AI-BASED PORT SURVEILLANCE AND ENCROACHMENT DETECTION

A PROJECT REPORT

Submitted by

JHANANI N (113221072020)

LEKHA S (113221072027)

In the partial fulfillment of the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

VELAMMAL
ENGINEERING COLLEGE
The Wheel of Knowledge rolls on!

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

VELAMMAL ENGINEERING COLLEGE, CHENNAI

APRIL 2024

VELEMMAL ENGINEERING COLLEGE

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

BONAFIDE CERTIFICATE

Certified that this project report titled "**AI BASED PORT SURVEILLANCE AND ENCROACHMENT DETECTION** is the bonafide work of **Ms. JHANANI N** (113221072020)**, Ms. LEKHA S** (113221072027)**,** who carried out the Mini project work under my supervision.

SIGNATURE

INTERNAL GUIDE,

Mrs. T. Hannah Rose Esther

Assistant Professor,

Department of Artificial Intelligence and Data Science,

Velammal Engineering College,

Chennai - 600066.

SIGNATURE

HEAD OF THE DEPARTMENT,

Dr. P. Visu,

Professor and Head,

Department of Artificial Intelligence and Data Science,

Velammal Engineering College,

Chennai - 600066.

MINI PROJECT EXAMINATION

The MINI PROJECT Examination of this project work "**AI BASED PORT SURVEILLANCE AND ENCROACHMENT DETECTION"** is a bonafide record of project done at the department of Artificial Intelligence and Data Science, Velammal Engineering College during the academic year 2023-2024 by

JHANANI N (113221072020)
LEKHA S (113221072027)

Of Third year Bachelor of Technology in Artificial Intelligence and Data Science submitted for the university examination held on .....

INTERNAL EXAMINER

ABSTRACT

In today's era, maritime safety and security are of paramount importance, with ports serving as vital hubs for global trade and transportation. However, ensuring effective surveillance and detecting encroachments in port areas remain challenging tasks. In this project, we propose an AI-based port surveillance and encroachment detection system aimed at enhancing maritime security and safety by detecting ships. The project leverages advanced technologies such as MATLAB and Python, along with deep learning models like AlexNet and VGG-16, to develop a comprehensive solution. The system consists of several modules, including preprocessing, feature extraction, segmentation, classification, and model evaluation, each playing a crucial role in achieving the project objectives The proposed system offers several advantages over existing methods, including improved accuracy, real-time monitoring capabilities, and enhanced maritime security. By providing early detection of encroachments and unauthorized activities in port areas, the system aims to mitigate potential risks and ensure the safety of maritime operations. Overall, the AI-based port surveillance and encroachment detection system presented in this project represent a significant step forward in enhancing maritime security and safety, contributing to the protection of critical infrastructure and safeguarding global maritime trade. However, their strategic importance also makes them susceptible to various security threats, including terrorism, smuggling, illegal immigration, and environmental hazards. Traditional surveillance methods often fall short of providing comprehensive coverage and timely threat detection, necessitating the development of advanced monitoring systems. The proposed system integrates cutting-edge artificial intelligence (AI) techniques with existing port surveillance infrastructure to create a robust and proactive monitoring framework.

ACKNOWLEDGEMENT

First of all, I thank the lord Almighty for showering blessings on me, which enabled to carry out this project successfully.

I express my sincere gratitude to our beloved and respected **Chairman**, **Dr.M.V.Muthuramalingam**, I wish to acknowledge with thanks to our college Chief Executive Officer, **Thiru.M.V.M.Velmurugan** for his support and encouragement also I would like to express my gratitude to **Thiru.V.Karthik Muthuramalingam**, the deputy CEO of our college, for his encouragement and assistance.

I extend my sincere gratitude to **Dr. S. Satish kumar, Principal**, for his motivation and support towards this project.

I express my sincere gratitude to **Dr. P. Visu**, **Professor and Head of the Department** of Artificial Intelligence and Data Science, who has been guiding force and constant source of inspiration.

I express my sincere thanks to the Project Coordinator, **Mrs.J.SAVIJA**, Assistant Professor, Department of Artificial Intelligence and Data Science for her valuable guidance in shaping this project.

