Smoke Plume Image Segmentation

Project report submitted in partial fulfillment
of the requirements for the degree of

Bachelor of Technology
in
Computer Science and Engineering

by

Mayank Saini - 20UCS114
Mitanshi Garg - 20UCS120
Nipun - 20UCS131

Under Guidance of
Dr. Aloke Datta

Department of Computer Science and Engineering
The LNM Institute of Information Technology, Jaipur

November 2023

The LNM Institute of Information Technology
Jaipur, India

CERTIFICATE

This is to certify that the project entitled “Smoke Plume Segmentation” , submitted by Mayank Saini (20UCS114), Mitanshi Garg (20UCS120) and Nipun (20UCS131) in partial fulfillment of the requirement of degree in Bachelor of Technology (B. Tech), is a bonafide record of work carried out by them at the Department of Computer Science Engineering, The LNM Institute of Information Technology, Jaipur, (Rajasthan) India, during the academic session 2023-2024 under my supervision and guidance and the same has not been submitted elsewhere for award of any other degree. In my/our opinion, this report is of standard required for the award of the degree of Bachelor of Technology (B. Tech).

20 November 2023
Date

Adviser: Dr. Aloke Datta

Acknowledgments

We would like to take this opportunity to express our gratitude to all those who have helped us completing our BTP.

We are deeply thankful to our project coordinator [Dr. Aloke Datta] for his guidance, supervision, and support throughout the project. His valuable insights, suggestions, and timely feedback have been instrumental in shaping our research to achieve the ultimate goals. We are also grateful that we have being provided with the opportunity to work in a team which helped us inculcating team as well as time management skills. We are grateful to the other faculty members who have reviewed our work and provided us with constructive feedback, which has helped us improving the quality of our research and analysis. Finally, we would like to thank the institution [The LNM Institute of Information Technology] for providing us with the opportunity to pursue this project and for the resources and infrastructure that have made it possible.

We am deeply indebted to all of these individuals and organizations, and their support has played a vital role in the successful completion of our BTP.

Abstract

In our project of Smoke plume segmentation , we need to identify and extract smoke plumes from images or videos captured. Smoke plumes are typically generated from fires, industrial processes, or other sources of combustion, and can be a major hazard to health and safety. Smoke plume segmentation is an important task in many fields, including environmental monitoring, public safety, and industrial process control.

The process of smoke plume segmentation typically involves using computer vision and image processing techniques to identify regions of an image or video that contain smoke. This can be done using various algorithms that analyze the color, texture, and shape of the smoke plume, as well as its motion and other features. Once the smoke plume has been segmented, it can be analyzed further to extract information such as its size, shape, and location, which can be used to assess the extent of the smoke plume and the potential hazards it may pose.

Smoke plume segmentation is an important tool for monitoring and controlling smoke emissions in many industries, as well as for assessing the impact of wildfires and other environmental hazards. It can also be used to develop early warning systems and to support emergency response efforts in the event of a fire or other disaster.

Therefore, we will be heading towards smoke plume segmentation starting through certain image and vedio processing techniques in order to postprocess the data to segment out the smoke from them in first phase of our BTP. Following it, some machine learning or computer vision techniques will be used to extract useful info from the segmented smoke in the later phase.

Contents

List of Figures **vii**

List of Tables **viii**

1 Introduction **1**
1.1 The Area of Work 1
1.2 Problem Addressed 2

2 Image Processing Techniques **3**
2.1 Conversion of image to grayscale 3
2.2 Histogram Equaliztion 4
2.3 Morplological Techniques : Erosion and Dilation 5
2.4 Filtering : Median , Low and High Pass Filter 7
2.5 Thresholding Techniques 9
2.6 Largest Component Computation 9

3 Applied Thresholding Techniques **12**
3.1 Otsu's Method 12
3.2 Kittler Method 13
3.3 Deluca Entropy Method 15
3.4 Evaluation of used thresholding techniques 17
3.4.1 Evaluation on Image 1 17
3.4.2 Evaluation on Image 2 17

4 Video Processing **19**
4.1 Video Segmentation: Without any pre-knowledge 20
4.1.1 Frame-by-Frame Segmentation 20
4.1.2 Median Frame Segmentation 21
4.1.3 Comparative analysis of both Techniques 21

5 Segmentation Using Transfer Learning **23**
5.1 Feature Extraction: Without using any pre-trained model 24
5.2 Feature Extraction: Using VGG16 pre-trained model 26

6 Conclusions and Future Work **28**

Bibliography **28**

List of Figures

2.1 RGB to Grayscale image 3
2.2 Code for Histogram Equalization of an image 5
2.3 Histogram Equalization 5
2.4 Images of Histogram Equalization 6
2.5 Code for edge detection using dilation and erosion techniques 7
2.6 Edge Detection of image 7
2.7 Median Pass Image Filter 8
2.8 Low Pass Image Filter 8
2.9 High Pass Image Filter 9
2.10 Code for largest component 10
2.11 Largest Connected Component: Step 1 - Thresholding 10
2.12 Largest Connected Component: Step 2 - Erosion and Dilation 11
2.13 Final image of largest component 11

3.1 Code for Otsu Thresholding 13
3.2 Smoke Segmentation using Otsu Image 1 14
3.3 Smoke Segmentation using Otsu Image 2 14
3.4 Smoke Segmentation using Kittler Image 1 15
3.5 Smoke Segmentation using Kittler Image 2 15
3.6 Code for Deluca Entropy Thresholding 16
3.7 Smoke Segmentation using Deluca Image 1 17
3.8 Smoke Segmentation using Deluca Image 2 17

4.1 Frame by Frame Video Segmentation 21
4.2 Median Frame Video Segmentation 22

List of Tables

3.1 Quantitative Analysis of Image 1 17
3.2 Quantitative Analysis of Image 2 18

4.1 Quantitative Analysis of Frame by Frame method 22
4.2 Quantitative Analysis of Median Frame method 22
4.3 Time Analysis of both Techniques 22

Chapter 1

Introduction

1.1 The Area of Work

Smoke plume segmentation is a common application of image processing, especially in the field of environmental monitoring and pollution control. The goal of smoke plume segmentation is to detect and segment smoke plumes from images captured by remote sensing devices such as satellites or drones.

The process of smoke plume segmentation typically involves the following steps:

- Image Acquisition: The first step is to acquire images of the area under observation. This can be done using various remote sensing techniques, such as satellites, drones, or ground-based sensors.
- Preprocessing: Once the images are acquired, they are preprocessed to remove noise and other artifacts that can interfere with the segmentation process. This may involve techniques such as image filtering, noise reduction, and image equalization.
- Smoke Plume Detection: The next step is to detect the smoke plume in the preprocessed image. This can be done using various techniques such as thresholding, edge detection, or region growing. Thresholding involves setting a pixel intensity value above which the pixel is considered part of the smoke plume. Edge detection involves finding the edges of the smoke plume by looking for abrupt changes in pixel intensity values. Region growing involves grouping adjacent pixels that have similar intensity values.However, in this project we have used various thresholding techniques for the detection of smoke.
- Smoke Plume Segmentation: Once the smoke plume is detected, it needs to be segmented or separated from the rest of the image. This can be done using various techniques such as active contours, watershed segmentation, or graph-based segmentation. Active contours involve defining a boundary around the smoke plume and iteratively

adjusting the boundary until it fits the smoke plume shape. Watershed segmentation involves dividing the image into regions based on the intensity gradient and identifying the smoke plume as a separate region. Graph-based segmentation involves modeling the image as a graph and using graph algorithms to segment the smoke plume.

