteroids and meteors that threaten the Earth in the future. This project aims to design the essential first part of the data analysis, providing a reliable detection algorithm.

1.2 Background

1.2.1 Bolides

To understand how celestial objects entering the earth’s atmosphere are described, there are some terms to define. When a solid object is traveling through space, it is a meteoroid. Once it enters the Earth’s atmosphere and is large enough to produce light, this phenomenon is called a meteor. Apparent magnitude is the brightness of a celestial object on Earth. A fireball has an apparent magnitude of -4 or brighter, and a bolide is -14 or brighter (Belton et al., 2004). For reference, -12 is the moon, and -27 is the Sun (Agrawal, 2016). The objects creating the fireball are larger than less bright meteors (0.1 - 10 kg (Svoren et al., 2007)), and thus it is more likely that the meteorite fragments will reach the earth’s surface (Spurný et al., 2006). The meteor collides with atmospheric gases at a high velocity and undergoes the process of ablation, which heats the object and produces light (Trigo-Rodríquez et al., 2006). Larger, faster bodies create more heat and light to become known as fireballs (bolides). The largest ones can light up the entire sky for seconds.

Figure 1.1 shows a composite image combining all the video’s frames into a single picture, like a long exposure capturing light over time. This figure displays a critical phenomenon: the trajectory’s tapered shape. As a meteor enters the atmosphere, it heats up quickly, eventually becoming hot enough until it starts to shine brightly. It
shines brighter and brighter until it loses enough mass to slow down, gets dimmer, and finally stops being luminous (Ceplecha et al., 1998).

1.2.2 History of bolide observation

Recording details from fireballs have multiple observational methods. Satellite and ground recorders get opposite perspectives on the bolide with different ways to identify bolides and calculate their characteristics.

The first ground-based observation from camera footage is Whipple (1954), which recorded 144 meteors orbits, with 52 in Massachusetts and 92 in New Mexico. This experiment used multiple cameras with rotating shutters to mark time intervals at a regular rate, calculating the trajectory, speed, number of fragments and mass (all metrics now standard in describing bolides). In 1959, the Ondřejov Observatory used its two-station rotation shutter observatory to record its first meteor (Ceplecha, 1987). In
the United States in 1970, the Prairie Network recorded the second fireball ever photographed (Ceplecha et al., 1998; McCrosky et al., 1978). Soon the rotating shutters were replaced by the fish-eye lens cameras - starting with the European Fireball Network (EN) in 1963 (Spurny, 1997). Next, the networks adopted digital cameras, with EN adopting the Digital Autonomous Fireball Observatory (DAFO) in 2013 (Borovička et al., 2022). Coupled-Charge Devices (CCD) and video cameras allow new optics and spectroscopy techniques because they provide digital tools and color image information (Trigo-Rodríguez et al., 2006). Multi-station networks are needed to accurately measure the meteor’s trajectory, which has led to the development of many fireball detection networks (Ceplecha et al., 1998). Some of the current ground-based networks include the EN, NASA All-Sky Fireball Network (Ehlert and Erskine, 2020), the Australian Desert Fireball Network (DFN) (Bland et al., 2012), the Polish Network (Wisniewski et al., 2018), a South Korean double-station system (Hinse et al., 2017) and the Global Fireball Observatory (GFO) (Devillepoix et al., 2020). This paper uses data from the Spalding Allsky Camera Network (SACN), where the Florida Institute of Technology is associated.

The most accessible space-based fireball instrument with hemispherical coverage is Geostationary Lightning Mapper (GLM). The GLM instrument is on the Geostationary Operational Environmental Satellites (GOES), originally designed to detect lightning but now also can detect bolides (Goodman et al., 2013; Smith et al., 2021). Additionally, in 2018 a bolide (reflections and selected parts of the trail) was recorded using the Moderate-resolution Imaging Spectroradiometer (MODIS), which is on board the Terra satellite (Borovička et al., 2020). The distance from the atmosphere and the low resolution makes calculating the entire trajectory more difficult from the sky than from the ground (Smith et al., 2021).

Studying bolides is vital to learn the location and composition of objects entering
our atmosphere from outer space (Gritsevich, 2007). Meteors come from the remains of planetary formation in the Oort Cloud (Ceplecha et al., 1998). Detections of meteors, calculating their trajectory, and finding the fragments on Earth are all essential to characterizing and understanding meteors (Smith et al., 2021).

1.2.3 Computer vision techniques

Computer vision is the study of images and video by computer software, and its challenges are prevalent in the literature at the time of writing. Color images in print have each pixel defined by three qualities - the amount of Red, Green and Blue, known as RGB values. Therefore a 2D image is described by an array of pixel values whose dimensions are height by width by 3. Image processing techniques aim to use the connection and relationship between these pixel values to gather the image's content. A standard method is the Hough transform which searches for straight lines in an image (Gonzales and Wintz, 1987). The explanation of how the Hough transform works are illustrated in Figure 1.2. This project aims to analyze video data (discretized into images along different frames) and the composite image to assess if it is a bolide. The goal of detection methods is to identify camera recordings of meteors accurately, and there are two ways these methods are inaccurate: categorizing a recording as including a meteor when it does not (false positive) and categorizing a recording as not including a meteor when it does (false negative).

Many methods use the composite image to search for the meteor’s path. In Gural (1997) and Cheselka (1999), a Hough transform and the parameterization of a straight line identify the meteor’s path. The traditional method in Towner et al. (2020) splits the image into tiles and weeds out false positives with a list of ad-hoc rules.

Methods focusing on video analysis use video data by comparing the differences between successive frames using image techniques. The detection method in Brown
Figure 1.2: This shows how the Hough transform identifies lines. First, the algorithm detects the edges of the meteor path in b, then the two longest lines are identified in green in c and d. The meteor shown occurred on August 5, 2017.
et al. (2007) identifies all the pixels in a given frame that are higher than a chosen threshold. If any groups of pixels are greater than two solar degrees (the right ascension position in the sky) in size in the all-sky frame, the pixel group is flagged to be tracked. A bolide is possible if these bright pixel groups (followed from successive frame to frame) connect on a consistent gradient angle. This method is not fully automated, so people must review the videos and identify meteors (Brown et al., 2007). The technique in Weryk et al. (2007) compares each frame pixel by pixel, and an increase of 70 digital units is considered a potential bolide. In Peña-Asensio et al. (2021), they use the same pixel-by-pixel method but remove false positives by eliminating results that are too small or have an incorrect contour. Then the remaining false positives are eliminated by estimating and then comparing to reality, where a group of pixels would move to the successive frame if it is a meteor using a Kalman filter. The straight line is the shape of the predicted trajectories as potential bolides (Peña-Asensio et al., 2021). Wisniewski et al. (2018) uses a method that subtracts the background star map from a series of mean images and is compared in the video to find the meteor.

This project develops a method using computer vision techniques using both video and composite image techniques. The goal is to improve upon the method introduced in Botella (2018), which tested only 25 meteors from 2017 and 2018, where it got an 80% success rate. This project will re-benchmark the method introduced in Botella (2018) on hundreds of videos from 2017 to 2021 - some with meteors and some without. The new approach focuses on the Hough Transform over brightness and uses composite images and videos to reduce computational time. The SACN plans to grow and includes more nodes, meaning the number of videos and the composite image will soon be on the order of hundreds, which the automatic detection will check daily.
1.2.4 Machine learning techniques

Machine learning is the study of algorithms that train on data to predict patterns as part of artificial intelligence. Deep learning is a part of machine learning which includes neural networks that mimic the learning routines of humans with interconnecting nodes in layers akin to neurons in the brain). These methods have been popular in object identification, a problem that has been difficult to solve by other means. For these methods to be successful, there is a need for a large amount of data that is accurately labeled and represents various scenarios (Liu et al., 2020).

This paper, and many other bolide identification projects, use Convolutional Neural Networks (CNN) to label images. In neural networks, information propagates through layers. To be classified as a CNN, at least one of the layers must use a convolution operation (a specialized linear operation) instead of standard matrix multiplications (Yamashita et al., 2018). In the CNN architecture, first introduced by LeCun et al. (1989), each layer adds complexities and hierarchies such that the way information is propagated through the network in a way the user cannot predict. The highly processed CNN is a black box where data is input, and after rounds of mathematical operations, weights are output. In Krizhevsky et al. (2017), a generic object identifying CNN broke records, and since then, deep learning has been the focus of object identification. The machine learning method in Towner et al. (2020) is a single-layer neural network with hidden units trained on hand-picked examples.

Previous efforts include work with bolide detection as an object identification problem using machine learning and deep learning methods. Smith et al. (2021) used a Random Forest and a Support Vector Machine (SVM) to find bolides. A Random Forest uses tree predictors (which refers to the decision tree model that is a connection of nodes as a non-linear data representation), and each tree is dependent on a randomly sampled vector with better features getting selected with each sample (Breiman, 2001).
SVM is a classification machine learning technique that creates a vector hyperplane to separate two groups based on characteristics. Multiple SVM can be combined for more groups (Hearst et al., 1998). In Longenbaugh et al. (2020), bolides are identified using a Random Forest and a CNN. Smeresky et al. (2021) uses radar data, not camera recordings, and uses Principal Component Analysis (PCA) for preprocessing. PCA reduces the image’s dimensions while identifying components with weights into a unit vector (Gonzales and Wintz, 1987). Then Nearest Neighbors Density Pruning (NNDP) and t-distributed Stochastic Neighbor Embedding (t-SNE) are implemented (Smeresky et al., 2021). This paper uses these techniques to use trees and probability distributions in a more structured way than random forest (Smeresky et al., 2021).

This project will investigate a basic CNN for image identification with a few layers and YOLOv5 (You Only Look Once version 5), a popular object identification software designed to be used off the shelf. By comparing machine learning to the traditional methods, this project looks to find the most suitable strategy for identifying bolides.

1.3 Limitations and Assumptions

Hardware is a notable limitation to the performance of the detection method. The computer memory used in traditional and machine learning methods is limited to the processor and the graphics card. More Graphics Processing Units (GPUs) train machine learning methods better (Chen and Lin, 2014). More Central Processing Units (CPUs) can test more images using traditional techniques to allow more accurate metrics or complex processes. Data is another limitation. More observations of brighter meteors are needed to identify future bolides. Large meteors are rare and more scientifically significant - the debris can be found on the ground, and the composition studied. However, they are less frequent than smaller meteors. More data on the crit-
ical events can be collected over time and will likely improve the training and testing of the methods. Another limitation is that the observing nodes record most all-sky camera recordings in areas with light pollution, street lights, and planes with flickering lights. These make methods that rely on the night sky’s brightness difficult.

There are additional factors that affect the analysis. Computers are instruments, so that they can have errors and machine failures; however, in modern processors, they are negligible. Therefore, the assumption is that the CPU and GPU perform optimally for their specifications in Section 3.1 and that all measurements experience negligible error. Labeling all the video and image data by a single person may lead to human error. A mislabelled camera recording could impact training data or testing accuracy percentages. Even with checking, human error is still possible. For convenience, the accuracy of labeling is an assumption.
Chapter 2

Research Methodology

This chapter covers how the four methods this project tests are carried out and measured. The preprocessing data section describes how the computer program reads and prepares the data. The metrics are the values used to describe the solution quality and method performance. The algorithm section describes each one with flowcharts, code segments and explanations. All the algorithms hope to reduce users’ time to sort camera recordings daily. The Optimized Botella Method is a benchmark to show if the new methods designed can improve its approach. Optimized Botella uses brightness and analyzes the video for every event providing human classification. The Traditional Hough Detection Method is a new approach using the Hough transform, video, and composite image. Two machine learning methods are described - a popular simplistic model in the CNN and a pre-trained, prepackaged software in YOLOv5. Chapter 4 evaluates these four methods on the data described in Chapter 3 and compares their performance on the metrics relayed.
2.1 Data Preprocessing

The major Python libraries used for preprocessing images for computer vision are OpenCV (Bradski, 2000), PyAV (Boers, 2020) and Matplotlib (Hunter, 2007b) (note the others in Table A.1). These Python libraries read in the jpeg or png images (either composite images or frames of the video) like a multi-dimensional matrix in an object known as a NumPy array (Harris et al., 2020). The code reads the integer pixel values as integer values from 0 (no color) to 255 (full color) since it is an 8-bit image (each bit is either 1 or 0 and $2^8 = 256$) in the digital image standard of RGB. Another way to express pixel values is to divide the 8-bit image pixel values by 255 and normalize the image where 0 is black and 1 is full color and the pixel values are floats. The code reads all images (black-and-white and color) in this format, with the dimensions of an array being the image’s resolution. The black and white images have identical R, G and B values.

For all the methods, reduced dimensionality is crucial for the function. A High Definition (HD) color image has an array size of 1080 by 1920 by 3, which is 6220800 8-bit values. With hundreds and thousands of images - some being processed on a pixel-by-pixel level - high dimensionality means intractable computation times or running out of computer memory. Additionally, color is less critical in meteor detection because the background is very dark and the objects are very bright. The colors in a meteor help characterize the composition but not identify it. Therefore the image’s color values are averaged to one value, producing a gray image with a third of the data.