I wish to express my sincere gratitude to my Project Guide, **Mrs. T. HANNAH ROSE ESTHER**, Assistant Professor, Department of Artificial Intelligence and Data Science for her guidance, support, motivation and encouragement throughout this project.

I also take this opportunity to thank all the **Faculty and Non-Teaching Staff Members** of Department of Artificial Intelligence and Data Science for their constant support. Finally I thank each and every one who helped me to complete this project.

JHANANI N

LEKHA S

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
1	**INTRODUCTION**	
	1.1 HARDWARE & SOFTWARE USED	**1**
	1.2 EXISTING SYSTEM	**3**
	1.3 PROPOSED SYSTEM	**4**
2	**LITERATURE SURVEY**	**6**
3	**SYSTEM ANALYSIS**	
	3.1 FUNCTIONAL REQUIREMENTS	**8**
	3.2 NON-FUNCTIONAL REQUIREMENTS	**9**
	3.3 LIBRARY AND TOOLS USED	**10**
4	**SYSTEM DESIGN**	
	4.1 DESIGN ARCHITECTURE	**11**
5	**MODULE IMPLEMENTATION**	
	5.1 PREPROCESSING MODULE	**12**
	5.2 FEATURE EXTRACTION	**14**
	5.3 SEGMENTATION MODULE	**15**
	5.4 CLASSIFICATION MODULE	**16**
6	**MODEL EVALUATION**	
	6.1 CROSS-ENTROPY LOSS	**27**
	6.2 ROC CURVE	**27**
	6.3 MEAN SQUARED ERROR	**29**
7	**OUTPUT SCREENSHOTS**	**31**

8	**CONCLUSION AND FUTURE ENHANCEMENT**	**34**
9	**REFERENCES**	**35**
	APPENDIX A- PREPROCESSING MODULE	**37**
	APPENDIX B- FEATURE EXTRACTION	**39**
	APPENDIX C- SEGMENTATION MODULE	**41**
	APPENDIX D- CLASSIFICATION MODULE	**44**
	APPENDIX E- TRAINING	**47**
	APPENDIX F- TESTING	**49**
	APPENDIX G- ERROR HANDLING	**50**
	APPENDIX H- MODEL EVALUATION	**51**
	APPENDIX I- CONFUSION MATRIX	**52**

CHAPTER 1

INTRODUCTION

Project Outline:

Title of the project: - AI-Based Port Surveillance and Encroachment Detection.

Tools/Platform:

Hardware Configuration

The proposed system is designed to run on hardware with the following specifications:

- **Intel Processor:** The system leverages the processing power of Intel processors to execute complex image processing and machine learning algorithms efficiently.

- **4GB RAM:** Sufficient RAM capacity ensures smooth execution of computational tasks, allowing for seamless processing of large datasets and real-time analysis of satellite imagery.

- **250GB HDD:** The storage capacity provided by the 250GB hard disk drive enables the system to store datasets, models, and other relevant files necessary for training and operation.

- **Windows OS:** The Windows operating system provides a familiar and user-friendly environment for software development and deployment. It offers compatibility with a wide range of software tools and libraries, including MATLAB, Python, and Anaconda Navigator.

Software Used

The project utilizes a combination of software tools and libraries to implement the proposed ship detection system:

- **MATLAB:** MATLAB serves as the primary programming environment for the development and testing of image processing algorithms, machine learning models, and data analysis tasks. Its rich set of built-in functions and toolboxes make it well-suited for prototyping and experimentation.

- **Python:** Python is used for its versatility and extensive library support, particularly in the realm of deep learning and computer vision. Libraries such as TensorFlow, Keras, and OpenCV are employed for model training, evaluation, and deployment.

- **Anaconda Navigator:** Anaconda Navigator provides a convenient platform for managing Python environments and packages. It offers a user-friendly interface for installing, updating, and managing software packages and dependencies, streamlining the development workflow.