- Smoke Plume Analysis: Once the smoke plume is segmented, various parameters such as its size, shape, and location can be analyzed to provide insights into the source and extent of the smoke plume. This can be done using various statistical and machine learning techniques.

Overall, smoke plume segmentation is an important application of image processing that can provide valuable insights into environmental pollution and help in developing strategies to mitigate its impact.

1.2 Problem Addressed

Once the useful information is extracted by detecting smoke plumes , it can be used for a variety of applications, such as:

- Environmental monitoring: Smoke plume segmentation can be used to track the extent and movement of smoke emissions from industrial processes, wildfires, and other sources of combustion. This information can be used to assess the impact of these emissions on air quality, human health, and the environment.
- Public safety: Smoke plume segmentation can be used to develop early warning systems for wildfires and other hazards, and to support emergency response efforts by providing real-time information on the location and extent of smoke plumes.
- Industrial process control: Smoke plume segmentation can be used to monitor and optimize industrial processes that generate smoke emissions, such as power generation, manufacturing, and waste incineration. This can help reduce emissions and improve environmental performance.

Overall, smoke plume segmentation is an important tool for understanding and managing the impact of smoke emissions on human health and the environment, and for supporting a range of industrial and public safety applications.

Chapter 2

Image Processing Techniques

Image processing is the use of algorithms to manipulate and analyze digital images. It involves applying techniques to improve the quality, enhance the information content, or extract useful information from images. Here we will be mentioning about all the image processing techniques that have been used in order to postprocess given images. [1]

2.1 Conversion of image to grayscale

Converting images to grayscale before applying image processing techniques simplifies the processing algorithms, improves the quality of the images, and makes them more suitable for various image processing applications since grayscale images have only one color channel, whereas color images have three color channels (red, green, and blue). This simplifies the image processing algorithms and reduces the computational complexity.

$$I_O = (I_R + I_G + I_B)/3 \tag{2.1}$$

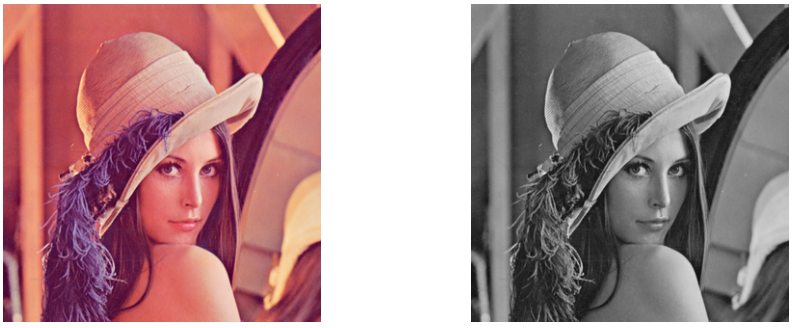


FIGURE 2.1: RGB (left) to Grayscale (Right) image

2.2 Histogram Equaliztion

Histogram equalization is a technique used in image processing to improve the contrast and brightness of an image. The aim of histogram equalization is to enhance the overall appearance of an image by stretching the range of intensity values to cover the entire range of possible values.

In histogram equalization, the cumulative distribution function (CDF) of the intensity values of an image is calculated, and the CDF is then equalized by mapping the intensity values to a new range of values. The result is an image with a more uniform distribution of intensity values, which enhances the contrast and details in the image.However, it may not always produce desirable results, especially when the original image has a narrow or irregular distribution of intensity values.

Algorithm

(1) Normailize Histogram

$$p(I_k) = n_k/n \tag{2.2}$$

(2) Apply Cumulative Tranformation Function

$$S_k = \sum_{i=0}^{k} p(I_i) \tag{2.3}$$

(3) Intensity of Output Image

$$S'_k = INT([L-1] * S_k) \tag{2.4}$$

```python
def equalize(histo,height,width,image):
    size=height*width # calculating size
   # eqTable = [0]*len(histo) # Declaring mapping table
    #eqTable = np.array([len(histo)],dtype='int64')
    eqTable = np.zeros(len(histo), dtype='int64') # Histogram Array

    # Traversing through histogram and calculating cumulative histogram
    for i in range(len(histo)):
        if(i == 0):
            eqTable[i] = round((histo[i]*(len(histo)-1))/size)
        else:
            histo[i] += histo[i-1]
            eqTable[i] = round((histo[i]*(len(histo)-1))/size)
            print(eqTable[i])

    # Equalizing image using mapping table
    image = (np.around(image[:, :] * 255)).astype(int)
    for i in range(height):
        for j in  range(width):
            image[i][j] = eqTable[image[i][j]]

    # Scaling intensity levels from [0...255] to [0...1]
    image = image[:,:]/len(histo-1)
    return image
```

FIGURE 2.2: Code for Histogram Equalization of an image

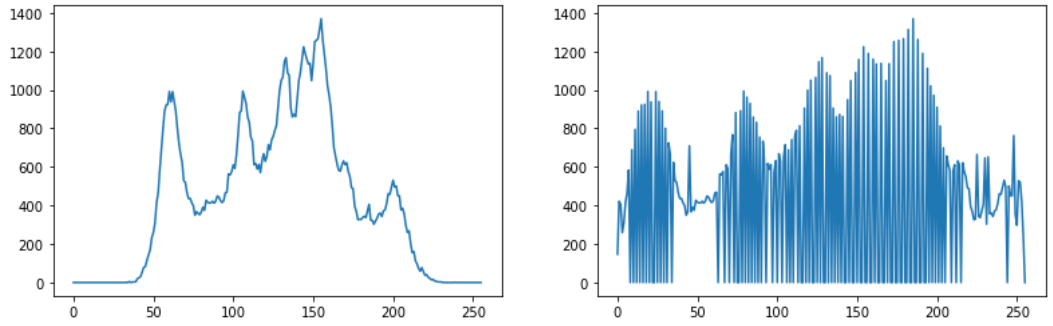


FIGURE 2.3: Original (left) to Equalized (Right) Histogram

2.3 Morplological Techniques : Erosion and Dilation

Morphological operations are fundamental image processing techniques that modify the shape and structure of objects in an image. Two commonly used morphological operations are erosion and dilation.

- Erosion: Erosion is a morphological operation that shrinks the boundaries of objects in an image. It does this by removing pixels from the boundaries of objects in the image based on a structuring element. The structuring element is a small binary image that is placed over the input image, and each pixel in the structuring element is compared to

FIGURE 2.4: Original (left) to Equalized (Right) Image

the corresponding pixel in the input image. If all the pixels in the structuring element overlap with the object in the input image, then the center pixel in the structuring element is retained in the output image, otherwise, it is removed. Erosion can be used for various purposes such as noise reduction, object separation, and object boundary detection.

- Erosion of image f is done by structuring element s
- The structuring element s is positioned with its origin at (x, y) and the new pixel value is determined using the rule:

$$g(x,y) = \begin{cases} 1, \text{if } s \text{ fits } f \\ 0, \text{otherwise} \end{cases}$$

- Dilation: Dilation is a morphological operation that expands the boundaries of objects in an image. It does this by adding pixels to the boundaries of objects in the image based on a structuring element.The structuring element is placed over the input image, and each pixel in the structuring element is compared to the corresponding pixel in the input image. If any pixel in the structuring element overlaps with the object in the input image, then the center pixel in the structuring element is retained in the output image. Dilation can be used for various purposes such as filling gaps in objects, joining broken objects, and thickening object boundaries.