Method evaluation uses two parts - training and testing. Training is about honing parameters, finding the weights for models, and understanding patterns in the method. Testing is about seeing the parameters at their best, sometimes model selection and preventing over-fitting.
2.2 Metrics

The following metrics are prolific in machine learning and object classification in general (Hossin and Sulaiman, 2015). Each metric aims to identify all the goals of an automatic detection method. The computational time is about measuring efficiency. The precision value reduces the amount of time humans will classify false positives. Recall shows whether a method is missing meteors and if it can fulfill the role of facilitating the work of human classification. Accuracy and F-score are overall quality performance metrics to test the balance of the qualities needed. The confusion matrix shows the values used in the metric calculations.

Efficiency is essential because each meteor needs to be identified and studied timely. The faster a meteor is specified, the more rapidly the researchers in the group can write papers and search for potential debris that may have landed. Therefore the shorter the computational time, the more efficient the method is. \textit{Computational time} is the time to predict whether a particular observed event is a meteor. Computational time starts from the very beginning of the method - it includes the time taken to read data, output results, and everything in between. Since this project tests each method on a different number of images and videos, we are comparing the average time per event.

The automatic detection method must classify camera recordings correctly. The first naive approach to solution quality is accuracy. The following formula defines accuracy:

\[
\text{Accuracy} = \frac{\text{number of correct predictions}}{\text{total number predictions}}.
\]  

(2.1)

The benefit of using accuracy is that it describes the number of correct classifications. This metric’s weakness is that the dataset’s composition strongly influences it. If a dataset is 10\% true positive events, and a theoretical method classifies every event negative regardless of content, the method has a 90\% accuracy. Since most camera
recordings do not contain a meteor, this unbalanced composition skews the accuracy. Therefore accuracy speaks more to the dataset’s composition than the method’s quality.

To overcome accuracy weaknesses, splitting the data up to see which events are falsely classified reduces the impact of the data composition. The metric to describe false positives is \textit{precision}:

\begin{equation}
\text{Precision} = \frac{\text{number of correctly predicted meteors}}{\text{number of predicted meteors}} \quad (2.2)
\end{equation}

This metric weighs the impact of false positives. Maximizing the precision is important to see which methods are the best. Low precision means the method could be improved, but human classification must continue. A lower precision requires more time for users to classify recordings.

The metric to describe false negatives is \textit{recall}:

\begin{equation}
\text{Recall} = \frac{\text{number of correctly predicted meteors}}{\text{number of true meteors}}. \quad (2.3)
\end{equation}

Recall must be very high because the process fails if meteors are missed (huge, high-energy events). Therefore false negatives are a bigger problem than false positives.

\textit{F-score} is the harmonic mean of recall and precision, first used in Chinchor and Sundheim (1993):

\begin{equation}
\text{F score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \quad (2.4)
\end{equation}

It is a way to show the impact of both false positives and false negatives as an overall score of the success of the method.

A \textit{confusion matrix} has classifications on the x-axis and predictions on the y-axis, so the quadrants are true positive, false positive, true negative and false negative. The goal is to maximize truthful predictions. Table 2.1 shows the matrix used all the
methods, but the YOLOv5 metric is slightly different. In multiclass problems, the matrix splits up false negatives among all the other classes, and instead of integers, the matrix has a percentage of each category in the dataset.

Table 2.1: Labeled confusion matrix example

<table>
<thead>
<tr>
<th>Predicted class</th>
<th>True class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive</td>
</tr>
<tr>
<td></td>
<td>Negative</td>
</tr>
<tr>
<td>Positive</td>
<td>true positive</td>
</tr>
<tr>
<td></td>
<td>false positive</td>
</tr>
<tr>
<td>Negative</td>
<td>false negative</td>
</tr>
<tr>
<td></td>
<td>true negative</td>
</tr>
</tbody>
</table>

2.3 Algorithm description

2.3.1 Optimized Botella Automated Detection

The Optimized Botella Automated Detection algorithm is from Botella (2018). Botella (2018) tested their automated detection method on a suite of low-resolution video recordings of meteors. In this project, with some semantic improvements, this algorithm acts as a benchmark and is tested on a much larger set of events (some with and some without meteors). Any frames with a total brightness above a certain threshold are considered positive, called pixel analysis. Then the brightest pixel in that frame has its coordinates recorded and compared to other frames and is called the coordinate analysis (Botella, 2018).

There are two reasons why this project reworks the original method from Botella (2018). Firstly the original method is partially automated. The user chooses the threshold values for the brightness test through trial and error until the best plots are output. They produce three plots from the video data - two lightcurves and a trajectory analysis. Then three plots are output as jpeg images for the user to discern whether the video shows a meteor with a qualitative note. This project aims to compare the
benchmark method with other automated methods, which requires the algorithm to output a binary True or False with no user intervention. The second reason is that the method needs to be re-tested on the current dataset to compare a similar dataset. The initial testing used 25 meteor videos because the user-driven parameter tuning and building jpeg plots for every video are computationally expensive. Therefore, testing the original scheme on hundreds of videos is not feasible. So in Optimized Botella, the core of the code is the same as the 2018 method, but the user-driven aspects are now automatic without an image output. To honor the implementation by Botella (2018), the parameters (called “limits” in the paper) are fixed values at the lowest from the range given. The qualitative explanation and graph pattern recognition are automated but search for collinearity. Therefore the core difference between the original and optimized method is the automation of the parameters (parameters for threshold for brightness and the impact of the coordinate analysis).

The Optimized Botella Automated Detection Method uses the same steps as the original method shown in Figure 2.1. First, the code discretized the video into sequential frames as images, and then each frame is tested in sequence in a for loop (a strategy to repeat pieces of code). This method adds a mask to the frame to reduce light pollution from the horizon. The preprocessing process (from Section 2.1) then leads to the gray frame doing pixel analysis and coordinate analysis. The Optimized Botella method predicts a video contains a positive meteor event if it finds any frame to have a meteor from pixel or coordinate analysis. All the frames undergo both the pixel and coordinate analysis regardless of the results from previous frames, which leads to some redundancy in the for-loop.

The pixel analysis aims to follow changes in the pixel values between consecutive frames, which means bright objects in a different position from the previous frame are considered a meteor. The pixel analysis first turns the current gray frame into a binary
video discretized into frames

mask horizon lights

any positive events found are output

Figure 2.1: This is the flowchart for Optimized Botella Automated Detection. From left to right, it displays the different steps in the code that each video goes through. The method discretizes the video into consecutive frames that are each masked and runs pixel and coordinate analyses.

Figure 2.2: This is the flowchart of the pixel analysis Optimized Botella Automated Detection. The analysis subtracts the binary versions of two consecutive frames from one another and then adds together all the pixels to test for the brightest moving light.

image using a threshold of 0.3 - i.e., all gray pixels (with white as 1 and black as 0) of value 0.3 or higher become 1 and lower than 0.3 become 0. The same processing happens to the previous frame, creating the absolute value difference between each frame. The binary image of the differences should show any bright moving objects - of which a meteor would qualify. If there are more than six white pixels in this binary image, the method classifies a video as a positive meteor. The pixel analysis is in Figure 2.2 flowchart.

The coordinate analysis aims to use the position of the brightest event in the sky and see if its trajectory suggests a potential meteor. The flowchart in Figure 2.3 visualizes the process. The method subtracts the gray frame image from the background (the
The gray difference image would only show a bright object that shows up during the video. The method identifies the brightest pixel’s position and calculates the center of mass in two tiers - one for those where the brightest pixel’s value is greater than 0.7 and the other where it is less than 0.7. If the brightest pixel is less than 0.26 and the center of mass is the same as the previous frame, the coordinate analysis adds the center of mass to two lists - the center of mass is a pair of x and y coordinates. If the final lists of coordinates (one for x and one for y) have three consecutive values that are 80% collinear, then the coordinate analysis shows a positive meteor event. This collinearity test is my interpretation of the reading of the third plot in the original Botella (2018). Searching for straight-line trajectories prevents the impact of the background noise that appears to move between frames but is not a meteor. The code shown below calculates collinearity with the gradient between two pairs from three consecutive non-zero centers of mass.
count = 0
n = len(x_arr)
x = []
y = []
traj = False
for i in range(n):
    if x_arr[i] + y_arr[i] > 0:
        count += 1
        x.append(x_arr[i])
        y.append(y_arr[i])
    else:
        count = 0
        x = []
        y = []

if count == 3:
        traj = True

The complete code for this method is in Appendix A.1. In Botella (2018), the success rate claimed is 80% (testing only 25 meteor videos with 20 accurately identified). The paper suggests that this method more easily detects bright instances, and looping over all the frames of all videos is time-consuming. Since the code’s core is similar, Optimized Botella’s weaknesses are likely the same. The optimized parts of the method move the results from the qualitative to the quantitative. In Botella (2018), the user reads the three plots together using clues from the trajectory to get information from the lightcurves. Quantizing and automating this process may remove the user’s ability to confirm the event status using the video, which was encouraged in the paper.
2.3.2 Traditional Hough Detection Method

This algorithm implements the Hough transform on a composite image (either from the SACN or built from the video frames). The literature focused on identifying the trajectory in composite images (Gural, 1997; Cheselka, 1999) and calculating movement between frames in videos (Brown et al., 2007; Wisniewski et al., 2018). This method combines the two to improve performance (make the video method more efficient and strengthen image results). The Hough transform is used in image analysis to look for straight lines to identify the meteor’s path and in video analysis to find meteor light in the frame. The extensive grid search automates parameter selection over the different thresholds. A comprehensive grid search requires rerunning the same data on the same algorithm with suitable parameters and is computationally expensive for video analysis in particular.

The Hough transform originates from a patent from 1962 by P.V.C. Hough (Hough, 1963) from the idea at the core of the modern Hough transform: geometric rules can identify straight lines from collinear points in the image plane (Hart, 2009). The Hough transform’s current formulation uses polar coordinates (parameterization of the pixels in terms of the distance $\rho$ and angle from the origin $\theta$), a binary image after edge detection, and support lines (the normal to lines connecting pixels) (Duda and Hart, 1972). These components are shown in Figure 2.4, where the three pixels are labeled red, blue and green. The Hough transform compares the values of $\rho$ and $\theta$ to show if the pixels are in a line. The Hough transform shows this by cycling through different angles for each pixel and then selects angles where the support lines line up as a vote (Hart, 2009). The more votes there are, the more likely the algorithm has found a line. In Figure 2.4, the red and blue have a very similar $\rho$ value for the same $\theta$ value, while the green is 2 pixels larger. If the resolution of the Hough transform is 1, red and blue are in line, but green is not. If the resolution is 3, then all three dots are
in line. The Hough Gradient Method uses gradients like the edge detection method to improve efficiency. The edge detection step is called Canny Edge Detection from Canny (1986). Canny Edge Detection calculates differences in the pixel values as a gradient, and the maximum local gradients are the edges. Therefore sudden, large color changes are counted as an edge. The Hough Gradient Method calculates local gradients, the same as Canny Edge Detection, and uses that to infer the support lines and make comparisons. The gradient is from the Sobel operator, defined by using a convolution of square masks to find the local gradient (Kanopoulos et al., 1988). The OpenCV function performing the Hough transform is Probabilistic Hough Transform, which does not check all pixels but infers the pattern from a random sample (Kiryati et al., 1991).

The original Hough transform was for straight-line detection, but the same principles apply to other shapes. The video analysis part of the Traditional Hough Detection method uses Hough Circle Detection. Straight lines can be described fully with two variables \((\rho, \theta)\), while circles are described by three variables \((x_{\text{centre}}, y_{\text{centre}}, r)\) because
\[
(x - x_{\text{centre}})^2 + (y - y_{\text{centre}})^2 = r^2
\]
is the equation of the circle. Therefore Hough Circle detection takes two steps to create two 2D problems. This method starts with a grayscale image (not a binary image like the straight-line version). The first stage looks for \((x_{\text{center}}, y_{\text{center}})\) and does edge detection (the gradient techniques are discussed in the last paragraph explaining why these work well simultaneously). The second stage uses the candidate centers from stage 1 and looks for appropriate \(r\) values (Davies, 1988; Ballard, 1981).

Figure 2.5 is the flowchart describing the Traditional Hough Detection Method. Both the composite image and video analyses remove the background before identifying patterns. The best way to build a background image is to take the video’s first five frames and average them to produce a less noisy background than the first image.
Figure 2.4: This is a binary edge detection image of a meteor path with Hough line calculations drawn to compare $\rho$ values for a set $\theta$.

The composite image analysis goes first because it is much less time-consuming than the video analysis and then reduces the average computational time per video. Both analyses result in a binary output of True or False (represented with T or F respectively) of “this analysis predicts a meteor” (T) and “this analysis predicts there is no meteor” (F). The structure of the Traditional Hough Detection Method was developed and
Figure 2.5: This is the flowchart for Traditional Hough Detection. After finding the background image, the composite image analysis looks for meteors before the video analysis checks events that test positive.