Introduction:

In an era marked by global trade expansion and increasing security concerns, maritime ports serve as critical hubs for commerce, transportation, and national security. Ensuring the safety, security, and efficiency of port operations is paramount for governments, port authorities, and maritime stakeholders worldwide. However, the vast expanse of maritime environments, coupled with the diverse array of vessels navigating through ports, presents significant challenges for effective surveillance and monitoring. Our project, titled "AI-based Ship Detection and Feret Diameter Estimation for Port Surveillance," addresses these challenges by harnessing the power of artificial intelligence (AI) and computer vision technologies. At its core, the project focuses on the development of advanced algorithms and systems capable of automatically detecting ships within port areas, segmenting them from the surrounding background, and providing essential metrics such as the feret diameter for each detected vessel.

Motivation:

The motivation behind our project is multifaceted and driven by the evolving landscape of maritime security, operational efficiency, and technological advancements. Traditional methods of port surveillance, reliant on manual observation and labor-intensive processes, are inherently limited in scalability, accuracy, and responsiveness. Moreover, the increasing volume of maritime traffic, coupled with emerging security threats such as smuggling, piracy, and illegal encroachments, underscores the need for innovative and intelligent surveillance solutions. By leveraging AI and computer vision, our project seeks to address these challenges by offering automated, real-time ship detection and monitoring capabilities. The ability to swiftly and accurately identify vessels entering, exiting, or maneuvering within port areas is instrumental in enhancing situational awareness and facilitating timely decision-making for port authorities. Additionally, by estimating essential parameters such as the feret diameter of detected ships, our system provides valuable insights into vessel size, orientation, and potential security risks. Through this project, we aim to empower port authorities, maritime security agencies, and stakeholders with the tools and technologies needed to ensure the safety, security, and efficiency of port operations. By bridging the gap between cutting-edge AI research and practical maritime applications, we strive to contribute to the advancement of port surveillance methodologies and the protection of maritime assets and resources.

Existing System:

1. Manual Surveillance: Traditional port surveillance relies heavily on human operators to visually monitor maritime activities using cameras, radar, and other sensor systems.

2. Human-Centric: Surveillance tasks are primarily performed by human operators, leading to potential limitations such as human error, fatigue, and subjective interpretation of data.

3. Labor-Intensive: Manual surveillance is labor-intensive and time-consuming, requiring continuous monitoring by personnel to identify and respond to security threats or encroachments.

4. Limited Coverage: Due to manpower constraints and the inability to analyze vast amounts of data in real-time, manual surveillance may result in gaps or blind spots in coverage, leaving certain

port areas vulnerable to security breaches.

5. Dependence on Visual Inspection: Detection of ships and encroachments heavily relies on human visual inspection, which may overlook subtle or hidden threats, especially in large port areas with complex maritime activities.

6. Challenges in Analysis: Analyzing and interpreting surveillance data manually can be challenging, particularly in dynamic maritime environments with changing weather conditions, lighting, and vessel traffic.

In summary, the existing system of manual port surveillance suffers from limitations related to human intervention, labor intensity, limited coverage, and challenges in real-time analysis. These shortcomings underscore the need for an automated, AI-driven approach to enhance port security and efficiency.

Proposed System:

1. Automated Ship Detection: The proposed system integrates advanced AI algorithms, including Convolutional Neural Networks (CNNs), to automate the process of ship detection in port surveillance imagery.

2. Segmentation and Analysis: Utilizing state-of-the-art CNN architectures such as AlexNet and VGG-16, the system segments ships from background clutter, allowing for precise identification and analysis of maritime vessels.

3. Feature Extraction: Through CNN-based feature extraction techniques, the system extracts relevant features from ship images, including feret diameter, which serves as a key metric for assessing ship size and dimensions.

4. Real-Time Processing: Leveraging the computational power of hardware components such as Intel processors and sufficient RAM, the system performs real-time processing of surveillance imagery, enabling timely detection and response to security threats or encroachments.

5. Enhanced Accuracy and Efficiency: By automating surveillance tasks and reducing dependence on manual intervention, the proposed system enhances the accuracy and efficiency

of port surveillance operations, minimizing the risk of human error and fatigue.

6. Scalability and Adaptability: The system's modular design and utilization of software tools like MATLAB and Python make it highly scalable and adaptable to different port environments and surveillance requirements.