  - Dilation of image f is done by structuring element s
  - The structuring element s is positioned with its origin at (x, y) and the new pixel value is determined using the rule:

$$g(x,y) = \begin{cases} 1, \text{if } s \text{ hits } f \\ 0, \text{otherwise} \end{cases}$$

```
#edge detection
boat = cv.imread("boat.jpg")
boat_gray = cv.cvtColor(boat, cv.COLOR_BGR2GRAY)
kernel = np.ones((5,5),np.uint8)
erosion = cv.erode(boat_gray,kernel,iterations = 1)

# Apply dilation to increase the size of foreground pixels
dilation = cv.dilate(erosion,kernel,iterations = 2)
erosion = cv.erode(dilation,kernel,iterations = 2)
dilation = cv.dilate(erosion,kernel,iterations = 2)
# Subtract the eroded image from the dilated image to get the edges
edges = cv.absdiff(dilation,erosion)

# Threshold the image to get binary image
threshold = cv.threshold(edges, 50, 255, cv.THRESH_BINARY)[1]

plt.imshow(boat , cmap="gray")
plt.axis('off')
plt.title('Boat Image')
plt.show()
```

FIGURE 2.5: Code for edge detection using dilation and erosion techniques

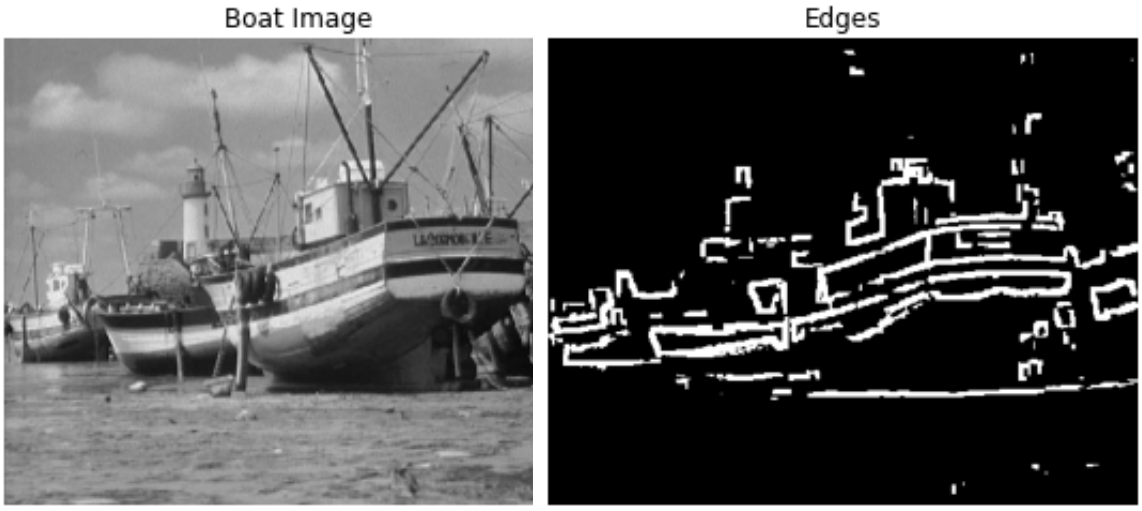


FIGURE 2.6: Edge Detection of image

2.4 Filtering : Median , Low and High Pass Filter

Filtering is a fundamental image processing technique that is used to enhance, blur or remove certain features in an image. Three commonly used filters in image processing are the median filter, low-pass filter, and high-pass filter.

- Median Filter: The median filter is a non-linear filter that replaces each pixel in an image with the median value of its neighboring pixels. It is often used to remove noise from

an image while preserving its edges. The median filter works by sorting the neighboring pixels of a central pixel in ascending or descending order and then selecting the median value as the new value for the central pixel.

FIGURE 2.7: Original (left) to Median Filtered (Right) Image

- Low-Pass Filter: A low-pass filter is a linear filter that allows low-frequency components of an image to pass through while attenuating high-frequency components. It is often used to remove high-frequency noise from an image, resulting in a smoother image. The low-pass filter works by convolving the image with a filter kernel, which assigns weights to each pixel in the image and its neighboring pixels. The weighted average of these pixels is then computed, and this value replaces the central pixel in the output image.

FIGURE 2.8: Original (left) to Low Pass Filtered (Right) Image

- High-Pass Filter: A high-pass filter is a linear filter that allows high-frequency components of an image to pass through while attenuating low-frequency components. It is often used to enhance the edges and details in an image, resulting in a sharper image.

The high-pass filter works by convolving the image with a filter kernel that emphasizes the differences between neighboring pixels. The result is an image that highlights the edges and details in the image.

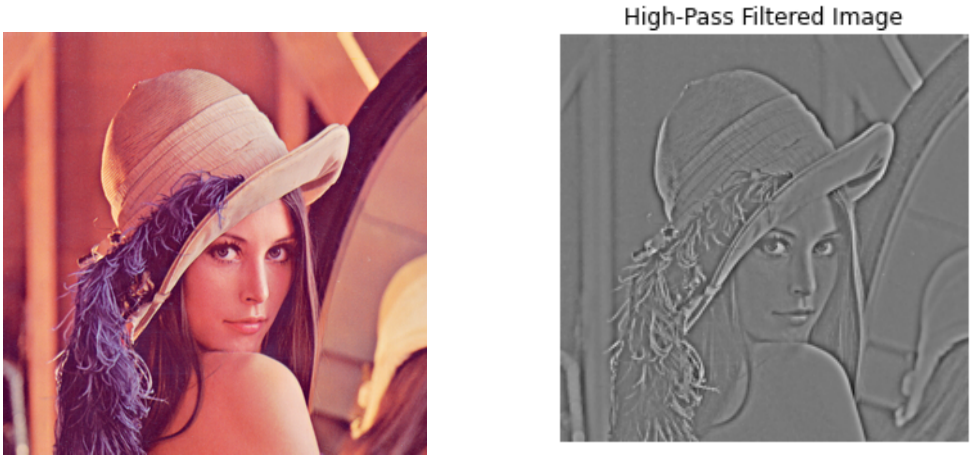


FIGURE 2.9: Original (left) to High Pass Filtered (Right) Image

2.5 Thresholding Techniques

Thresholding is a image processing technique used to segment an image into foreground and background regions. In thresholding, a grayscale or color image is converted to a binary image, where the pixels in the foreground are assigned one value, usually white (255), while the pixels in the background are assigned another value, usually black (0).

The threshold value is a critical parameter in thresholding, which determines the separation between the foreground and background regions. All pixels with intensity values above the threshold are assigned to the foreground, while those below the threshold are assigned to the background. There are various thresholding techniques involving Binary thresholding , Otsu's Method , Deluca's Entropy Method , Kittler Method , and many more which we will be going through a little later.

2.6 Largest Component Computation

In order to find the largest component in an image , multiple image processing techniques are used. Algorithm for largest component compuatation is given below:

- Load the image and convert it into grayscale.