Honied using the training set. The results showed that the composite image and video analyses have a high recall and low precision while the composite analysis has a shorter computational time. Therefore, the composite image analysis runs first to increase the precision and decrease the computational time, and then the video analysis checks the composite image analysis’s predicted meteors.

The composite image analysis focuses on straight lines in the composite image where the meteor’s trajectory approximates a straight line. First, the code subtracts the background from the composite image. This image is converted from an RGB image to a gray image as a requirement for Canny Edge and Hough detection. The gray image is filtered twice to improve the quality of the edge detection and reduce artifacts. Filtering is a neighborhood operation where matrix multiplications occur in successive patches over an image. The median filter of size 3 removes hot pixels or bright stationary objects in the sky that is not in the background by taking a 3-by-3 matrix and making the central pixel the median value of all 9 pixels. The Gaussian filter of sigma 1 removes noise from the light pollution, hot pixels and any artifacts from the discretization of frames from a video. Canny edge detection is performed on this filtered gray image and returns a binary image of the edges of objects. Canny
edge detection has two thresholds - every intensity gradient below the low-bound is not an edge, and everything above the high-bound is a “sure edge”. If the intensity is in between, it is only an edge if connected to a “sure edge” Bradski (2000). From looking at training data, the low-bound selected is 15, and the high-bound is 200 (using the 8-bit image format) because the events are very bright and the background is very dark. These edges go into the Hough transform that uses parameters from a grid search. The result is all the lines detected - many of which are small with duplicates. The grouping of the Hough results focuses on connecting pairs of lines closer to half the length of the long line. To test the alignment, the dot product of the vectors describing the lines is 0.99 or greater (i.e., the angle between them is less than 8.1 degrees). Then if the longest line found is greater than 15 pixels, the analysis gives a positive result to go to the second analysis. Short lines are likely noise or minimal meteors, not bolides.

```python
lines = cv2.HoughLinesP(edges, # Input binary edge image
                        1, # Distance resolution in pixels
                        np.pi/180, # Angle resolution in radians
                        threshold=25, # Min number of votes for valid line
                        minLineLength=2, # Min allowed length of line
                        maxLineGap=10 # Max allowed gap between lines
                       )
```

The video analysis focuses on the speed of meteor-shaped objects between frames. The goal is to check whether the lines found are related to meteors or are stationary flashes, slow-moving planes, or artifacts to filter out false positives without creating more false negatives. Maintaining a high recall justifies the decision tree for the flowchart in Figure 2.5, where the positives from the composite analysis run through the video analysis. For the video analysis, the video is discretized into frames using PyAV, and the code loops through the frames. Each frame is processed by subtracting the background and turning the difference into a gray image. The gray image is
Figure 2.6: This is the flowchart for Video Analysis of Traditional Hough Detection. The analysis discretizes the video then the procedure loops over each consecutive frame. The procedure formats the frames so the Otsu test can decide which circle test the procedure performs before calculating the frame’s speed.

Blurred to reduce noise and improve edge detection, thresholding, and the Hough Circle Detection. The blurred image then runs through an Otsu threshold.

Thresholding turns a gray image into a binary image, and the difficult part is knowing where to set the cut-offs (which pixels should be 1 or 0). Automating thresholding can be difficult because bright and dark images need different threshold points. The Otsu method selects a threshold value using a histogram and variance ranking method, which returns a binary image and a threshold value Otsu (1979). A very low threshold value indicates that a frame is very dark, and because the images are noisy, dark images perform poorly in the threshold. This Otsu threshold value separates bright and dark frames. Bright images first get tested for Hough Circles with a check for false circles. Since not all lights from the ablation of the meteor are circular, the frames that fail the Hough Circle test progress to the approximate circles test with the dark frames. Frames that pass the Hough Circles test move on to the speed calculation.
circles = cv2.HoughCircles(final_im, # Gray image
                      cv2.HOUGH_GRADIENT, # Hough Method
                      1, # Accumulator resolution
                      20, # Minimum distance between circles
                      param1 = 1, # Stage 1 for the edge detector
                      param2 = 10, # Stage 2 for the accumulator
                      minRadius = 1,
                      maxRadius = 70)

The dark frames that are inappropriate for the Hough Circle image and the brighter images that do not have a circle undergo the approximate circle search. The approximate circle search, like in Brown et al. (2007), uses the brightest pixel from the gray image difference between the frame and the background. That position is stored, and the blurred processed frame (step 2 in Figure 2.6) is thresholded with a set parameter. Figure 2.7 visualizes this search where the position of the brightest pixel (the red dot) is projected onto the binary threshold image result. The 4-by-4 pixels square should be bright (where 10 or more out of 16 pixels are 1), while the green border should be dark (where 10 or fewer out of 336 pixels are 1). This geometry allows the video analysis to identify small and large light from the meteor (4 to 40 pixels long) in the frame. It also prevents flashing lights (typically larger than 40 pixels in a frame 1600 pixels wide) from being falsely identified as meteors. Ablations larger than 40 pixels are likely identified as circles by the Hough Circles. If the video analysis misses a meteor in one frame, it can locate it in the other frames it appears in.

Each frame returns a position - from Hough Circles, Approximate Circles, or if both are false, the potential meteor position is (0,0). The Euclidean distance between the current and previous frames’ positions is calculated as the speed (since frames are equidistant in time). The acceptable speed range is 2.5 to 15 pixels - found from
training the model where the noise was larger than 15 pixels, and planes move 1 or 2 pixels per frame ($\sqrt{2}$ was the modal speed of planes). Using speed as an indicator of meteors is akin to using a Kalman filter in Peña-Asensio et al. (2021).

The traditional Hough Detection has some weaknesses that future work can improve upon. The composite image analysis focuses on straight lines, while the fisheye all-sky
lens can curve the path making it difficult to find lines with more than an 8-degree curve. The composite analysis also makes it challenging to identify glitches, rain, lightning, and clouds as negative because they can produce light in unusual shapes and lines that the Traditional Hough Detection method can misidentify as meteors. Additionally, planes’ lights are inconsistent and appear at different speeds depending on the altitude. One of the nodes records data from the roof at Florida Tech, where there is a nearby airport, flight school, and frequent rocket launches, all creating unusual false positives to plan for. The video analysis ensures the identification is for a moving round object at an appropriate speed (slower than a glitch but faster than a plane or rocket launch). In the future, there could be a focus on minimizing noise with video analysis. The meteor is often a tiny dot, so filters (the primary tool for removing noise) must be very small and less effective. Artifacts, like the halo around bright lights, happen when you subtract the background from the frame. Lastly, there are many more negatives than positives, so building a high-precision method with high-quality rules is difficult.

The complete code for the Traditional Hough Detection Method is in Appendix A.2. Machine learning - as the boom in artificial intelligence - has replaced most traditional computer vision functions with deep learning methods and neural networks. The benefit of conventional methods that search for image patterns is that the techniques are not a “black box” (Liu et al., 2020). Machine learning methods are described as “black box” because we cannot know how they identify objects because of the highly processed layers or training. One of the difficulties with traditional methods is parameterization and choosing constants. Grid searches are computationally expensive. An alternative could include data reduction with resolution reduction, PCA, or new methods which require less information. Using a traditional method (high recall, low precision) with a machine learning algorithm (low recall, high precision) could give more information to the user. Most methods in the literature have many steps making grid searches even
more complex on limited software. This methodology proves a Hough transform can form an essential part of a multistage method tested with a higher-capacity CPU.

2.3.3 Convolutional Neural Network

The neural network chosen is a basic structure typical in image identification to training on the limited capacity of our single GPU (Pedersen, 2020). The library Keras (Team, 2020) is for high-level neural networks, and it runs on top of TensorFlow 2.9 (Abadi et al., 2016), which is famous for machine learning tasks. The network has been successfully tested in Pedersen (2020) on image classification on the well-known MNIST dataset (reading handwritten numbers). This paper refers to this naive approach to a CNN and “the CNN” where YOLOv5 - which includes a CNN backbone - as “YOLOv5”.

The structure is in Figure 2.8. The following chart shows how the data flows in the CNN is implemented. As described in Chapter 1, Convolutional Neural Networks are layers of mathematical operations that learn on data in a method inspired by the human brain. Figure 2.8 shows the organization of the layers (for more details of the logic behind the CNN, their inventor has an explainer in LeCun et al. (2015)). The CNN implemented includes two convolutional layers, each with corresponding downsampling using max-pooling, and two fully connected layers. The convolutional layers use kernels to convolve the image to create feature maps. The pooling layers reduce the dimensionality of the feature maps and parameters (Gholamalinezhad and Khosravi, 2020). These layers are flattened from 2D to 1D before the fully connected layers. The first fully connected layer uses Rectified Linear Units (ReLU) as an activation function, where ReLU sets negative values to 0, and positive values are linear with a slope of 1 (Agarap, 2018). The second fully connected layer (and final layer) uses the softmax classifier to classify images. Softmax comes from linear regression for
solving the probability of getting each one of mutually exclusive classes (Wolfe et al., 2017). Additionally, the model weights are optimized using Adam, a famous stochastic optimizer used for noisy problems (Kingma and Ba, 2014).

```python
from tensorflow.keras.optimizers import Adam
optimizer = Adam(1r=1e-3)
model.compile(optimizer=optimizer)
```

The GPU’s capacity means that the size of the original images is too large with too many pixels. A grid search is a method to optimize the parameters for optimal final results. For the grid search, the parameters varied are scale percentage (the percentage of the original width and height of the new images) and the number of epochs (an epoch is a run of the dataset through the algorithm (Brownlee, 2018)). A higher scale percentage means more data in each image, preventing tiny meteors from being missed. A lower percentage results in more small events being ignored but is less computationally expensive (in terms of computational time and memory). More epochs allow for more training of the model. However, too many epochs can lead to overfitting - where the model is fitted closely to the training data and cannot take the differences in a testing set (Brownlee, 2018).

```python
width = int(grayImage.shape[1] * scale_percent / 100)
height = int(grayImage.shape[0] * scale_percent / 100)
dim = (width, height)
resized = cv2.resize(grayImage, dim, interpolation = cv2.INTER_AREA)
```

The complete code for the CNN is in Appendix A.3. CNNs have advanced in recent years, and in the literature, variations of the CNN have experienced high accuracy rates Liu et al. (2020). The weakness is that the neural networks’ success is based on how representative their data is (Krizhevsky et al., 2017). The dataset size is relatively small (large groups often include tens of thousands of datasets for training, and we
Figure 2.8: This is the architecture of the Convolutional Neural Network. The pooling layers are smaller than the convolutional ones because their job is to reduce the dimensions of the feature maps. The fully connected output layer condenses the feature map into a classification prediction.

only have about a thousand), so this bias is evident in the results. The strength of this method is that more data continues to be captured by the SACN. Additionally, this is a promising field, especially in video object recognition, and the hardware (GPUs and neural processing units (NPU)) continues to improve Ishida et al. (2020). Therefore this methodology is a proof of concept that can be easily enhanced in future work.

2.3.4 YOLOv5

YOLOv5 (meaning You Only Look Once) is a powerful object identification software from Ultralytics Jocher et al. (2020). YOLOv5 has been applied and modified in many fields for object identification in video and images (Karthi et al., 2021). It has outperformed popular methods like Resnet and Fast RCNN (Luo et al., 2021).

The way that YOLOv5 works is the model uses dense sampling within the image. It has three parts known as the backbone, neck and head. The backbone is EfficientNet, a CNN that scales and balances the depth, width and resolution to be more accurate and efficient (Tan and Le, 2019). This backbone is a pre-trained network with image feature representations - it acts like a head start when trained on a custom data set. The neck
Figure 2.9: The anatomy of an object detector like YOLOv5. The EfficientNet CNN into the BiFPN layer nodes until the class prediction Tan et al. (2020) includes BiFPN (Bidirectional Feature Pyramid Network), which merges features to optimize prediction and to map to objects Tan et al. (2020). The head predicts where the box is, its shape from the object, and the probabilities of each class prediction. The anatomy for object detectors like YOLOv5 is in Figure 2.9. An example output showing the box around the object with the probability of the class prediction is in Figure 2.10.

In all the other methods, a composite image has the label “contains a meteor” or “does not contain a meteor”. However, to train this model, a person must manually draw a box around the object in the image, which is time-consuming. This project runs YOLOv5 on the labeled composite images with meteors and typical light sources. The goal is to have the result with a few photos with highly accurately labeled data as a proof of concept for more data in the future. The labels chosen were bolide (for meteors of all sizes), lights, moon, cloud and planes as the identified classes of objects. Therefore the metrics are not just “bolide: True or False” but each class is tested, and the output is labeled in a frame within the image.

Jocher et al. (2022) designed YOLOv5 to be optimized for high accuracy, precision and efficiency. Therefore, after the time-consuming labeling, the user does not build
Figure 2.10: These are some example results to YOLOv5 for composite images from the SACN data. The color-coordinated labels show how images are output by YOLOv5.

any code and can just run the script after downloading software with notifications on updates by cloning for GitHub. The adjustable parameters are the number of epochs - an arbitrarily high enough number works because the training outputs the best and last models. Re-scaling images are necessary, as it was for the CNN. The scaling percentage is from the CNN grid search since YOLOv5 has a CNN backbone.