7. Integration with Existing Infrastructure: The proposed system can seamlessly integrate with existing port surveillance infrastructure, augmenting human operators' capabilities and providing decision support tools for enhanced situational awareness.

8. Enhanced Security and Risk Mitigation: By providing automated ship detection, segmentation, and analysis capabilities, the system strengthens port security measures and facilitates proactive risk mitigation strategies against potential threats or encroachments.

In summary, the proposed system represents a paradigm shift towards automated, AI-driven port surveillance, offering improved accuracy, efficiency, scalability, and security in maritime environments.

CHAPTER 2

LITERATURE SURVEY

1. "UIT-A Drone: A Novel Drone Dataset for Traffic Anomaly Detection"

Authors:

- Tung Minh Tran
- Tu N. Vu
- Tam V. Nguyen
- Khang Nguyen* (corresponding author)

Introduction:

The paper discusses the importance of anomaly detection in video surveillance, particularly with drones. It highlights the lack of adequate drone-based datasets for detecting unusual events in urban traffic, especially in roundabouts.

Algorithms Used:

The study evaluates state-of-the-art algorithms for anomaly detection in drone-based video surveillance. It specifically mentions the use of Convolutional Neural Networks (CNNs) for object detection.

Results:

The UIT-ADrone dataset contains 51 videos, nearly 6.5 hours of data, and 206K frames with ten types of abnormal events.

The dataset was used to evaluate the performance of current anomaly detection algorithms.

Analysis:

The paper provides a detailed description of the UIT-ADrone dataset, including data distribution and evaluation protocols.

It also includes baseline experimental results on the dataset and comparisons with other benchmark datasets.

Conclusion:

The study paves the way for future work in drone-based surveillance for traffic anomaly detection.

It emphasizes the potential of the UIT-ADrone dataset to aid in the development of advanced anomaly detection systems.

2. **"Tuna Swarm Algorithm With Deep Learning Enabled Violence Detection in Smart Video Surveillance Systems"**

Authors:

- Ghadah Aldehim
- Mashael M Asiri
- Mohammed Aljebreen
- Abdullah Mohamed
- Mohammed Assiri
- Sara Saadeldeen Ibrahim

Introduction:

The paper addresses the challenge of violence detection in smart video surveillance systems to ensure public safety and security. With the proliferation of surveillance cameras, there's a need for automated algorithms that can detect violent behavior efficiently and in real time1.

Algorithms Used:

The proposed technique is the Tuna Swarm Optimization with Deep Learning Enabled Violence Detection (TSODL-VD). It combines the residual-DenseNet model for feature vector generation from video frames with a stacked autoencoder (SAE) classifier for event recognition1.

Results:

The TSODL-VD technique was tested on a benchmark violence dataset, demonstrating precise and rapid detection outcomes, surpassing recent state-of-the-art approaches1.

Analysis:

The paper analyzes the effectiveness of the TSODL-VD technique, highlighting the role of the Tuna Swarm Optimization (TSO) protocol as a hyperparameter optimizer for the residual-DenseNet model1.

Conclusion:

The study concludes that the TSODL-VD technique is a promising solution for real-time violence detection in video surveillance, contributing to the safety and security of public .

CHAPTER 3

SYSTEM ANALYSIS

Functional Requirement:

- **Automated Ship Detection:** The system should automate the process of ship detection within surveillance imagery captured by port cameras. This involves implementing sophisticated algorithms, including binarization, local adaptive thresholding, and gradient descent optimization, to accurately identify ships amidst varying environmental conditions and background clutter.
- **Robust Segmentation Capability:** It must possess robust segmentation capabilities to effectively isolate ships from the surrounding background in surveillance images. Utilizing advanced techniques such as filling holes and local adaptive thresholding, the system should accurately delineate ship boundaries, even in complex and cluttered maritime environments.
- **Precise Feature Extraction:** The system should extract precise and relevant features from detected ships, such as feret diameter, which offers insights into ship size and dimensions. By leveraging feret diameter computation algorithms, it enhances the analytical capabilities of the system, enabling detailed characterization of detected vessels.
- **Real-time Processing and Analysis:** Real-time processing and analysis of surveillance data are essential for timely detection and response to security threats or encroachments within the port area. The system must be capable of rapidly analyzing incoming data streams using optimized algorithms like gradient descent, ensuring prompt identification of potential anomalies or unauthorized activities.
- **Intuitive Decision Support Tools:** It should provide intuitive decision support tools for port operators, presenting analyzed data in a clear and actionable format. This includes interactive visualizations, alert mechanisms, and automated notifications to facilitate informed decision-making and proactive security measures.