- Apply a threshold to the grayscale image to convert it into a binary image, where each pixel is either black or white depending on whether it is above or below a certain threshold value.
- Compute the size of each connected component by counting the number of pixels in each component.
- Find the largest component by selecting the connected component with the largest number of pixels.

```python
def Largest_Component (image):
    image = image.astype('uint8')
    nb_components, output, stats, centroids = cv.connectedComponentsWithStats(image, connectivity=8)
    sizes = stats[:, -1]

    max_label = 1
    max_size = sizes[1]
    for i in range(2, nb_components):
        if sizes[i] > max_size:
            max_label = i
            max_size = sizes[i]

    img2 = np.zeros(output.shape)
    img2[output == max_label] = 255
    return img2
```

FIGURE 2.10: Code for largest component

FIGURE 2.11: Original (left) to Thresholded (Right) Image

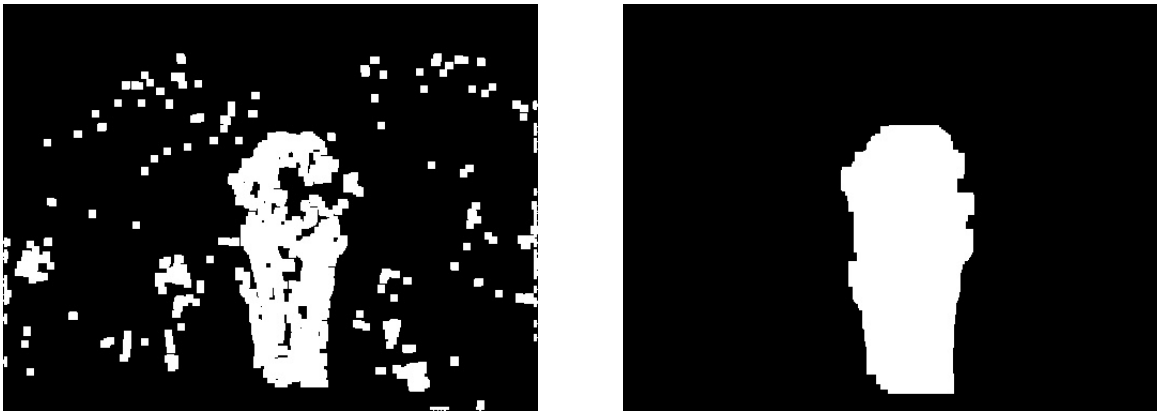


FIGURE 2.12: Eroded and Dilated (left) to Largest Connected Component (Right) Image

FIGURE 2.13: Final image of largest component

Chapter 3

Applied Thresholding Techniques

Here , we will be discussing about some of the most used thresholding techniques , their implementation and their qualitative analysis. Finally , we will be having a comparative study among all the techniques with the compuatation of false alarms , overall error , optimal thresholded value given by them and accuracy compuatation. Below is the description of each of the thresholding techniques used. [2]

3.1 Otsu's Method

Otsu's thresholding method is a widely used image processing technique for automatic thresholding .It determines the optimal threshold value that separates the foreground and background regions of an image based on the intensity values of the pixels. The main idea behind Otsu's method is to maximize the between-class variance of the intensity values of the pixels in the image. The between-class variance is a measure of the separation between the foreground and background regions. The optimal threshold value is the one that maximizes the between-class variance. However, it may not always produce desirable results, especially when the foreground and background regions have complex distributions of intensity values.

The steps involved in Otsu's thresholding method are as follows:

- Compute histogram and probabilities of each intensity level.
- Set up initial $\omega_i(0)$ and $\mu_i(0)$

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i) \tag{3.1}$$

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)} \tag{3.2}$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i) \tag{3.3}$$

$$\mu_1(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)} \tag{3.4}$$

- Step through all possible thresholds $t = 1, \ldots$ maximum intensity
  - Update ω_i and μ_i
  - Compute $\sigma_b^2(t)$

$$\sigma_b^2(t) = \omega_0(t) * \omega_1(t)[\mu_0(t) - \mu_1(t)]^2 \tag{3.5}$$

- Desired threshold corresponds to the maximum $\sigma_b^2(t)$
- Obtain the binary image by assigning the foreground and background values based on the threshold value.

```
def otsu_threshold(img):
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    hist = cv.calcHist([gray],[0],None,[256],[0,256])
    hist_norm = hist.ravel()/hist.max()
    Q = hist_norm.cumsum()
    bins = np.arange(256)
    fn_min = np.inf
    thresh = -1
    for i in range(1,256):
        p1,p2 = np.hsplit(hist_norm,[i])
        q1,q2 = Q[i] + 0.000001 , Q[255]-Q[i] + 0.000001
        b1,b2 = np.hsplit(bins,[i])
        m1,m2 = np.sum(p1*b1)/q1, np.sum(p2*b2)/q2
        v1,v2 = np.sum(((b1-m1)**2)*p1)/q1,np.sum(((b2-m2)**2)*p2)/q2
        fn = v1*q1 + v2*q2
        if fn < fn_min:
            fn_min = fn
            thresh = i
    ret, threshold = cv.threshold(gray, thresh, 255, cv.THRESH_BINARY)
    return threshold
```

FIGURE 3.1: Code for Otsu Thresholding

3.2 Kittler Method

The Kittler thresholding method is a widely used image processing technique for automatic thresholding to determine the optimal threshold value that separates the foreground and background regions of an image based on the intensity values of the pixels.In this method , the histogram is viewed as an estimate of the probability density function of the mixture population comprising the grey levels of the changed and unchanged pixels. It is assumed that the probability density functions and of the changed and unchanged classes, respectively are normally

FIGURE 3.2: Original (left) Image, Segmented image (right)

FIGURE 3.3: Original (left) Image, Segmented image (right)

distributed with mean and standard deviation. The steps involved in the Kittler thresholding method are as follows:

- Step through all possible thresholds $t = 1, \ldots$ maximum intensity
  - Compute prior probablities of non-smoke P_0 and smoke pixels P_1

$$P_0 = \frac{n_0}{n} \tag{3.6}$$

$$P_1 = \frac{n_1}{n} \tag{3.7}$$

  - Compute variance of both classes.

$$\sigma_0^2 = \frac{\sum_{i=0}^{t}(i-\mu_0)^2 p_i}{P_0} \tag{3.8}$$

$$\sigma_1^2 = \frac{\sum_{i=t+1}^{L-1}(i-\mu_1)^2 p_i}{P_1} \tag{3.9}$$

  - Compute $J(t)$

$$J(t) = 1 + 2(P_0 \log \sigma_0 + P_1 \log \sigma_1) - 2(P_0 \log P_0 + P_1 \log P_1) \tag{3.10}$$

- Desired threshold corresponds to the minimum $J(t)$

FIGURE 3.4: Original (left) Image, Segmented image (right)

FIGURE 3.5: Original (left) Image, Segmented image (right)

The Kittler thresholding method is a powerful and robust technique for automatic thresholding, especially in cases where the foreground and background regions have complex distributions of intensity values. However, it may be computationally expensive and require longer processing times than other thresholding techniques.

3.3 Deluca Entropy Method

This method is based on the concept of fuzzy set theory, which allows for the partial membership of each pixel to different classes, rather than assigning a binary label of foreground or background. In the deLuca entropy method, the threshold value is determined by maximizing the entropy of the fuzzy sets. The entropy is a measure of the uncertainty or randomness of the membership values of the pixels in the fuzzy sets. The optimal threshold value is the one that maximizes the entropy, and it is determined iteratively using an optimization algorithm.It is a powerful technique for image segmentation, especially in cases where the foreground and background regions have complex distributions of intensity values.