The parameters I run on the code for the YOLOv5 are in Appendix A.4. YOLOv5 code is constantly updated. I used YOLOv5s v7.0, which I cloned from https://github.com/ultralytics/yolov5. The YOLOv5 “s” model is the most miniature - it has the least node connections and takes up the least amount of computer memory. YOLOv5 comes in different sizes, and “s” was selected to suit the available hardware and labeled data. Jocher et al. (2022) released YOLOv8, and groups have built on and added to different YOLO methods proving this field keeps improving (Wang et al., 2021, 2023; Al-Ahmad, 2022). YOLOv5 is still essentially a black box making it difficult to know why and how the method returns results. Labeling is time-consuming, so more
accurately labeled data would be helpful. Like in the case of the CNN, hardware is a bottleneck since lots of computational power is needed. YOLOv5 is pre-trained, which should help the training to have a head-start. This method could be the easiest to scale and improve with more data and training. Artificial intelligence for object detection is prevalent in the literature and is in the zeitgeist at the time of writing. Therefore, there will likely be improvements in software like YOLOv5 soon, and it should make fully automated bolide detection a reality.
Chapter 3

Datasets and Data Collection

This chapter is about data handling. First, to contextualize all the discussion on computational performance, the hardware and software where all the data is processed and the methods run is detailed. The following sections describe the Spalding Allsky Camera Network (SACN) and its two instruments, which provide the data this paper analyzes. Lastly, this explains how the data is segmented to ensure that the models are tested fairly. By detailing how the data is collected and segmented within the constraints of the hardware and software, this chapter is the scaffolding and context for the results in the next chapter.

3.1 Hardware and Software Specifications

All the testing and training results are run on the same computer (to ensure consistency) with the hardware specifications in Table 3.1. This project uses Python 3.8.10 for all the pre-processing, training and testing. The GPU is used to run the CNN and YOLOv5 using Tensorflow. All methods use the CPU. Displaying these specifications allows for the replicability of the results and contextualizes the computational time.
Table 3.1: Hardware and software specifications

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU:</td>
<td>GeForce GTX 1060 WINDFORCE OC 6G</td>
</tr>
<tr>
<td>OS:</td>
<td>Linux 5.15.0-56-generic (62 20.04.1-Ubuntu)</td>
</tr>
<tr>
<td>Architecture:</td>
<td>x86_64</td>
</tr>
<tr>
<td>CPU:</td>
<td>16 AMD Ryzen 7 1800X Eight-Core Processors</td>
</tr>
<tr>
<td>RAM:</td>
<td>31Gi</td>
</tr>
<tr>
<td>Program:</td>
<td>Python 3.8.10</td>
</tr>
</tbody>
</table>

3.2 Data Sources

The SACN includes two instruments as data sources: the SkySentinel and the PiSentinel instruments (Hughes et al., 2022). The SkySentinel instrument consists of a legacy black-and-white low-resolution camera with devices worldwide. The newer support systems are the PiSentinel instruments with HD full-color cameras and Raspberry Pis. The project gets most of its data from the PiSentinel instrument (around 95 %) because the nodes include radiometers that can confirm the camera data. Additionally, these are the newer technologies with plans to expand.

3.2.1 Spalding Allsky Camera Network

Mr. R. E. Spalding developed the Allsky Camera System from Sandia National Laboratories in 2005 (Spalding, 2005). The system’s goal was to monitor, track and analyze the night skies for the pursuit of science and as part of nuclear test treaty monitoring for the US government. This project is now a joint operation by the Florida Institute of Technology and the SkySentinel LLC called the Spalding Allsky Camera Network (Palotai et al., 2018).

The network aims to refine the energy and trajectory calculations, which is vital to know meteors’ origins and composition. The current number of active nodes at the time of writing is in Figure 3.1. The network provides software tools, including calibration, artifact removal, and finding trajectory Botella (2018).
3.2.2 SkySentinel Instrument

The SkySentinel instrument schematic is in Figure 3.2. The outside is the housing which includes a PVC pipe with a weatherproof acrylic dome on top. The HiCam HB710E Sony Ex-view HAD 1/2” size CCD camera with a Rainbow L163V lens looks up from the dome. The camera setup connects to a video capture card and a 12V DC power adapter. The unit only records during the nighttime because the photo sensor on the camera turns it off when there is sunlight. The system has a temperature and moisture control system consisting of heaters, a thermostat, and a fan to maintain an internal temperature of 100° F. The video signal (29.97 frames per second in a 640 by 480 format) is processed and streamed to a disk from the video signal cable (SkySentinel, 2013; Weryk et al., 2007; Botella, 2018).

The software tools (WSentinel) run on a computer connected to the video signal. The software produces four files for each event - a text file containing the event times, a CSV file with event metrics, a composite image of the frames, and an mp4 video. Each
node automatically transfers this data to the centralized SkySentinel server. Until now, human interaction was necessary to verify actual meteors and eliminate false positives (Spalding, 2005; Botella, 2018; Hughes et al., 2022).

3.2.3 PiSentinel Instrument

PiSentinel uses a USB webcam, Raspberry Pi, and a GPS module to record meteors. Hughes et al. (2022) introduces the system, and one of the nodes captured a fireball over the Bahamas. Hughes et al. (2022) uses the PiSentinel data to corroborate the data from the SkySentinel instruments. There is an ongoing effort at Florida Tech to improve the node setup and to deploy additional nodes for increased coverage. At the time of writing, there are three nodes with one on the roof at the Florida Institute of Technology - their locations are in black in Figure 3.1.

The housing is a PVC box with two acrylic domes - where the camera peers through one and a radiometer through the other. Silica bags absorb moisture, and a 3D-printed structure lifts the setup above the bottom and holds the pieces in place. A
fan controls the temperature. The camera is a Sony IMX322 complementary metal
oxide semiconductor (CMOS) image sensor with a 170° fisheye lens producing full-
color HD 1920 × 1080 2.12 megapixel resolution. The camera records at 30 frames
per second and records stars as bright as -2 and as dim as +2 in apparent magnitude.
The GPS notes the time and turns the camera on at night and off during the day. All
data from the day, events, and their results are stored locally on a Raspberry Pi, the
attached computer within the node. Daily code run on the server pulls data from those
computers (Hughes et al., 2022). Figure 3.3 shows the top-view image of the hardware
without the lid.

The benefit of the PiSentinel instrument is that it is connected to the camera and
can transfer the data to the server. The software used by this instrument is called
Pi-Sentinel, and it catches the events where the brightness of the sky changes (Hughes
et al., 2022).
3.2.4 How Automated Detection Fits In

Both instruments produce many false positives since the software records any change in brightness. Therefore, a person must view the videos (99% each a few seconds long) and the composite image and classify the events individually, constantly watching videos multiple times to confirm. This data requires a quick turnaround because details on fireballs are eminently noteworthy. The goal is that by running a daily script every morning after the night’s events, the automated system posts an auto-filled report with light curves and trajectory data to the site for ease of use. This process stage needs improvement, and this project aims to address it.

3.3 Data Classification and Segmentation

An essential step is labeling the data by hand so that this project can compare the correct classification to the model’s predictions. All these videos and images were labeled by hand as recordings with meteors and recordings without meteors so this project can test each method’s accuracy. A positive meteor event must only be visible to the naked eye to classify as a positive sighting, meaning small meteors up to bolides are positive.

The false positives often exhibit similar features, which explains why the software predicted a meteor. The dataset attempts to represent all these reasons to test if the methods are robust to errors. The false positives are often a plane’s lights on the wings or the tail. These lights are usually colored red or green and frequently blinking, but these all vary. Very bright sky events are also captured, like lightning or rocket launches. Lights from cars, people, bicycles, or street lights covered by trees are all sources of false positives.

The SkySentinel dataset is on a centralized server that posts to their website (Tim-
Table 3.2: Number of events used

<table>
<thead>
<tr>
<th>Method</th>
<th>Train</th>
<th>Test</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimized Botella Automated Detection</td>
<td>-</td>
<td>288*</td>
<td>288</td>
</tr>
<tr>
<td>Traditional Hough Detection</td>
<td>122</td>
<td>318</td>
<td>440</td>
</tr>
<tr>
<td>CNN</td>
<td>876*</td>
<td>482</td>
<td>1358</td>
</tr>
<tr>
<td>YOLOv5</td>
<td>197</td>
<td>95</td>
<td>292</td>
</tr>
</tbody>
</table>

* Includes some SkySentinel data

othy Penn, 2022), and the PiSentinel data is sent from the nodes to a centralized machine. Both data sources collect recordings nightly and thus give access to thousands of positive and negative events. The hardware of the Optimal Botella Method and Traditional Hough Detection Method is limited to a few hundred because they use video data and loop over each frame. To allow for testing on more events, parallelizing the code or using improved hardware would solve the problem in future work. To improve training, CNN requires thousands of composite images, but labeling data is time-consuming. There is not a pool of old labeled data because the SACN website only posts positive meteors. Therefore, the false positives were collected from the unlabelled data from the PiSentinel set. The number of events used per method is in Table 3.2.

Training and testing sets for the machine learning methods (CNN and YOLOv5) have about a two thirds-one third standard split (65-35 and 67-33, respectively). The test videos had more negatives which meant longer videos and more frames, affecting the ratio for the YOLOv5 video. These events are hand-sorted by the author. Large, bright meteors are rarer than smaller meteors, so more small meteors are in the training data to train the models, even though detecting large bolides is more critical. Therefore, the data is imbalanced within the two classes, affecting the results of the models generated (Khan et al., 2018). There are two datasets so that the best parameters can be found in a grid search while preventing over-fitting (where the model is tailored to
the specific data and does not generalize to future datasets.

The Traditional Hough Detection Method flipped this ratio with 28% training. Unlike machine learning, the training is not for the model to learn to create patterns between images but for finding appropriate parameters with a grid search. The Traditional Hough Detection method’s grid search is time-consuming, so keeping most of the events for testing metrics is more time-effective.
Chapter 4

Results

This chapter shows all four methods - Optimized Botella Automated Detection, Traditional Hough Detection, Convolutional Neural Network and YOLOv5. Optimized Botella is the benchmark method the other three methods try to outperform. All three new methods have a training and testing phase. The training phase is where the model is put together - the results are better because the technique is designed to perform best on this data. The testing is more unseen to try and test if the method is too specialized for one dataset and prove the replicability of the method. The metrics described in Section 2.2 are calculated for each method and described in tables and plots. There are also figures with composite images of the successes and failures of the method to show the strengths and weaknesses of each method. The next chapter compares these methods to their test result metrics and proposes a combined approach.

4.1 Optimized Botella Automated Detection

(Botella, 2018) tested their method only on positives to get an 80 % accuracy. It was a partially automated method - so the user used three plots to discern whether there
was a meteor. By automating this decision, the method becomes Optimized Botella, which this project can test against hundreds of events as a benchmark. This project aims to show if the methods designed can improve on the last method created in 2018.

The Optimized Botella method uses the given parameters from Botella (2018), and there is no additional grid search or training compared to the original to make the results a fair comparison. Optimized Botella represents using brightness to identify meteors and only using video analysis. The confusion matrix in Table 4.1 shows that the method classifies videos as negative more easily than positive. The metrics are in Table 4.2 where the accuracy is high since the dataset has a lot of negatives and the method favors classifying events as negative. The recall value is very low since there are many false negatives. The automation method must not miss fireballs (the brightest meteors), so a very low recall makes Optimized Botella ineffective as an automated method.

Table 4.1: The confusion matrix for the Optimized Botella Method.

<table>
<thead>
<tr>
<th>Predicted class</th>
<th>True class</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive</td>
<td>Negative</td>
</tr>
<tr>
<td>Positive</td>
<td>13</td>
<td>17</td>
</tr>
<tr>
<td>Negative</td>
<td>66</td>
<td>192</td>
</tr>
</tbody>
</table>

Table 4.2: The metrics from the Optimized Botella Method.

<table>
<thead>
<tr>
<th></th>
<th><strong>Test</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (%)</td>
<td>71.18</td>
</tr>
<tr>
<td>Recall (%)</td>
<td>16.46</td>
</tr>
<tr>
<td>Precision (%)</td>
<td>43.33</td>
</tr>
<tr>
<td>F-score (%)</td>
<td>23.85</td>
</tr>
<tr>
<td>Time (mean) (s)</td>
<td>99.43</td>
</tr>
<tr>
<td>Time (standard deviation) (s)</td>
<td>250.50</td>
</tr>
</tbody>
</table>

Figure 4.1 shows the composite images from the videos that Optimized Botella accurately identified as meteors. It includes the brightest bolide in this test dataset, “s20210413_021644_675”, a fireball over the Bahamas on April 13, 2021, that any au
tomated detection method cannot miss. The pattern between these four images is that the background is very dark (no lights or clouds reflecting moonlight) or very bright (a bolide big enough to light up the entire sky). These backgrounds make a comparison of brightness easier because there is a significant difference between the brightest pixels value of the bolide and the background pixels (the first frame). In all these events, the meteor is the brightest in each frame. This figure shows the strength of the Optimized Botella method - it identifies the most important events. Still, it misses medium and small meteors because the difference in brightness between the background and the brightest pixel in the meteor is too small for the code to pick up.