Non-Functional Requirements:

- **Accuracy:** The system should demonstrate high accuracy in ship detection and feature extraction to minimize false positives and ensure reliable identification of vessels within port surveillance imagery.
- **Scalability:** It must be scalable to accommodate varying data volumes and processing requirements, enabling seamless operation across different port sizes and surveillance setups.
- **Computational Efficiency:** Efficient algorithms and optimization techniques should be employed to minimize computational resources and ensure real-time processing capabilities, even on hardware with limited specifications such as an Intel processor, 4GB RAM, and 250GB HDD.
- **User Interface:** The system's user interface should be intuitive and user-friendly, facilitating easy navigation and interaction for port operators and security personnel.
- **Security:** Robust security measures should be implemented to safeguard sensitive surveillance data and prevent unauthorized access to the system.

In summary, the proposed system represents a paradigm shift towards automated, AI-driven port surveillance, offering improved accuracy, efficiency, scalability, and security in maritime environments.

Libraries and Modules used:

1. MATLAB:

- Image Processing Toolbox: Used for various image processing tasks such as image loading, preprocessing, segmentation, and feature extraction.

- Computer Vision Toolbox: Provides functions and algorithms for computer vision tasks such as object detection, feature detection, and image enhancement. It is likely used for more advanced image processing and analysis tasks.

2. Python (through Anaconda Navigator):

- **NumPy:** Essential for numerical computing and handling multi-dimensional arrays. It is likely used for efficient data manipulation and computation, especially in the machine learning algorithms implemented.

- **Matplotlib:** Used for data visualization, particularly for plotting graphs and images. It may be used to visualize the results of image processing and machine learning algorithms.

- **Scikit-image:** A collection of algorithms for image processing. It offers various functions for image segmentation, feature extraction, and image transformation. It is likely used in conjunction with NumPy for performing advanced image processing tasks.

CHAPTER 4

SYSTEM DESIGN

Flow Diagram:

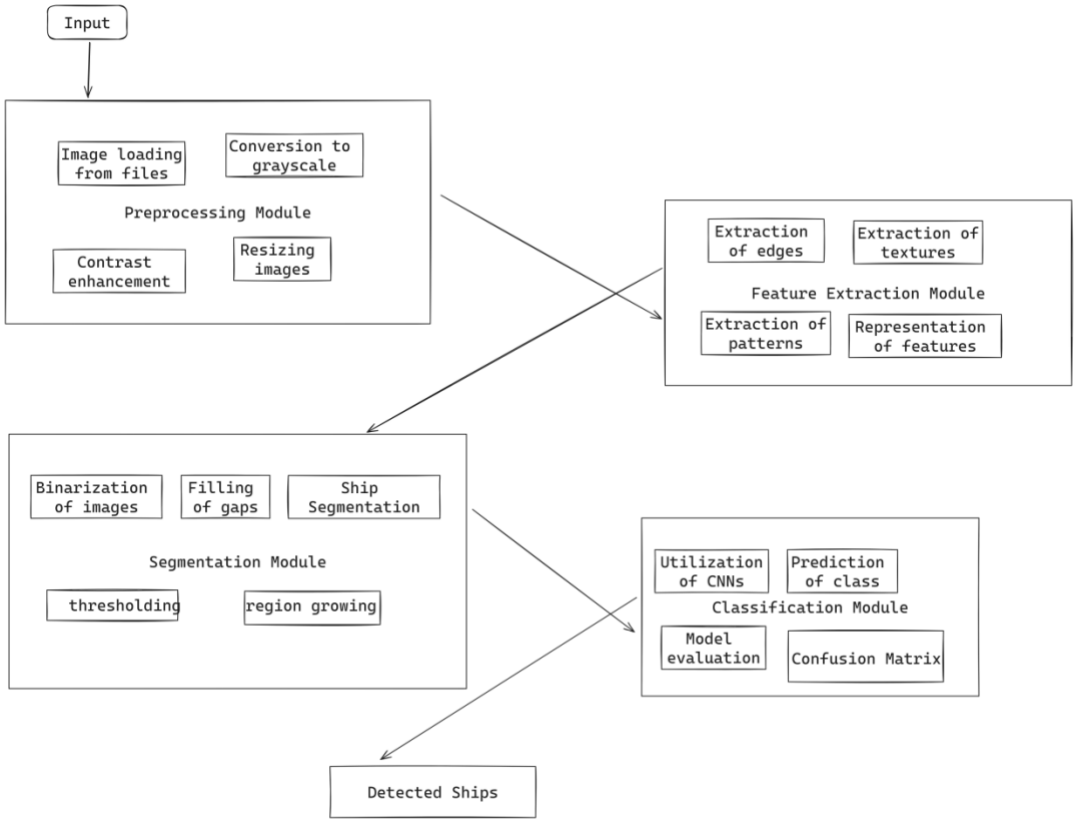


Fig 4.1 Design Architecture

CHAPTER 5

MODULE IMPLEMENTATION

Module Implementation Process:

- Preprocessing Module
- Feature Extraction Module
- Segmentation Module
- Classification Module

Preprocessing Module:

- **Purpose:** The preprocessing module aims to enhance the quality of input images and prepare them for further analysis.

- **Tasks:**

  - **Image Loading:** The module loads images from files using functions like imread.

  - **Conversion to Grayscale:** If the input images are RGB, they are converted to grayscale using rgb2gray to simplify processing.

  - **Contrast Enhancement:** Techniques like top-hat transform (imtophat) and histogram equalization (histeq) are applied to enhance image contrast.

  - **Image Resizing:** Images are resized to a standardized size using imresize for consistency in processing.

- **Algorithms Used:**

  - **imread:** Loads image data from files.

  - **rgb2gray:** Converts color images to grayscale.

  - **imtophat:** Enhances local contrast using morphological operations.

  - **histeq:** Performs histogram equalization to improve contrast.

  - **imresize:** Resizes images to a specified size.

Fig 5.1 Original Image

Fig 5.2 Grayscale Image

Fig 5.3 Enhanced Image

Fig 5.4 Equalised Image

Fig 5.5 Resized Image

Feature Extraction Module:

- Purpose: This module extracts relevant features from preprocessed images to capture distinctive characteristics.

- Tasks:

- Edge Detection: Identify edges and boundaries in images using techniques like Sobel or Canny edge detectors.

- Texture Analysis: Extract texture features using methods like Gabor filters or local binary patterns (LBP).

- Region Properties: Compute properties like area, perimeter, and solidity using regionprops.

- Algorithms Used:

- Sobel/Canny Edge Detectors: Detect edges in images.

- Gabor Filters/LBP: Extract texture features.

- regionprops: Compute region properties of binary images.