The deLuca entropy method of fuzzy thresholding consists of the following steps:

- Initialize a BandWidth ΔT and membership function μ_D

$$\Delta T = t - a = c - t = b = \frac{a+c}{2} \tag{3.11}$$

$$\mu_D(i) = \begin{cases} 0, \text{ if } i \leq a \\ 2(\frac{i-a}{c-a})^2, \text{ if } a \leq i \leq b \\ 1 - 2(\frac{i-c}{c-a})^2, \text{ if } b \leq i \leq c \\ 1, \text{ if } c \leq i \end{cases} \tag{3.12}$$

- Step through all possible thresholds $t = 1, \ldots$ maximum intensity
  - Compute entropy $H(t)$

$$H(t) = \frac{\sum_{i=0}^{L-1}[-\mu_D(i)\ln(\mu_D(i)) - (1-\mu_D(i))\ln(1-\mu_D(i))]n_i}{pq\ln 2} \tag{3.13}$$

- Desired threshold corresponds to the minimum $H(t)$

```
def de_luca_entropy(image,band):
    (height, width) = image.shape[:2]
    histo = makeHistoINT(image)
    threshold = {}

    for lvl in range(70,230):
      mem=calcMem(lvl,band)
      sum=0
      for i in range(256):
        if mem[i]==0 or mem[i]==1:
          sum+=0
        else:
          sum+=(-1*mem[i]*m.log(mem[i])-(1-mem[i])*(m.log(1-mem[i])))*histo[i]
      entropy=1*(sum/(height*width*m.log(2)))
      threshold[lvl]=entropy
    return threshold
    # return min(threshold,key=threshold.get)
```

FIGURE 3.6: Code for Deluca Entropy Thresholding

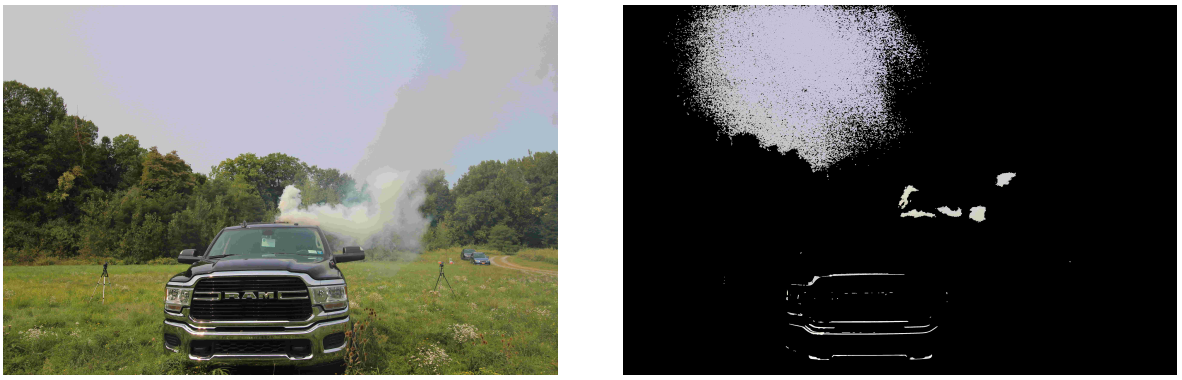


FIGURE 3.7: Original (left) Image, Segmented image (right)

FIGURE 3.8: Original (left) Image, Segmented image (right)

3.4 Evaluation of used thresholding techniques

We will be evaluating the above three thresholding techniques based on metrics involving
False Alarms(FA) - non-smoke pixels identified as smoke
Miss Alarms(MA) - smoke pixels identified as non-smoke
overall error , number of correct pixels detected , thresholded value computed by the algorithm and accuracy. It will be compared with the ideal segmented smoke plume image(done manually) as shown below:

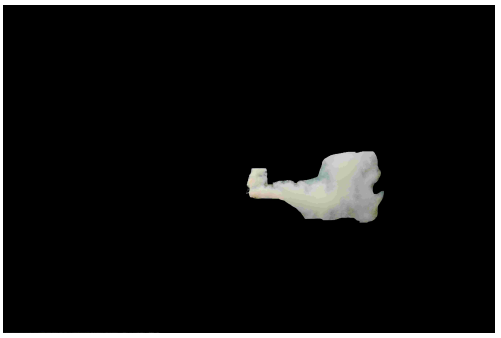
3.4.1 Evaluation on Image 1

METHOD	FA	MA	CORRECT	ACCURACY
OTSU	7289453	27196	10599255	59.1611
KITTLER	6986983	327691	10601230	59.1774
DELUCA	2111442	581854	15222608	84.967

TABLE 3.1: Quantitative Analysis of Image 1

3.4.2 Evaluation on Image 2

(a) Original Image1



(b) Ideal smoke segmented image1

(c) Original Image2

(d) Ideal smoke segmented image2

METHOD	FA	MA	CORRECT	ACCURACY
OTSU	128975	91431	4258570	95.079
KITTLER	79740	150225	4249011	94.8656
DELUCA	73889	251923	4153164	92.7257

TABLE 3.2: Quantitative Analysis of Image 2

From the above evaluation , we can conclude that the Otsu method is particularly effective or give better results and close to ideal accuracy when the image has a bimodal intensity histogram, meaning that the intensities are clustered into two distinct groups. In this case as in Image 2, the Otsu method can accurately determine the threshold value that separates the two groups.

However, the Otsu method may not perform as well when the image has a unimodal intensity histogram, where the intensity values are distributed more evenly across a wide range. In such cases, other thresholding method such as DeLuca Entropy thresholding may be more appropriate as seen in Image 1 where the foreground and background were not distinguished separately because of the sky.

Therefore, it depends on input data that which thresholding technique should be applied in which scenario in order to generate best results.

Chapter 4

Video Processing

Smoke plume segmentation in video processing involves isolating and delineating regions containing smoke within a video sequence. This task is crucial in various applications like fire monitoring, pollution detection, or environmental analysis. Here are the steps involved in smoke plume segmentation:

- Frame Extraction: Extract frames from the video sequence.

  Color Space Conversion: Convert the frames to different color spaces (e.g., RGB, HSV, YCbCr) to identify the color characteristics of smoke, as smoke often has distinct color properties.

  Noise Reduction: Apply filters (e.g., Gaussian blur, median filter) to reduce noise and smoothen the frames.

- Feature Extraction:

  Color-based Features: Identify color thresholds or histograms that represent the characteristics of smoke in the chosen color space. This helps in distinguishing smoke from the background.

  Texture and Shape Features: Analyze texture or shape characteristics that are common in smoke plumes, such as irregular patterns or elongated shapes.

- Segmentation Techniques:

  Thresholding: Apply color thresholding techniques to segment smoke regions based on predefined color characteristics. This involves setting thresholds on specific color channels to extract smoke-like regions.

  Edge Detection: Employ edge detection algorithms (e.g., Canny edge detector) to highlight the boundaries of smoke plumes.

  Region-based Segmentation: Utilize region-based segmentation methods (e.g., watershed segmentation) to separate smoke regions from the background.

- Post-processing:

  Morphological Operations: Use morphological operations (e.g., dilation, erosion) to refine the segmented regions and remove small artifacts or noise.