![Figure 4.1](image1)

**Figure 4.1:** These are a selection of composites of true positives from the Optimal Botella Method. The bottom right image is a bolide, while the other three are meteors.

Optimized Botella Automated Detection uses brightness as the primary way to decide which events include meteors. The disadvantage of identifying meteors as the brightest pixel in the image is that medium or small meteors get excluded - especially
if there is light pollution. This method also has no noise reduction for the frames and relies on a fixed number of pixels for comparing frames’ brightness. Gaussian noise, hot pixels, and straight-line artifacts across the frame make accurate image processing calculations challenging. This noise lowers the method’s precision because artifacts get to read as meteors since they are the brightest pixels in the image.

The coordinate analysis fails in dim events because it tracks the noise, not the meteor. Part of the pre-processing includes adding the mask to reduce noise by obscuring the horizon. A mask can be a weakness since many events happen on or near the horizon.

Additionally, this method discretizes every video and loops around every frame, making the computational time very long and unpredictable from day to day. A cloudy day during a new moon versus a full moon clear night with many planes and meteors means that the Optimized Botella method could run for less than an hour or a couple of days respectively. This is evident in Table 4.2 where the time averages 100 seconds per video with 250 seconds of a standard deviation - the very high standard deviation means the average will likely vary day-to-day. The key parameters do not change - using methods that rely on dynamic thresholding will likely yield more accurate results.

4.2 Traditional Hough Detection Method

The Traditional Hough Detection Method aims to use image processing principles to use patterns in composite images and videos. The main goal is to avoid missing medium or large meteors and bolides, so a high recall is a focus.
4.2.1 Training

The training in this traditional method focuses on the grid search on parameters. One of them is the speed which calculates the Euclidean distance between two consecutive frames of the potential meteor. The other vital parameters are the threshold values - i.e., the pixel value of the gray image that separates which pixels are black and which are white in the binary image. This study focused on the grid search parameters: Hough parameters, speed of pixels between frames, and thresholds for binary images – all marked clearly in the code in Appendix A.2. The main parameters are minlinelength and vid_thresh, which the training phase adjusts to maximize the recall value without reducing the precision. Adjusting parameters for the Traditional Hough Method requires lots of decisions by the user. Therefore if a different user with the same method ran a grid search again but with other goals, the outcomes of the technique would be different. Accordingly, future work could change the same process with a new grid search focussing on various metrics.

The confusion matrix for the training data for the Traditional Hough Detection Method is in Table 4.3. The training data contains 34 % positive meteors, which are not representative of a typical day’s events; however, it ensures the training investigates a diversity of positive meteors. Figure 4.2 shows all eight composite images of the false negatives. All the false negatives in the training set appear thin, dim and short in the composite image, so they are low-brightness meteors with trials caused by dust-size objects. The algorithm can miss these without incident since objects the size of millimeters are not essential for our research group to track.
Figure 4.2: These are all the images of the composites of false negatives from training in the Traditional Hough Detection Method. All of these have very small meteors - some of which are hard to see.
Table 4.3: The confusion matrix for the training data for the Traditional Hough Detection Method.

<table>
<thead>
<tr>
<th>Predicted class</th>
<th>True class</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive</td>
<td>Negative</td>
</tr>
<tr>
<td>Positive</td>
<td>32</td>
<td>30</td>
</tr>
<tr>
<td>Negative</td>
<td>8</td>
<td>46</td>
</tr>
</tbody>
</table>

4.2.2 Testing

This step is crucial for validating the grid search from the testing set and showing how well the parameters work on a different dataset from the training. Since the parameters the code uses are not explicitly tailored to the new dataset, the performance metrics will likely decrease. This testing dataset represents a week’s worth of data from an observing node with 10% positive events. The confusion matrix for the testing set on the Traditional Hough Method is Table 4.4. The computational time is the average time for each potential event and the standard deviation of all the times. The video analysis has more computational time than the computational analysis because it separates the video into many frames with computation on each one. Since this method performs both video and composite image analysis on some events and just the composite image analysis on others, the standard deviation is expected to be sizeable.

Table 4.4: The confusion matrix for the testing data for the Traditional Hough Detection Method.

<table>
<thead>
<tr>
<th>Predicted class</th>
<th>True class</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive</td>
<td>Negative</td>
</tr>
<tr>
<td>Positive</td>
<td>27</td>
<td>81</td>
</tr>
<tr>
<td>Negative</td>
<td>9</td>
<td>227</td>
</tr>
</tbody>
</table>

The results on the performance metrics for the training and testing sets are in Table 4.5. The accuracy and recall values are similar, suggesting that the training grid search consistently reduces false negatives. The precision and, therefore, the F-score are much lower than they are for the training set, partly because the testing set has
many more negatives. Of all the negatives, 61 % and 63 % are accurately identified in
the training set and testing set, respectively; the method correctly identifies less than
two-thirds of the negatives for both datasets. The precision is likely also affected by
the parameters in the grid search being too loose - using stricter parameters yields a
reduced number of false positives and increases precision, but it also increases false
negatives. Therefore, more stringent parameters would reduce the recall value, but
it is worth investigating in future work. The computational time is similar - the test
set has a slightly lower average time and a much lower standard deviation because
of the high proportion of negative events. Since all positive events go through both
video and composite image analysis while most negatives go through just the composite
image analysis, more negative events reduce the required computational time and the
standard deviation.

Table 4.5: The results from the Traditional Hough Detection Method

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (%)</td>
<td>76.72</td>
<td>73.84</td>
</tr>
<tr>
<td>Recall (%)</td>
<td>76.74</td>
<td>75.00</td>
</tr>
<tr>
<td>Precision (%)</td>
<td>66.00</td>
<td>25.00</td>
</tr>
<tr>
<td>F-score (%)</td>
<td>70.97</td>
<td>37.50</td>
</tr>
<tr>
<td>Time (mean) (s)</td>
<td>35.64</td>
<td>27.09</td>
</tr>
<tr>
<td>Time (standard deviation) (s)</td>
<td>89.36</td>
<td>43.77</td>
</tr>
</tbody>
</table>

The strengths of this method include the high recall percentage, which results from
fewer false negatives, all shown in Figure 4.3. Since all the false negatives among the
tested events have short and thin paths (a strong indicator for tiny meteors), any events
deemed as negative can be discarded as not containing a meteor. The February 25,
2021 event is the longest streak but still not a bright bolide and likely a tiny meteor.
Since the Traditional Hough Method correctly labeled all of the more luminous events,
it achieved an essential aim of this paper. The high recall percentage is a strength that
shows the success of the grid search on the parameters, while the low precision means
more events require human classification to sort through.

Figure 4.3: These are a selection of images of the composites of false negatives from testing the Traditional Hough Detection Method. It includes mostly short, with the largest meteor in the bottom right image.

The weakness of this method is the high number of false positives, which reduced
the precision, accuracy, and F-score percentages. The many false positives mean that the Traditional Hough Detection Method needs to be complemented by other methods to automate the detection process fully. Another computational tool must be used in unison, or a person must sort through the positives, where about half will be actual positive events. Another area for potential improvement of this method is the computational time on the order of seconds with a substantial standard deviation. It takes hours to run hundreds of videos, making it challenging to implement daily on camera recordings from multiple nodes. Additionally, the captured videos are of varied lengths, and longer videos take longer to analyze, making daily computational time unpredictable. Predictability is critical in automation to prevent endless troubleshooting, and variability is prone to errors.

4.3 Convolutional Neural Network

The CNN uses the training dataset to build a deep learning model and the testing set to verify that the parameters are not specific to the training set but applicable to any data. Therefore for the grid search, the training metrics are very high and say little about the model’s performance. The training accuracy was 97 % which is very high. Testing whether the CNN training over-fit or under-fit is difficult without running the testing set simultaneously is difficult. Additionally, it is challenging to carry out a precise grid search with so few images. The typical CNN code trains on tens of thousands of images, so training it on hundreds is a proof of concept that will improve with more labeled data. The CNN trains on composite images labeled “bolide” or “not bolide” - that somebody manually has to label (here, bolide label stands for meteors of all sizes). The grid search is over the scaling percent (amount the image’s dimensions are reduced) and the number of epochs. The goal of the grid search is to optimize the
performance metrics of the test set to prevent over-fitting.

The grid search results are different models that have been trained with other parameters and saved as .h5 files. The user can call different datasets to run on these models and return predictions based on the weights from the training. The best model has the highest F-score, so the method is balanced - 26 % scale of the original image and 75 epochs. The accuracy of all the models in Table 4.6 is over 91 %, the highest of all the methods this project implemented. The number of false positives is the lowest of all the other methods, but the number of false negatives is high. Unlike the Traditional Hough Detection Method, users have less control over adjusting parameters to optimize specific metrics because the mathematical complexity hides the feature and pattern selection. The disadvantage is that focusing on specifically maximizing the recall percentage is impossible, but the training is far more time efficient.

<table>
<thead>
<tr>
<th>Scale</th>
<th>Epochs</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>TP</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>300</td>
<td>406</td>
<td>14</td>
<td>27</td>
<td>37</td>
<td>91.53</td>
<td>57.81</td>
<td>72.55</td>
<td>64.35</td>
</tr>
<tr>
<td>22</td>
<td>100</td>
<td>407</td>
<td>13</td>
<td>27</td>
<td>37</td>
<td>91.74</td>
<td>57.81</td>
<td>74</td>
<td>64.91</td>
</tr>
<tr>
<td>26</td>
<td>75</td>
<td>405</td>
<td>15</td>
<td>26</td>
<td>38</td>
<td>91.53</td>
<td>59.38</td>
<td>71.7</td>
<td>64.96</td>
</tr>
<tr>
<td>21</td>
<td>50</td>
<td>406</td>
<td>14</td>
<td>27</td>
<td>37</td>
<td>91.53</td>
<td>57.81</td>
<td>72.55</td>
<td>64.35</td>
</tr>
</tbody>
</table>

The false negatives for the testing results of the CNN are in Figure 4.4, which includes large and small meteors. This result shows that rare and dim meteors are challenging to identify - there are few very large bolides in the training and testing datasets because they are rare. The total computational time for training the best model is 388.97s and testing is 33.23s for all the images. These computational times are orders of magnitude faster than traditional methods, so additional training is inexpensive and improves the procedure's outcome.

The CNN approach to finding meteors in the literature (Towner et al., 2020) has
Figure 4.4: Some false negatives from the test set of the best CNN model. These composite images include very short meteors and the large meteor in the top left corner.

an average recall of 86 % and 23 % average precision training on 24000 images. This shows that with more training data, the results could improve the recall and also that these results are more precise than the literature. Their training accuracy is 92 %, and there are many false positives, so the way the CNN got implemented for this study yields better metrics than other applications in some aspects. Towner et al. (2020) only includes a single layer, so this paper’s CNN is more complex. Towner et al. (2020) compares the neural network approach to a Hough-based method, and both methods require a lot of human interaction to check the results. Therefore this project’s results are similar to what Towner et al. (2020) reported.
4.4 YOLOv5

The benefit of applying YOLOv5 is that it is a pre-trained model that builds on the base of a CNN. However, unlike all the other methods, events are not classified as “meteor” or “no meteor”. YOLOv5 uses only the composite image, where each one has key elements labeled. These labels are beneficial since they add more details to the test data and context to a negative event (often a cloud, light or moon triggers the false positive). YOLOv5 is free software that Ultranalytics designed for ease of use. One line of batch script trains it, another line tests it, and the software comes with data visualizations shown in the figures below. The weakness of using YOLOv5 is that the labeling boxes within the image are time-consuming, and the user has to perform it by hand.

4.4.1 Training

In this section, the model trains with just the training set to optimize the parameters. It is helpful to see how the model is learning from the training data but has a high chance of over-fitting. The training time was 6 seconds per epoch with 300 epochs - the metrics stopped improving after this point. This training takes longer than the CNN, but that is expected since there are more layers and sampling occurs multiple times within the same composite image.

Figure 4.5 illustrates the qualities of the labels. This summary of the input data can explain the results because machine learning models are as good as their input data. The bar plot on the top left-hand corner shows the number of each label. Lights and the moon are present in a majority of composite images. The training set included more

---

1https://github.com/ultralytics/yolov5
Figure 4.5: Labels used in the training and testing set. The YOLOv5 software generates these plots to show patterns within the dataset. They show the number of labels per class and summarize the size.
than 50 composite images with planes, as they are a prevalent cause of false positives. The number of clouds is meager because they are difficult to label. Clouds are only visible at night if they reflect the light from the moon and often fill the entire frame. Herefore drawing a frame around the clouds that does not include the moon, bolide or lights and is not the size of most of the frame is a complex and time-consuming task. In hindsight, this cloud category probably should not have been included. The top right shows the size of all the boxes around objects where most frames are small rectangles and that the meteors and the moon (in red and pink) are smaller than the lights (in green). That is likely because the lights are on the ground and nearby while the meteor, planes and moon are visible higher in the sky. The bottom left panel shows the position in the image where the labels are, and that appears essentially random with slightly more labels around the horizon (where meteors, planes and lights are more likely to be seen). The bottom right plot is a scatter plot of the boxes’ sizes; most of them are small (less than 10 % of the size of the image), and generally, the heights are larger than the widths.