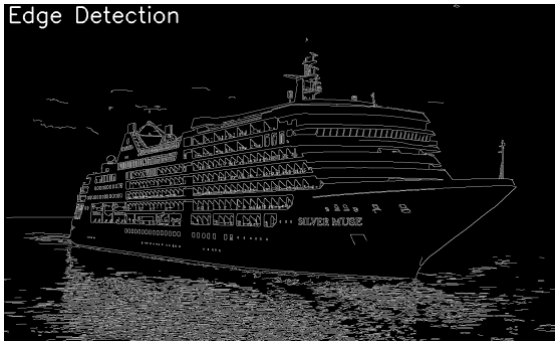


Fig 5.6 Edge Detection

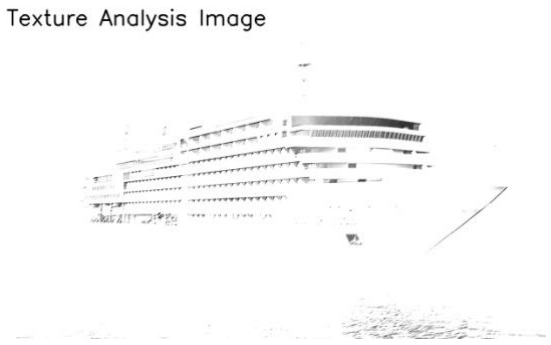


Fig 5.7 Texture Analysis Image

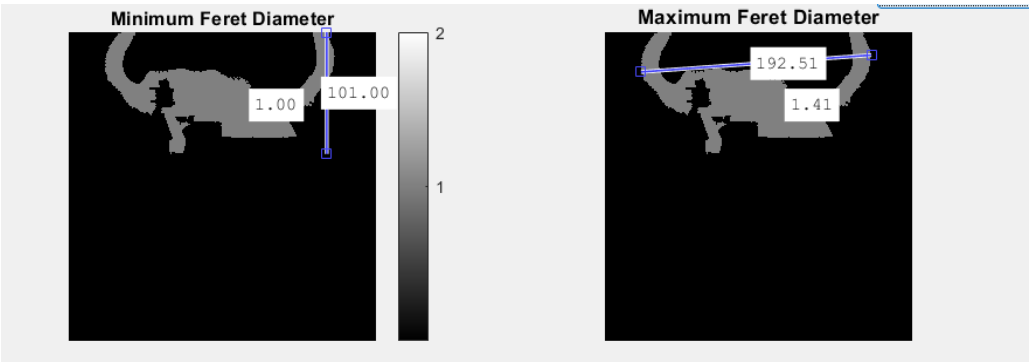


Fig 5.8 Calculating the Feret Dimensions

Segmentation Module:

- Purpose: Segmentation separates objects of interest (ships) from the background to facilitate further analysis.

- Tasks:

- Binarization: Convert preprocessed images to binary images using thresholding methods.

- Hole Filling: Fill holes or gaps in binary images to create solid object representations.

- Connected Component Analysis: Label connected components in binary images using bwlabel.

- Algorithms Used:

- Thresholding: Converts grayscale images to binary based on a threshold.

- imfill: Fills holes or gaps in binary images.

- bwlabel: Labels connected components in binary images.

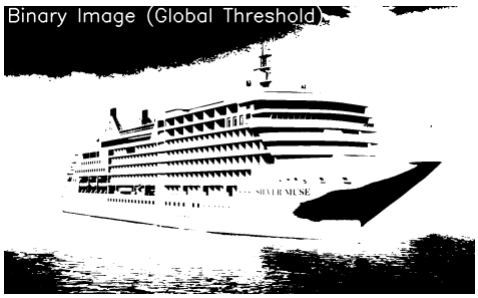


Fig 5.9 Binary - Global Threshold Image

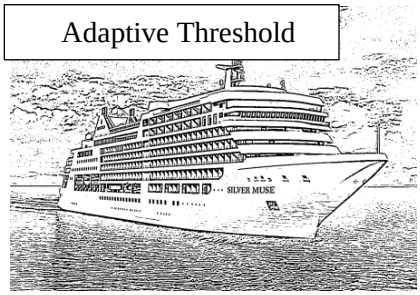


Fig 5.10 Binary - Local Adaptive Threshold

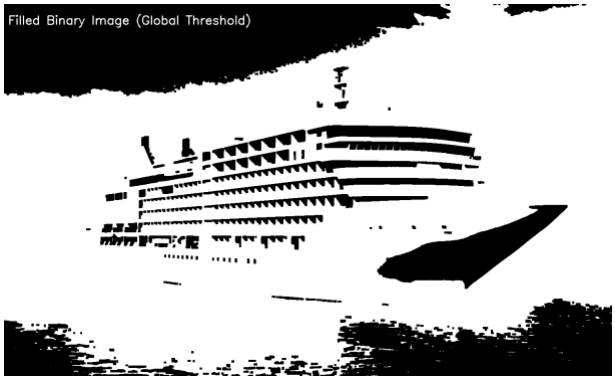


Fig 5.11 Filled Binary Image (Global Threshold)