  Connected Component Analysis: Perform connected component analysis to identify and label different smoke regions within each frame.

- Validation and Evaluation:

  Visual Inspection: Visually inspect the segmented results to ensure accuracy and make adjustments if necessary.

  Quantitative Metrics: Measure the performance of the segmentation using metrics like precision, recall, and F1-score against ground truth annotations or expert-labeled data.

- Refinement and Optimization:

  Parameter Tuning: Adjust parameters of segmentation algorithms to optimize performance.

  Machine Learning Integration: Consider utilizing machine learning models (e.g., neural networks, SVMs) for smoke detection and segmentation, especially when dealing with complex or diverse smoke patterns.

These steps constitute a systematic approach to segmenting smoke plumes within video sequences, aiming to accurately identify and isolate regions containing smoke for various practical applications. [3]

4.1 Video Segmentation: Without any pre-knowledge

In this section, we will not be extracting any features of an image using a model or feeding the dataset to a machine learning model. Rather, we will be carrying out thresholding techniques on different parts of the input video to get the required output. We have taken into consideration two techniques for segmenting videos into meaningful parts or extracting relevant information from them.

4.1.1 Frame-by-Frame Segmentation

Frame-by-frame segmentation involves analyzing each individual frame of a video independently to identify and segment objects, actions, or regions of interest. This technique treats each frame as a standalone image and applies image processing or computer vision algorithms to detect and segment objects or features within each frame. Common methods used in frame-by-frame segmentation include background subtraction, object detection, contour analysis, and

various segmentation algorithms like thresholding, watershed segmentation, or graph-based segmentation. While this method can provide accurate segmentation at a frame level, it might not consider temporal information or context across frames, potentially leading to inconsistencies or challenges in maintaining continuity across the video sequence. In our case, we have used this technique to apply Otsu thresholding on each frame to get the segmented smoke and then combining the results into an output video.

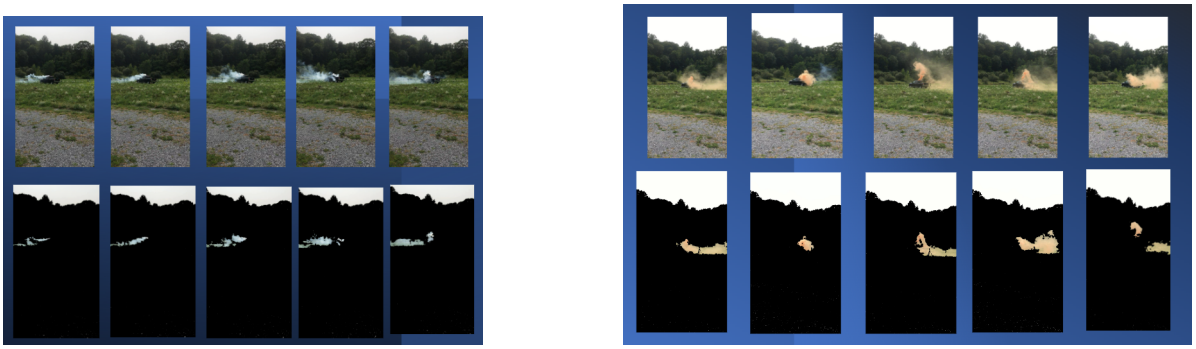


FIGURE 4.1: Video 1 Frames (left) , Video 2 Frames (right)

4.1.2 Median Frame Segmentation

median frame segmentation involves processing a sequence of frames by calculating a median frame from multiple frames at corresponding time points. The median frame is created by pixel-wise median calculations across the frames, essentially creating a representative frame that mitigates outliers or noise present in individual frames. This technique aims to reduce noise and retain relevant information across the video sequence. Median frame segmentation is particularly useful in scenarios where the content in a video is consistent across most frames, such as videos with static backgrounds and moving objects. It helps in background subtraction, anomaly detection, or identifying persistent objects in a video sequence by emphasizing the stable elements while minimizing transient changes or outliers. In our case, since smoke videos have less temporal differences, therefore median frame segmentation can led to less time consumption as thresholding techniques are applied only on median frames.

4.1.3 Comparative analysis of both Techniques

Frame-by-frame segmentation offers detailed segmentation at the expense of computational resources and may lack temporal consistency. Median frame segmentation, while computationally efficient and robust to outliers, might struggle with videos containing complex, dynamic scenes or rapid changes. The choice between these methods often depends on the specific characteristics of the video data and the objectives of the video processing task at hand. Integrating these techniques or combining them with other approaches can often enhance the accuracy and reliability of video segmentation tasks.

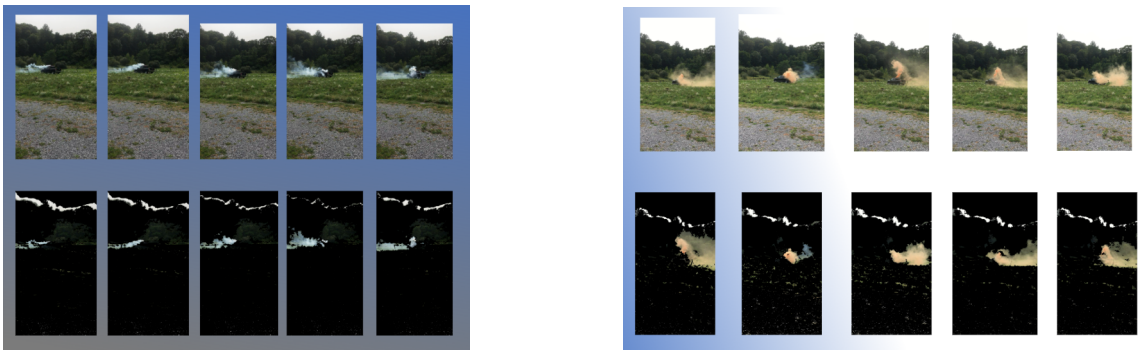


FIGURE 4.2: Video 1 Frames (left) , Video 2 Frames (right)

	FA	MA	Correct	Accuracy
Video 1	223332	2243	1848023	89.121
Video 2	443479	6988	1623132	78.276

TABLE 4.1: Quantitative Analysis of Frame by Frame method

	FA	MA	Correct	Accuracy
Video 1	254301	1033	1818265	87.686
Video 2	258110	9888	1813512	87.457

TABLE 4.2: Quantitative Analysis of Median Frame method

	Frame By Frame Segmentation (In Seconds)	Median Frame Segmentation (In Seconds)	Correct	Accuracy
Video 1	84.64619875	82.01575732	1818265	87.686
Video 2	73.61024022	71.88774872	1813512	87.457

TABLE 4.3: Time Analysis of both Techniques

Chapter 5

Segmentation Using Transfer Learning

Extracting features using transfer learning [4] and then training a machine learning model is a common approach to leverage pre-trained deep learning models for various tasks. Here are the steps involved in this process:

- Select a Pre-trained Model: Choose a pre-trained deep learning model that has been trained on a large and diverse dataset. Popular choices include models like VGG, ResNet, Inception, or MobileNet, which are often pre-trained on ImageNet.
- Data Preparation:

  a. Collect and preprocess your dataset for the specific task. Ensure that the data is properly labeled and organized.

  b. Resize and normalize the data to match the input requirements of the pre-trained model. This may involve resizing images or applying specific data transformations.
- Feature Extraction:

  a. Initialize the model with the weights of the pre-trained model and freeze some of the layers especially the classification layer, in the early stages of the network. These frozen layers retain their learned knowledge, while you update the weights of the new layers you added for your specific task.

  b. Pass your dataset through the pre-trained model to extract features from the intermediate layers. These features capture high-level representations of the data.

  c. Store the extracted features for each data point in your dataset.
- Data Split: Split your dataset into training, validation, and test sets. This ensures that you can train, validate, and evaluate the performance of your machine learning model.