Figure 4.6 shows the confusion matrix for YOLOv5 on the training set. YOLOv5 is a multi-class method where the other methods are binary choice problems. The multiple classes change the concept of a false negative, so the confusion matrix differs. The plot shows that YOLOv5 correctly identified all meteors and clouds, 98 % of the moon and light sources, and 95 % of planes. Of all the false positives, YOLOv5 predicts 17 % as meteors, another 17 % as airplanes, and the rest as lights. These results show that these categories will likely be weaknesses in the test set and that the precision for those three labels will probably stay low. The plane’s class has the highest percentage of false negatives because its lights can be dim and irregular. Many different kinds of airplanes use the airport, all at different altitudes making the light look different between composite images.
Figure 4.6: YOLOv5 training confusion matrix. This training performed very well with true positives near 100 % for all the classes.

Table 4.7 shows the metrics of each label and all of them combined. The worst-performing F-score is for the bolide label because bolides vary much more in size and shape than the others in the composite image. The recall percentage was higher than the precision for most labels. Figure 4.7 shows the progress from epoch to epoch of the precision and recall. The value increases at a decreasing rate for both the precision and recall plots of the first hundred epochs. At around the 100 epoch mark, there is a significant shift in the results for both plots. This shift implies a large change in the model plots - perhaps a new feature or relationship was identified - which,
unfortunately, is difficult to locate. Then for the remaining epochs, the precision increases at a decreasing rate, and the recall levels out at 100 %.

4.4.2 Testing

The testing includes building the model with the testing set used to calculate the metrics. The model trains the same as before but optimizes on the condition of the test set to avoid over-fitting. The training time was 6 seconds per epoch for 200 epochs.

Figure 4.8 shows how the testing metrics evolve over the epochs Figure 4.7. Instead of steadily increasing with a slight noise, there is a significant variation between epochs, creating more oscillation than in Figure 4.7. The precision in Figure 4.8 keeps

<table>
<thead>
<tr>
<th>Category</th>
<th>Instances</th>
<th>Precision (%)</th>
<th>Recall (%)</th>
<th>F-score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>235</td>
<td>88</td>
<td>97.5</td>
<td>92.5</td>
</tr>
<tr>
<td>Bolide</td>
<td>30</td>
<td>77.9</td>
<td>93.9</td>
<td>85.2</td>
</tr>
<tr>
<td>Moon</td>
<td>84</td>
<td>96.2</td>
<td>98.8</td>
<td>97.5</td>
</tr>
<tr>
<td>Planes</td>
<td>57</td>
<td>98</td>
<td>94.7</td>
<td>96.3</td>
</tr>
<tr>
<td>Clouds</td>
<td>2</td>
<td>77.7</td>
<td>100</td>
<td>87.5</td>
</tr>
<tr>
<td>Lights</td>
<td>62</td>
<td>90.1</td>
<td>100</td>
<td>94.8</td>
</tr>
</tbody>
</table>
oscillating at a decreasing amplitude of around 50% while the recall increases at a
decreasing rate. Both metrics are lower for the test set than the training set, which is
typical. In Figure 4.9, it is easier to see the relationship between precision and recall
for each label. The goal is to have the plot in the top right corner to maximize recall
and precision. Therefore, light labels perform the best while bolides perform the worst.

![Graph showing precision and recall values over epochs for testing set.](image)

Figure 4.8: The precision and recall values over the epochs for the testing set. These plots
show that introducing a testing dataset adds more noise to the metrics with larger amplitudes
between epochs. However, the precision is not increasing while the recall is.

The confusion matrix for the test results for YOLOv5 in Figure 4.10 differs from
the training version in Figure 4.6. Firstly there are no cloud events in the test dataset.
The best-performing label is the lights. Planes and bolides are often confused with each
other. In composite images, this is possible because the lines can look similar. False
positives are primarily bolides and lights. YOLOv5 falsely labeled the background
more on the test than the training set.

It took 2 seconds to test 95 images in the best - the expected time for any group
of images that size. The breakdown of that computational time is 0.3ms pre-process,
9.1ms inference, and 4.5ms NMS per image on average. The performance metrics in
Table 4.8 show that the recall is lower for the testing than the training set. However,
the precision drops steeply, oddly similar to that in the Traditional Hough Detection
Figure 4.9: Precision vs. Recall over the epochs for each label. The goal is high precision and recall, so the bolide label is the least successful while the lights label is the most.

Method. The bolide label performs the worst on all three metrics, similar to the training set.

Table 4.8: The results from the testing set on the best model.

<table>
<thead>
<tr>
<th>Category</th>
<th>Instances</th>
<th>Precision (%)</th>
<th>Recall (%)</th>
<th>F-score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>103</td>
<td>43.9</td>
<td>71.9</td>
<td>54.5</td>
</tr>
<tr>
<td>Bolide</td>
<td>22</td>
<td>36.6</td>
<td>57.7</td>
<td>44.8</td>
</tr>
<tr>
<td>Moon</td>
<td>31</td>
<td>39.5</td>
<td>64.5</td>
<td>49.0</td>
</tr>
<tr>
<td>Planes</td>
<td>17</td>
<td>39.9</td>
<td>74.3</td>
<td>51.9</td>
</tr>
<tr>
<td>Lights</td>
<td>33</td>
<td>59.5</td>
<td>90.9</td>
<td>71.9</td>
</tr>
</tbody>
</table>

Figure 4.11 compares the labels to the predictions in the test set. YOLOv5 identifies all the meteors in the figure except the large one. The model falsely finds bolides in the background while correctly identifying lights with various probabilities. These probabilities help to add insight into the identifying labels. The image in the top left corner has the bolide path specified as a bolide and a plane with a different probability.
Figure 4.10: YOLOv5 testing confusion matrix. These plots show the lights and the moon have a high percentage of true positives.
Figure 4.11: This shows the difference between the labeled test images and their results. The lights label performs well, while there are bolides misidentified most often by YOLOv5.
The gap between the training and test data results implies that more training data with more diverse events will likely improve the results. The large YOLO models perform better in Wang et al. (2021), so running YOLOv5 medium or large instead of small could improve the results. To run bigger models, more GPUs with better computational power are necessary. Wang et al. (2023) has an 88% precision for classifying galaxies - more training data could achieve this result.
Chapter 5

Conclusions and Future Work

The chapter ties together the results with the project aims to compare the automated detection methods. Then the whole paper is summarized, explaining the strategy to automate the detection of meteors best. More work in this field could redesign or retrain current methods or create new ones to improve the results.

5.1 Compare Performance

To achieve full automation, the best or combination of methods must optimize all the metrics. Using any of the four methods alone could partially automate bolide detection with their current performance. All automated methods have multiple processes and steps (Trigo-Rodriguez et al., 2008; Towner et al., 2020; Peña-Asensio et al., 2021). A final fully automated code must run all the events on numerous techniques and decide if it is a meteor on a balance of probabilities.

Figure 5.1 compares the metrics between the methods. The Optimized Botella method (the benchmark) had its recall, and F-score improved significantly for all the new techniques introduced. The precision for Optimized Botella is the second largest...
of all the methods, which shows that brightness tests may help reduce false positives. Since false negatives impede automation more than false positives, all the new techniques outperformed the benchmark Optimized Botella. The method that performs best on all three performance metrics is the CNN, which is very interesting because the structure of the method is the least tailored to this problem and the most generic algorithm setup. However, the size of the training set is the largest since it uses composite images only, so it is the quickest to identify and the fastest to test (unlike videos). The Traditional Hough Detection Method has the highest recall percentage, which shows that image processing techniques make it easier to control the number of false negatives. Combining the Traditional Hough Method and the CNN could minimize false negatives while reducing false positives.

Test results compared

![Bar chart showing test results for YOLOv5, CNN, Trad. Hough, and Opt. Botella.]

Figure 5.1: Metrics for each method. All three new methods improved the recall and F-score.

Figure 5.2 shows the computational time used for each method. The low F-score and the high computational time for Optimized Botella show that the new techniques have considerably improved since 2018. Traditional Hough Method uses a third of the time Optimized Botella uses, making it more feasible to test thousands of events per
day (1000 videos would take 8 hours for Optimized Botella). Additionally, most dark
nights with no positive events will run faster than average because the Traditional
Hough Method only analyzes the composite image without looping over video frames.
The machine learning methods are 1000 times faster than the traditional methods (1000
videos would take less than 2 minutes). This improvement is because the methods use
composite images only, and most of the computational work happens in the training
stage. The machine learning methods are worth focusing on from the computational
time because the time spent labeling and training translates to better results without
reducing feasibility. Also, a person takes seconds to identify a meteor in an image or
video, so only machine learning methods are faster than a person every time. Using
more powerful GPUs would also speed up the analysis.

![Computational time per image](image)

Figure 5.2: The computational time shows with methods is most efficient. YOLOv5 is the
fastest, while all new methods outperform Optimized Botella

Towner et al. (2020) has a similar approach with a Hough Detection method com-
pared to a neural network output, and the comparison is difficult because the metrics
they selected were different. The Hough performance has a high recall, while the pre-
cision for the traditional method is lower than the results in this project. Towner et al. (2020) has a Hough method that breaks up the composite image into tiles and does not use videos of the event. Their final product has human supervision to sort out the false positives - similar to this project.

No method works 100% on all events, and all methods in the literature include human interaction to sift through false positives. Artificial intelligence is booming in industry and academia, and object identification plays a large part. Deep learning techniques keep improving, and an “off-the-shelf” method like YOLOv5 could make this process fully automated in the future. In the short term, a compound method trained and optimized regularly is the easiest way to reduce the human workload. The Traditional Hough Method reduces the events needing classification by humans to 31% of the total without missing meteors. Machine learning methods can check thousands of images in minutes.

5.2 Future Work

Future work might reparameterize, reorganize or retrain the Traditional Hough Detection Method to increase precision. Incorporating other computer vision techniques and tests from the literature could improve the results. The challenge in combining traditional techniques is balancing the parameters and ordering the steps. Method design is very time-consuming, but future work could investigate a new design.

Future work could focus on increasing the labeled datasets for the CNN and YOLOv5 to improve training. Labeling is the most time-consuming part since the hardware generates events that could be meteors every night. Developing a machine learning method that uses video data would increase many more images and may increase the labeling speed. Labeling would be more straightforward because each video has hundreds
of frames with the same unmoving elements, like the moon or lights on the horizon. Machine learning methods improve with more high-quality labeled data.

Hardware improvements could improve computational time and allow for more complex methods. If the procedures are run on a supercomputer, in parallel, with many CPUs and GPUs, the quality of the results will increase.

The scope of the automated method was also vast - any meteor visible within the 180° visible to the human eye. Future work could look into a more limited meteor brightness within a narrow-angle in the frame. A more specialized problem with a specialized method could improve the results.

Future work could investigate some untested methods in this project, such as using unique encodings of images, including Principal Component Analysis, Graph encoding and different autoencoders as input for Machine Learning techniques. Recurrent Neural Networks, like Long Short-Term Memory networks, could categorize the video data well for time-based data. Deep Learning is a growing and dynamic field, so future work could develop or adapt new designs to solve this problem.

A complete validation study of the different combinations of the methods tested could create a fully automated or primarily automated process. The comparison from this paper makes the weaknesses and strengths clear so that a compound method is easier to build.

5.3 Conclusion

This project tested four methods on performance metrics, and none performed well enough to run fully automated. However, each method has different strengths that, when used together, can produce a fully automated process. The first chapter explains this project’s aims, the physics of meteors and bolides, and how automated methods
have identified them in the literature. Chapter 2 uses metrics, flowcharts, and code segments to explain all four tested methods. Chapter 4 shows the performance of the methods on the metrics described.

Combining the Traditional Hough Detection Method, the CNN, and YOLOv5, an automation method could efficiently identify meteors using videos and images. Running the CNN first on all the composite images results in the highest precision and F-score. To ensure that the combined automation method misses nothing, it helps run the Traditional Hough Metric on all the videos and composite images from events classified as “not a meteor” by the CNN. Then YOLOv5 runs on all the events and uses the labels and probabilities to confirm the contents of images, including lights, the moon, or bolides. Using the position data (used to find the speed) from the Traditional Hough Method and the position of label boxes in YOLOv5 allows a combined automation method to check if the bright event is a meteor. This combined automation method would be faster than running the Traditional Hough Detection method on all the images because the machine learning methods are so quick.

Automation is a powerful tool to replace labor that is tedious, time-consuming, and easily inaccurate when humans perform the analysis. However, image recognition, in particular, is notoriously difficult without massive datasets and high-powered computing. As these barriers decrease, these results will likely continue to improve until the entire meteor identification project is automated.
Bibliography


Appendix A

Code

Each method was tested and trained. Chapter 2 describes the training and the grid search, which is excluded in the code below for conciseness. The code shown would sufficiently recreate the results in Chapter 4.