- Machine Learning Model Selection: Choose the machine learning model you want to use for your specific task. The choice of the model depends on your problem, such as classification, regression, or clustering.

- Model Training:

  a. Feed the extracted features into your selected machine learning model. You can use traditional models like SVM, logistic regression, or random forests.

  b. Train the machine learning model on the training data, using the extracted features as input and the corresponding labels as targets.

  c. Hyperparameter tuning: Experiment with different hyperparameters to optimize the model's performance, such as regularization strength, learning rate, and model architecture.

- Model Validation: Evaluate the performance of your machine learning model using the validation dataset. Use appropriate metrics for your task, such as accuracy, F1 score, mean squared error, or others.

- Fine-Tuning: If the model's performance is not satisfactory, consider fine-tuning the feature extraction model by adjusting hyperparameters, changing the model architecture, or unfreezing more layers.

This approach allows you to leverage the feature extraction capabilities of pre-trained deep learning models, which have learned meaningful representations from large-scale datasets, and then apply machine learning models to make predictions or inferences based on these features. It can be especially useful when you have limited labeled data for your specific task but want to benefit from the knowledge learned by the pre-trained model.

5.1 Feature Extraction: Without using any pre-trained model

Here, we have built a method to extract features from the input image according to our needs. The below code utilizes various image processing techniques to generate a wide range of features and organize them into a DataFrame using the Pandas library. The extracted features include Gabor filter responses with different orientations, scales, and frequencies, which are computed for texture analysis. Additionally, it computes other image features such as Canny edges, Roberts edges, Sobel edges, Scharr edges, Prewitt edges, and various types of Gaussian and median filters with different sigma values. Furthermore, the code applies Otsu thresholding to segment the image into binary regions based on intensity. These extracted features can be useful for tasks like image classification, object detection. The resulting DataFrame df contains the original image pixels and all the extracted features, making it ready for further analysis or machine learning tasks.

```
def feature_extraction(img):
    df = pd.DataFrame()
    img2 = img.reshape(-1)
    df['Original Image'] = img2
    num = 1
    kernels = []


    #generate gabor filters
    for theta in range(2):
        theta = theta / 4. * np.pi
        for sigma in (1, 3):
            for lamda in np.arange(0, np.pi, np.pi / 4):
                for gamma in (0.05, 0.5):
                    if num in [20,28,13,2,9,10,19,14,16,27,26,25,17,18,1,15]:
                        num+=1
                        continue
                    gabor_label = 'Gabor' + str(num)
                    ksize=9
                    kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lamda, gamma, 0, ktype=cv2.CV_32F)
                    kernels.append(kernel)
                    fimg = cv2.filter2D(img2, cv2.CV_8UC3, kernel)
                    filtered_img = fimg.reshape(-1)
                    df[gabor_label] = filtered_img
                    num += 1
```

(a) Gabor filters as extracted features

```
#Geerate OTHER FEATURES and add them to the data frame
#Feature 3 is canny edge
    edges = cv2.Canny(img, 100,200)   #Image, min and max values
    edges1 = edges.reshape(-1)
    df['Canny Edge'] = edges1 #Add column to original dataframe

#Feature 4 is Roberts edge
    edge_roberts = roberts(img)
    edge_roberts1 = edge_roberts.reshape(-1)
    df['Roberts'] = edge_roberts1

#Feature 5 is Sobel
    edge_sobel = sobel(img)
    edge_sobel1 = edge_sobel.reshape(-1)
    df['Sobel'] = edge_sobel1

#Feature 6 is Scharr
    edge_scharr = scharr(img)
    edge_scharr1 = edge_scharr.reshape(-1)
    df['Scharr'] = edge_scharr1

    #Feature 7 is Prewitt
    edge_prewitt = prewitt(img)
    edge_prewitt1 = edge_prewitt.reshape(-1)
    df['Prewitt'] = edge_prewitt1

    #Feature 8 is Gaussian with sigma=3
    gaussian_img = nd.gaussian_filter(img, sigma=3)
    gaussian_img1 = gaussian_img.reshape(-1)
    df['Gaussian s3'] = gaussian_img1

    #Feature 9 is Gaussian with sigma=7
    gaussian_img2 = nd.gaussian_filter(img, sigma=7)
    gaussian_img3 = gaussian_img2.reshape(-1)
    df['Gaussian s7'] = gaussian_img3

    #Feature 10 is Median with sigma=3
    median_img = nd.median_filter(img, size=3)
    median_img1 = median_img.reshape(-1)
    df['Median s3'] = median_img1

    #Feature 11 is Otsu Thresholding
    ret,otsu_img = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    otsu_img1 = otsu_img.reshape(-1)
    df['Otsu'] = otsu_img1  #Add column to original dataframe
```

(b) Edges filter and Thresholded image as extracted features

After we have received the required features of an image, our image dataset including the original image frame and the feature frames is made to concat with mask dataset including the manually plotted frames depicting the actual segmented smoke in the given input image. The final dataset is now ready to be fed to a machine learning model for training and testing(here, we have used random forest model).

```
image_dataset = pd.DataFrame();
path = "images/train_images/*.jpg"
for file in glob.glob(path):
    img1= cv2.imread(file)
    img = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)

    X = feature_extraction(img)
    image_dataset = pd.concat([image_dataset,X],ignore_index = True)
```

```
mask_dataset = pd.DataFrame();
path = "images/train_masks/*.png"
for file in glob.glob(path):
    # print(file)
    temp = pd.DataFrame()
    img1= cv2.imread(file)
    img = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
    img = np.where(img>0,255,img)

    X = img.reshape(-1)
    temp['Labels'] = X
    mask_dataset = pd.concat([mask_dataset,temp],ignore_index = True)
```

(c) Image and mask dataset

```
dataset = pd.concat([image_dataset,mask_dataset],axis=1)
```

```
Y = dataset["Labels"].values

#Define the independent variables
X = dataset.drop(labels = ["Labels"], axis=1)

#Split data into train and test to verify accuracy after fitting the model.
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.4, random_state=20)
```

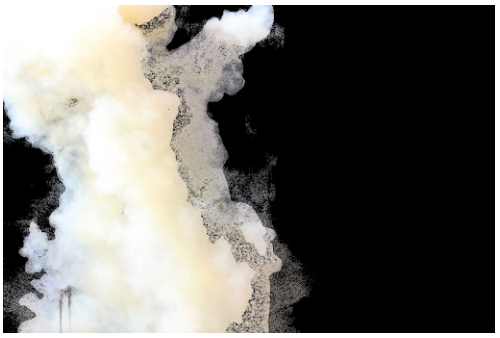
```
model = RandomForestClassifier(n_estimators = 25, random_state = 42,verbose=10)
```

```
model.fit(X_train, y_train)
```