Table A.1 shows the versions of the Python libraries used so the conda environment can be recreated.

<table>
<thead>
<tr>
<th>name</th>
<th>version</th>
<th>citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python 3</td>
<td>3.8.10</td>
<td>Van Rossum and Drake (2009)</td>
</tr>
<tr>
<td>Matplotlib</td>
<td>3.4.2</td>
<td>Hunter (2007a)</td>
</tr>
<tr>
<td>NumPy</td>
<td>1.21.4</td>
<td>Harris et al. (2020)</td>
</tr>
<tr>
<td>SciPy</td>
<td>1.7.1</td>
<td>Virtanen et al. (2020)</td>
</tr>
<tr>
<td>PyAV</td>
<td>9.2.0</td>
<td>Boers (2020)</td>
</tr>
<tr>
<td>OpenCV</td>
<td>4.5.5.64</td>
<td>Bradski (2000)</td>
</tr>
<tr>
<td>Pillow</td>
<td>9.3.0</td>
<td>Clark (2015)</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>2.9.1</td>
<td>Abadi et al. (2015)</td>
</tr>
<tr>
<td>Pandas</td>
<td>1.4.3</td>
<td>McKinney et al. (2010)</td>
</tr>
<tr>
<td>Scikit</td>
<td>0.24.2</td>
<td>Pedregosa et al. (2011)</td>
</tr>
<tr>
<td>Keras</td>
<td>2.9.0</td>
<td>Team (2020)</td>
</tr>
</tbody>
</table>
A.1 Optimized Botella Automated Detection

This code is very similar to its original in Botella (2018) in an attempt to keep it as a benchmark. Section A.1 discusses the differences between the Automated Bolide Detection Method and Optimized Botella.

```python
import matplotlib
matplotlib.use('Agg')
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt
import av
import os, time
import csv
import warnings
import matplotlib.cbook
import glob

def original_Elena(filename, category, outfile, infile):
    # initialization
    start_time = time.time()
    warnings.filterwarnings("ignore", category=matplotlib.cbook.mplDeprecation)

    nfiles = []
    thres = 0.3  # for the first function, min is always 0.25 or so
    thres1 = 0.7  # for most of the second function
    maxthres = 0.26  # for the maxval plot in com
    filedir = outfile + "/
    location = infile + "/" + category + "/
    capsdir = filedir + "/caps/"
    scaps = filedir+'/caps/'
    nfiles = []
```
events = []
startbigfor = time.time()
stfr = 0
outfolder = filedir+'/out/'
plotfolder = filedir+ '/plots/'
data = []
n_f = []
maxval = []
sump = []
sump1 = []
sump2 = []

# Video preprocessing, file management and related variables
startvidproc = time.time()
vid = av.open(location + '.mp4')
if not os.path.isdir('/'+ filename + '/'):
    os.makedirs(filedir, mode=0o777, exist_ok=True)
    os.makedirs(filedir + '/caps/', mode=0o777, exist_ok=True)
    for f in vid.decode(video=0):
        nfiles.append(f)
        f.to_image().save(capsdir + '/%04d.png'%f.index)
snapnum = len(nfiles) # number of snapshot images
xarr = np.zeros(snapnum)
yarr = np.zeros(snapnum)
f = np.array(range(snapnum))
vidproctime = time.time()- startvidproc
print(filename, "video processing complete")

# Loop over frames
endfr = len(glob.glob(capsdir + '/*.png'))-2
newfi = np.arange(stfr,endfr,step=1)
x_arr = np.zeros_like(newfi, dtype=np.float64)
y_arr = np.zeros_like(newfi, dtype=np.float64)
background = plt.imread(scaps +"/%04d.png"%(stfr))
background = maskimg(background)
for n, n_file in enumerate(newfi):
    # ----------- PIXEL ANALYSIS -----------
    image = plt.imread(scaps +"/%04d.png"%(n_file))
    n_f.append(n_file)
    image = maskimg(image)

    if n > 0:
        flat = plt.imread(scaps +"/%04d.png"%(n_file - 1))
        sum_p = np.sum(image)
        sump.append(sum_p)
        sump_1 = np.sum(image[np.where(image > thres)])
        sump1.append(sump_1)
    else:
        flat = background
        sump.append(np.sum(image))
        sump_1 = np.sum(image)
        sump1.append(np.sum(image))
    img = image - flat
    sump_2 = np.sum(img[np.where(img > thres)]) # 0.2 to 0.4
    if (sump_2 > 6.):
        print("Possible event detected: Frame", (n+stfr-1),
        "at time", ((n+stfr-1)/1800.), "min.", sump_2)
        events.append((n_file, sump_2))
    else:
        pass
    sump2.append(sump_2)
print("Pixel analysis complete", n)
```python
# --------- COORDINATE ANALYSIS -----------

flat = plt.imread(scaps + '/0000.png')
flat = maskimg(flat)

img = image - flat

img2 = (img[:,:,0] + img[:,:,1] + img[:,:,2])/3. # rgb avg

img2 = maskimg(img2)
max_val = np.max(img2)
maxval.append(max_val)

if max_val < (thres1):
    # coord arrays w max
    maxx, maxy = np.unravel_index(np.argmax(img2), img2.shape)
    minv = 5
else:
    big = (img2 > thres1).sum()
    ind = (img2).argpartition(big, axis=None)[:big]
    xx, yy = np.unravel_index(ind, img2.shape) #coord arrays
    maxx = int(np.average(xx))
    maxy = int(np.average(yy))
    if(big > 10):
        minv = 25
    else:
        minv = 10
    minv = np.min([np.min([maxx, img2.shape[1]-maxx]),
                   np.min([maxy, img2.shape[0]-maxy])])

if(max_val > maxthres):
    x, y = ndimage.center_of_mass(img2[(maxx-minv):(maxx+minv),
                                         (maxy-minv):(maxy+minv)])
else:
    continue
```

87
x = x + maxx - minv
y = y + maxy - minv

if (x != x_arr[n-1]) or (y != y_arr[n-1]):
    x_arr[n] = x
    y_arr[n] = y
else:
    x = np.nan
    y = np.nan
    x_arr[n] = x
    y_arr[n] = y

print("Coordinate analysis complete", n)

print("max frame: %d ; max sum/peak: %0.3f"%(np.argmax(sump2), np.max(sump2)))

numevents = len(events)
print("Number of pixel analysis events detected: %d"%numevents)

fintime = time.time()-start_time
bigfortime = time.time()-startbigfor
print(filename, "complete")

#Delete files
for f in range(len(nfiles)):
    try:
        os.remove(capsdir + "/%04d.png"%f)
    except:
        pass
os.rmdir(filedir + "/caps/")
os.rmdir(filedir)
# Evaluate coord analysis

count = 0
n = len(x_arr)
x = []
y = []
traj = False
for i in range(n):
    if x_arr[i] + y_arr[i] > 0:
        count += 1
        x.append(x_arr[i])
        y.append(y_arr[i])
    else:
        count = 0
        x = []
        y = []
        if count == 3:
                traj = True
        return np.round(fintime,2), np.round(vidproctime,2),
        np.round(bigfortime,2), numevents>0, traj

A.2 Traditional Hough Detection Method

The preprocessing was inspired by the Botella Method (Botella, 2018) and the Hough code comes from the OpenCV documentation (Bradski, 2000).

```python
import numpy as np
import av, cv2
import matplotlib.pyplot as plt
```
from scipy.ndimage import gaussian_filter, median_filter
import glob, time, os
from PIL import Image

def Trad_Hough(filename, category, minlinelength, vid_thresh, outfile, infile):
    
    """
    Input:
    filename (str)
    category (str)
    minlinelength (int [2, 2000]) - length of lines in composite
    vid_thresh (int [2, 70]) - threshold for dark videos
    outfile (str) - location for results
    infile (str) - location for data
    
    Output:
    runtimes
    binary true or false bolide event prediction
    """
    #initialisation of variables
    filedir = outfile + "/" + filename
    location = infile + "/" + category + "/" + filename
    capsdir = filedir + "/caps/
    nfiles = []
    start = time.time()
    print(filename, category, 'initialization complete')
    #Find the background
    def average_img_3(imlist):
        images = np.array([np.array(Image.open(f)) for f in imlist])
        arr = np.array(np.mean(images, axis=(0)), dtype=np.uint8)
        return Image.fromarray(arr)
vid = av.open(location + ".mp4")
if not os.path.isdir("/" + filename + "/"):  
    os.makedirs(filedir, mode=0o777, exist_ok=True)  
    os.makedirs(capsdir, mode=0o777, exist_ok=True)
background_locations = np.array([])  
for i in range(5):
    f = next(vid.decode(video=0))  
    f.to_image().save(capsdir + "/0" + str(i+1) + ".png")  
    background_locations = np.append(background_locations, capsdir + "/0" + str(i+1) + ".png")

imlist=[filename for filename in background_locations]
average_img_3(imlist).save(capsdir + "/background.png")
for m in glob.glob(capsdir + "0*"):
    os.remove(m)
print(filename, category, 'background built')

#________________COMPOSITE IMAGE________________
#Remove the background
try:
    composite = plt.imread(location + ".jpg")
except:
    composite = plt.imread(location + "m.jpg")
background = (plt.imread(capsdir + "/background.png")*255).astype(np.uint8)
final = abs(composite - background)
#Pre-process
grayImage = cv2.cvtColor(final, cv2.COLOR_BGR2GRAY)
med = median_filter(grayImage,3)
filtered = gaussian_filter(med, 1)
edges = cv2.Canny(image=filtered, threshold1=15, threshold2=200)
print(filename, "composite built, Hough transform began")
# Hough transform
lines_list_x = []
lines_list_y = []
lengths = []
lines = cv2.HoughLinesP(edges,  # Input edge image
                        1,  # Distance resolution in pixels
                        np.pi/180,  # Angle resolution in radians
                        threshold=25,  # Min number of votes for valid line
                        minLineLength=2,  # Min allowed length of line
                        maxLineGap=10  # Max allowed gap between line for
                                 # joining them
                        )

try:
    for points in lines:
        x1, y1, x2, y2 = points[0]
        if y1 != y2 and x1 != x2:
            lines_list_x.append([x1, x2])
            lines_list_y.append([y1, y2])
            length = np.sqrt((x1 - x2)**2 + (y1 - y2)**2)
            lengths.append(length)
except:
    lengths = []

print(filename, ' Hough lines detected. Line groupings begin.', str(len(lengths)), ' lines detected')

# Group the Hough lines
n = len(lengths)
cosine_angle = 0.99
group = 0
groupings = {0:[0]}
final_lines_x = []
final_lines_y = []
`final_lengths = []`

`if n > 1:`

`# build groups of lines`

`for i in range(n-1):`

`# distance between the lines centres`

`if lengths[i] > lengths[i+1]:`

`max_length = lengths[i]`

`else:`

`max_length = lengths[i+1]`

`center_x_0 = (x1_0 + x2_0) / 2`

`center_y_0 = (y1_0 + y2_0) / 2`

`center_x_1 = (x1_1 + x2_1) / 2`

`center_y_1 = (y1_1 + y2_1) / 2`

`distance_between_centers = np.sqrt((center_x_0 - center_x_1)**2 + (center_y_0 - center_y_1)**2)`

`if distance_between_centers > 0.5 * max_length:`

`group += 1`

`# angle between lines`

`vector_0 = np.array([x1_0, y1_0]) - np.array([x2_0, y2_0])`

`vector_1 = np.array([x1_1, y1_1]) - np.array([x2_1, y2_1])`

`dot = np.dot(vector_0 / np.linalg.norm(vector_0), vector_1 / np.linalg.norm(vector_1))`

`if dot < cosine_angle:`

`group += 1`

`try:`

`temp = groupings[group]`

`temp.append(i+1)`

`93`
```python
    groupings[group] = temp
    except:
        groupings[group] = [i+1]

    #build final lists from the groups
    for j in list(groupings.keys()):
        lines_list_x = np.array(lines_list_x)
        lines_list_y = np.array(lines_list_y)
        temp_list_x = lines_list_x[np.array(groupings[j])]
        temp_list_y = lines_list_y[np.array(groupings[j])]
        #find max x1,x2,y1,y2
        final_lines_x.append([np.min(temp_list_x),np.max(temp_list_x)])
        final_lines_y.append([np.min(temp_list_y),np.max(temp_list_y)])
        final_length = np.sqrt((np.min(temp_list_x)-np.max(temp_list_x))**2+(np.min(temp_list_y)-np.max(temp_list_y))**2)
        final_lengths.append(final_length)
    elif n==1:
        final_lines_x = lines_list_x
        final_lines_y = lines_list_y
        final_length = np.sqrt((lines_list_x[0][0]-lines_list_x[0][1])**2+(lines_list_y[0][0]-lines_list_y[0][1])**2)
        final_lengths.append(final_length)
    elif n==0:
        final_lengths = [0]
        print("final length", np.max(final_lengths))

houghstart = time.time()
houghtime = houghstart-start

#________________VIDEO ANALYSIS________________
if np.max(final_lengths)>minlinelength:
```
# Video preprocessing, file management and related variables

```python
vid = av.open(location + "\.mp4")
for f in vid.decode(video=0):
    nfiles.append(f)
    f.to_image().save(capsdir + "/%04d.png"%f.index)

snaps = np.sort(glob.glob(capsdir + "*.png"))  # number of snapshot images (NOTE ISSUES USING nfiles)
 nidproctime = np.round(time.time()-houghstart, 2)
print("Video processing complete. Runtime:", vidproctime, "sec")
```

# Looping over frames

```python
speeds = np.array([])
background = plt.imread(capsdir + "/background.png")
forstart = time.time()
positions = [[0,0]]
for n in snaps:
    # load image
    image = plt.imread(n)
    x_old, y_old = positions[-1]
    x, y = positions[-1]
    # pre-process
    contrastimg = abs(image - background)
    graycontrastimg = (cv2.cvtColor(contrastimg, cv2.COLOR_BGR2GRAY)*255).astype(np.uint16)
    blur = cv2.GaussianBlur(graycontrastimg,(5,5),0)
    ret_otsu,_ = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    circle_flag = False
    # separate bright vs dark
    if ret_otsu > 3: # bright image with lots going on
        # preprocess
```

95
_, thresh = cv2.threshold(blur, ret_otsu, 255, cv2.THRESH_BINARY)
final_im = cv2.normalize(src=thresh, dst=None, alpha=0,
                        beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)

# Hough circles
detected_circles = cv2.HoughCircles(final_im,  # Gray image
                                     cv2.HOUGH_GRADIENT,  # Hough Method
                                     1,  # Accumulator resolution
                                     20,  # Minimum distance between circles
                                     param1=1,  # Stage 1 for the edge detector
                                     param2=10,  # Stage 2 for the accumulator
                                     minRadius=1,
                                     maxRadius=70)

# find circles
if detected_circles is not None:
    for circle in detected_circles[0]:
        x, y = np.round(circle[:2]).astype(int)
        if ((y < np.shape(final_im)[0]) and
             (x < np.shape(final_im)[1]) and (final_im[y, x] == 255)):
            positions.append([x, y])
            circle_flag = True
if circle_flag == False:  # too small for hough to work
    bright_y, bright_x = np.unravel_index(graycontrastimg.argmax(),
                                           graycontrastimg.shape)
    _, thresh = cv2.threshold(blur, vid_thresh, 255, cv2.
                               THRESH_BINARY)
    bright_4_by_4 = np.sum(thresh[bright_y-2:bright_y+2,
                               bright_x-2:bright_x+2])/255 > 10
    dark_border = (np.sum(thresh[bright_y-22:bright_y+22,
                               bright_x-22:bright_x+22]) - np.sum(
                        thresh[bright_y-20:bright_y+20,
                               bright_x-20:bright_x+20]))/255 < 10
if (bright_4_by_4 and dark_border):
    x,y = [bright_x, bright_y]
else:
    x,y = [0,0]
positions.append([x,y])

#speed calc
speed = np.sqrt((x-x_old)**2+(y-y_old)**2)
if 2.5<speed<15:  #from gridsearch
    speeds = np.append(speeds,speed)
viddelstart = time.time()

#Delete snapshots
for f in glob.glob(capsdir + '*'):
    os.remove(f)
os.rmdir(filedir + '/caps/)
os.rmdir(filedir)
print("Video result length:", len(speeds)>1)
return np.round(houghtime,2), np.round(vidproctime+(time.time() - viddelstart),2), np.round(viddelstart-houghstart,2),
len(speeds)>1
else:
    viddelstart = time.time()
for m in glob.glob(capsdir + '*'):
    os.remove(m)
os.rmdir(filedir + '/caps/)
os.rmdir(filedir)
return np.round(houghtime,2), time.time()-viddelstart, 0,
np.max(final_lengths)>minlinelength
A.3 Convolutional Neural Network

This code both tests and trains the CNN in the same script. This is efficient because of the speed of training this method and that both training and testing are automated. This is based on Pedersen (2020).

```python
# Initialization
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import math
import pandas as pd
import time, datetime
from sklearn.metrics import confusion_matrix
from datetime import timedelta
import glob
import cv2

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, Input
from tensorflow.keras.layers import Reshape, MaxPooling2D
from tensorflow.keras.layers import Conv2D, Dense, Flatten
from tensorflow import keras
import keras.applications.resnet
from tensorflow.keras.optimizers import Adam
from keras.models import load_model

# LOAD AND PRESENT IMAGES

def load_images(location, scale_percent):
    
    """
    From the file location, find the images, turn gray, RESIZE and classify
    """
```
Inputs:
location (str): path to the files
scale percent (int): percent of original size

Outputs:
images (numpy array): array of gray AND RESIZED images
classifications (numpy array): array of strings
bolide (numpy array): array of bools
names (numpy array): array of str

#Find the categories
locationindex = len(location)
categories = np.array([])
for name in glob.glob(location + '*'):
    if name[-6:] != ' images':
        if name[locationindex:] != ' unclassified':
            categories = np.append(categories, name[locationindex:])

#rescale variables
image = plt.imread(glob.glob(location + categories[0] + '/*.jpg')[0])
width = int(image.shape[1] * scale_percent / 100)
height = int(image.shape[0] * scale_percent / 100)
dim = (width, height)

#initiate variables
dictoffilenames = {}
images = [np.zeros(dim[::-1])]
classifications = np.array([])
bolide = np.array([])
names = np.array([])
#loop over files

```python
for category in categories:
    index = locationindex + len(category)
    for name in glob.glob(location + category + '/*g'):
        # jpg and png
        try:
            dictoffilenames[name[index+1:-4]] = category
            image = plt.imread(name)
            # turn gray
            grayImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            # resize
            resized = [cv2.resize(grayImage, dim, interpolation = cv2.INTER_AREA)]
            images = np.concatenate((images, resized), axis=0)
            # categorise
            classifications = np.append(classifications, category)
            if category=='Bolides':
                bolide = np.append(bolide, True)
            else:
                bolide = np.append(bolide, False)
            # name
            names = np.append(names, name)
        except:
            continue
    print(category + " is resized.")
```

return images[1:], classifications, bolide, names

```python
def plot_images(images, name, cls_true, img_shape, cls_pred=None):
    """
    This saves a jpg with an image plot so the plots are standardized
    """
```
Input:
images
name
cls_true
cls_pred
img_shape

Output:
nothing returned
images saved

```
assert len(images) == len(cls_true) == 9
# Create figure with 3x3 sub-plots.
fig, axes = plt.subplots(3, 3)
fig.subplots_adjust(hspace=0.3, wspace=0.3)
for i, ax in enumerate(axes.flat):
    # Plot image.
    ax.imshow(images[i].reshape(img_shape), cmap='binary')
    # Show true and predicted classes.
    if cls_pred is None:
        xlabel = "True: {0}".format(cls_true[i])
    else:
        xlabel = "True: {0}, Pred: {1}".format(cls_true[i], cls_pred[i])

    # Clean up axes
    ax.set_xlabel(xlabel)
    ax.set_xticks([])
    ax.set_yticks([])
plt.show()
plt.savefig(name + '_image.jpg')
```
def plot_example_errors(cls_pred, name, test_names, test_cls, 
                        img_shape):
    
    """
The errors to understand what is being missed out on

Input:
cls_pred (numpy array) - predicted class-number for test images
name
test_names
test_cls
img_shape

Output:
nothing returned
images saved and the names saved in a csv
"""
# Boolean array whether the predicted class is incorrect.
incorrect = (cls_pred != test_y)
# Incorrectly classified images
images = test_X[incorrect]
# Get the predicted classes for those images.
cls_pred = cls_pred[incorrect]
# Get the true classes for those images.
cls_true = test_y[incorrect]
# Plot the first 9 images.
plot_images(images=images[0:9], name = name, 
            cls_true=cls_true[0:9], img_shape=img_shape, 
            cls_pred=cls_pred[0:9])
# Save error names
names = test_names[incorrect]
cls = test_cls[incorrect]
d = {"Name": names, "Category": cls}
df = pd.DataFrame(d)
df.to_csv(name + "_errors.csv")

def print_confusion_matrix(cls_pred, name, num_classes):
    
    The confusion matrix shows the difference between the true classifications and predicted ones

    Input:
    cls_pred (numpy array)
    name (str)
    num_classes (int)

    Output:
    nothing returned
    a csv and image output
    
    # Get the true classifications for the test-set.
    cls_true = test_y
    # Get the confusion matrix using sklearn.
    cm = confusion_matrix(y_true=cls_true, y_pred=cls_pred)
    # Print the confusion matrix as text.
    print(cm)
    # Plot the confusion matrix as an image.
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    # Make various adjustments to the plot.
    plt.tight_layout()
    plt.colorbar()
    tick_marks = np.arange(num_classes)
    plt.xticks(tick_marks, range(num_classes))
plt.yticks(tick_marks, range(num_classes))
plt.xlabel('Predicted')
plt.ylabel('True')
# Plot and save
plt.show()
plt.savefig(name + '_confusion.jpg')
dfcm = pd.DataFrame(cm)
dfcm.to_csv(name + '_confusion.csv')

def model_build_run_save(scale_percent, epoch_num, train_x, test_x, train_cls, train_y, test_cls, test_y):
    
    """
    This is the heart of code where all the magic happens.
    
    Input:
    scale_percent (int)
    epoch_num (int)
    destination (int)
    train_x (numpy array)
    test_x (numpy array)
    train_cls (numpy array)
    train_y (numpy array)
    test_cls (numpy array)
    test_y (numpy array)

    Output:
    nothing is returned
    keras model, confusion matrix csv and images of errors print out
    """
    
    #________________DATA FORMATTING_____________________________
    img_size_flat = train_x.shape[1]*train_x.shape[2] # 1D length
img_shape_full = (train_x.shape[1], train_x.shape[2], 1)

img_shape = (train_x.shape[1], train_x.shape[2])

num_classes = len(np.unique(test_y))  # Number of classes

train_X = train_x.ravel().reshape((train_x.shape[0], img_size_flat))

test_X = test_x.ravel().reshape((test_x.shape[0], img_size_flat))

# One-Hot-Encode the classes

one_hot_train_y = np.zeros((train_X.shape[0], 2))

for i in range(train_X.shape[0]):
    one_hot_train_y[i, round(train_y[i])] = 1

one_hot_test_y = np.zeros((test_X.shape[0], 2))

for i in range(test_X.shape[0]):
    one_hot_test_y[i, round(test_y[i])] = 1

# ____________________MODEL BUILDING____________________________

model = Sequential()  # Start with the Keras Sequential model

model.add(InputLayer(input_shape=(img_size_flat,)))

model.add(Reshape(img_shape_full))

# First convolutional layer with ReLU-activation and max-pooling

model.add(Conv2D(kernel_size=5, strides=1, filters=16, padding="same", activation="relu", name="layer_conv1"))

model.add(MaxPooling2D(pool_size=2, strides=2))

# Second convolutional layer with ReLU-activation and max-pooling

model.add(Conv2D(kernel_size=5, strides=1, filters=36, padding="same", activation="relu", name="layer_conv2"))

model.add(MaxPooling2D(pool_size=2, strides=2))

# Flatten the 4-rank output to 2-rank that can be input

model.add(Flatten())

# First fully-connected / dense layer with ReLU-activation

model.add(Dense(128, activation="relu"))

# Last fully-connected layer with softmax for classification.

model.add(Dense(num_classes, activation="softmax"))
# MODEL RUNNING

optimizer = Adam(lr=1e-3)
model.compile(optimizer=optimizer, loss="categorical_crossentropy",
metrics=['accuracy'])
model.fit(x=train_X, y=one_hot_train_y, epochs=epoch_num,
batch_size=128)
result = model.evaluate(x=train_X, y=one_hot_train_y)
for name, value in zip(model.metrics_names, result):
    print(name, value)
print("{0}: {1:.2%}".format(model.metrics_names[1], result[1]))
y_pred = model.predict(x=test_X)
cls_pred = np.argmax(y_pred, axis=1)

# MODEL SAVING

now = datetime.datetime.now()
own_str = now.strftime("%Y-%m-%dT%H:%M:%S")
path_model = destination + " model_ " + now_str + "_"+
str(scale_percent) + "_" + str(epoch_num) + ".h5"
model.save(path_model)
plot_example_errors(cls_pred, destination + " example_errors_ "+
now_str + "_" + str(scale_percent) + "_" + str(epoch_num),
test_names, test_cls, img_shape)
print_confusion_matrix(cls_pred, destination + "_" + now_str + "_"
+ str(scale_percent) + "_" + str(epoch_num), num_classes)
print("Model with", str(scale_percent), "scale percentage and
with", str(epoch_num), "epochs saved!!")
model_build_run_save(scale_percent, epoch_num, train_X, test_X,
train_cls, train_y, test_cls, test_y)
A.4 YOLOv5

There is lots of code to make it easier for the user to label the data. Additionally some code was built to automate the rescale of the frames and duplicate and label the corresponding labels. However all the user need to run yolov5 is a line of bash script:

```
$ python3 train.py --img 640 --batch 16 --epochs 300 --data bolideframes.yaml --weights yolov5s.pt
```