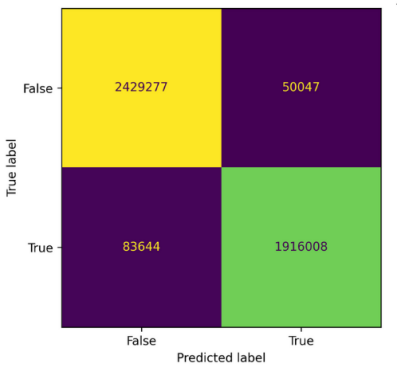
(d) Splitting and feeding dataset to Random Forest Model

Here are the results for the testing data including the original image, manually plotted segmented smoke image which is accurate, and the image received as output from the Random

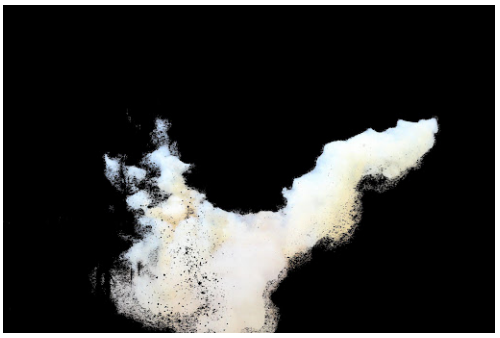
forest model. We have also calculated the metrics for the same.



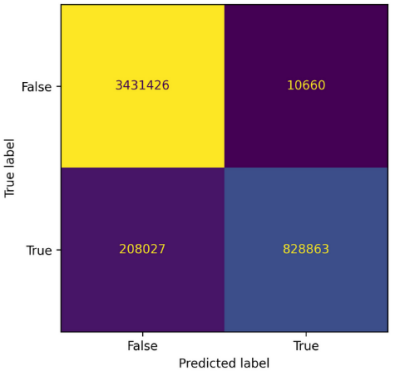
(e) Image 1



(f) Image 1 Confusion Matrix



(g) Image 2



(h) Image 2 Confusion Matrix

(a) Image 1 Evaluation Metrics

Measure	Value
Precision	0.9745
Recall	0.9582
Accuracy	0.9702
F1 Score	0.9663

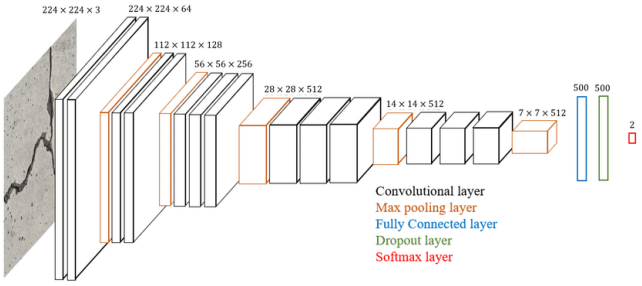
(b) Image 2 Evaluation Metrics

Measure	Value
Precision	0.9873
Recall	0.7994
Accuracy	0.9512
F1 Score	0.8835

5.2 Feature Extraction: Using VGG16 pre-trained model

If the VGG16 [5] model were used in the provided code to extract features from the input image, the process would be fundamentally different. The VGG16 model, a convolutional neural network (CNN) pre-trained on the ImageNet dataset, captures hierarchical and abstract representations of images through its layers. Instead of handcrafting features as done with Gabor filters, edge detectors, and filters of different types and scales, VGG16 would extract features automatically by passing the image through its layers. These features would represent hierarchical abstractions of the image, becoming more complex and abstract in deeper layers. The output from the final convolutional or fully connected layers of VGG16 would likely

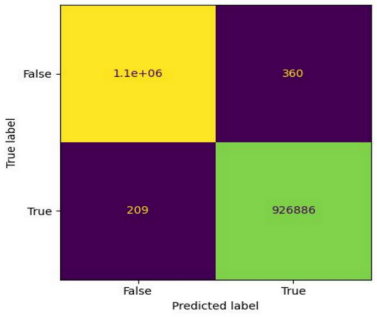
serve as the extracted features for the input image. These features, when flattened, could be organized into a DataFrame or directly fed into a machine learning model for various tasks such as image classification, object detection, or feature-based analysis.



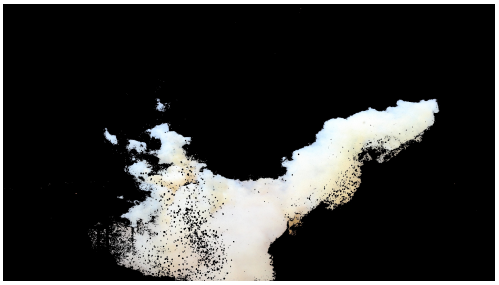
(i) VGG16 Architecture

Using a pre-trained CNN like VGG16 for feature extraction often leverages the learned representations from a large and diverse dataset, potentially offering more generalized and higher-level features for subsequent tasks.

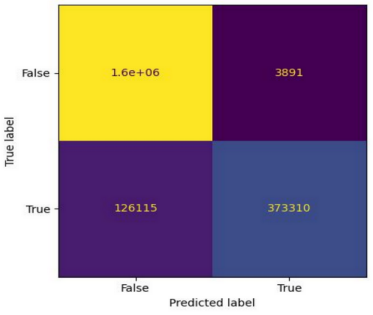
(j) Image 1



(k) Image 1 Confusion Matrix



(l) Image 2



(m) Image 2 Confusion Matrix

(c) Image 1 Evaluation Metrics

Measure	Value
Precision	0.9996
Recall	0.9998
Accuracy	0.9997
F1 Score	0.9997

(d) Image 2 Evaluation Metrics

Measure	Value
Precision	0.9897
Recall	0.7475
Accuracy	0.9373
F1 Score	0.8517

Chapter 6

Conclusions and Future Work

The development of an accurate smoke plume segmentation algorithm is essential for a variety of applications, including environmental monitoring, disaster management, and air quality control.

The segmentation algorithm can be implemented using various techniques such as thresholding, edge detection, morphological operations, and machine learning methods. Each technique has its own advantages and disadvantages, and the best choice depends on the specific requirements of the input given. However , in this project we have tried to use multiple thresholding and morphological techniques for smoke detection. The evaluation of the detection/segmentation algorithm has been done using various metrics such as false alarms , overall error , accuracy and precision, recall.

The segmentation of smoke plumes is a challenging task due to the complex and dynamic nature of smoke plumes. Factors such as wind speed, direction, and turbulence can affect the shape and size of the plume, making it difficult to accurately segment.The proposed or used algorithms can be further improved by incorporating additional features such as texture , shape , colour analysis of smoke and temporal information. The use of advanced techniques such as deep learning can also improve the accuracy of the segmentation.The results of the segmentation algorithm also depends on the quality of the input data. Therefore, it is essential to ensure that the input data is of high quality and captured under appropriate conditions.

Overall, the development of an accurate smoke plume segmentation algorithm is crucial for environmental monitoring and disaster management applications and further research or future work can lead to significant advancements in the results of segmentation and analysis of smoke plumes.

Bibliography

[1] R. C. Gonzalez, *Digital image processing*. Pearson education india, 2009.

[2] S. Patra, S. Ghosh, and A. Ghosh, "Histogram thresholding for unsupervised change detection of remote sensing images," *International journal of remote sensing*, vol. 32, no. 21, pp. 6071–6089, 2011.

[3] M. Malik and D. kavita, "Video processing and its application," 04 2020.

[4] S. Tammina, "Transfer learning using vgg-16 with deep convolutional neural network for classifying images," vol. 9, p. p9420, 10 2019.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.