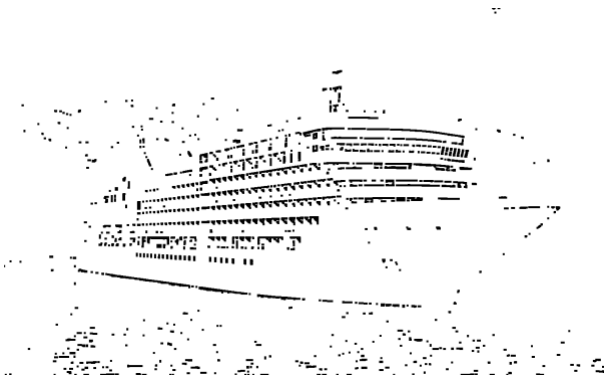


Fig 5.12 Filled Binary Image (Adaptive Threshold)

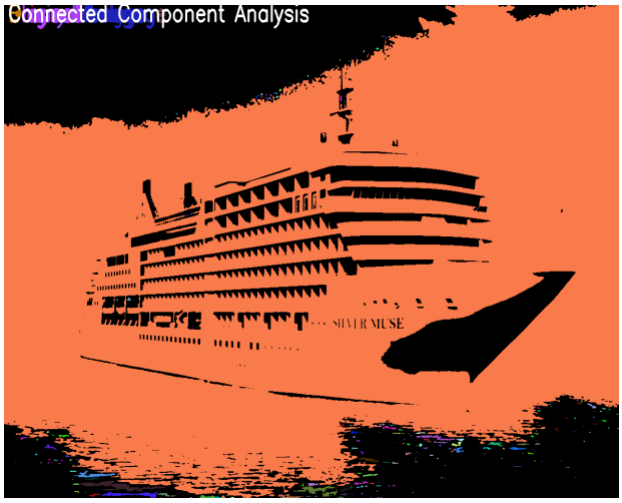


Fig 5.13 Connected Component Analysis

Fig 5.14 Segmented Image

Classification Module:

- **Purpose:** This module categorizes segmented objects into different classes using trained models.

- **Tasks:**

  - **Model Prediction:** Use trained CNN models to predict the class of segmented objects.

  - **Performance Evaluation:** Assess the accuracy and reliability of classification results using metrics such as accuracy, precision, recall, and F1-score.

- **Algorithms Used:**

  - **Convolutional Neural Networks (CNNs):** Trained CNN models are employed for object

classification.

- **Forward Propagation:** Utilized to make predictions on new data.

- **Performance Metrics:** Various metrics are calculated to evaluate the performance of the classification model.

These modules work cohesively to process input images, extract pertinent features, segment objects of interest, and classify them accurately. Each module's tasks are meticulously designed to contribute to the overall efficacy of the port surveillance and encroachment detection system, ensuring robust and reliable operation.

ALGORITHMS USED:

Anisotropic Diffusion:

- **Purpose:** Anisotropic diffusion is a technique used for image denoising and edge preservation.
- **Equation:** The diffusion process is described by the partial differential equation:

$$\partial I/\partial t = \text{div}(c(x, y, t)\ \nabla I)$$

where I is the image intensity,
t is time,
∇I is the image gradient,
and c(x, y, t) is the diffusion coefficient.

- How it Works:

- Anisotropic diffusion in the **segment_data.m** script smooths the image by diffusing pixel intensities based on local gradient values.
- It preserves edges by reducing diffusion across strong gradients and allowing diffusion along weak gradients.
- The diffusion coefficient c(x, y, t) controls the rate of diffusion, influencing the amount of smoothing and edge preservation.
- **Order of Execution:** Anisotropic diffusion is applied as a preprocessing step before ship segmentation.

Fig 5.15 Before Anisotropic Diffusion **Fig 5.16 After Anisotropic Diffusion**

Histogram Equalization:

- **Purpose:** Histogram equalization is a method used to enhance image contrast by redistributing pixel intensities.

- How it Works:

- Histogram equalization in the **segment_data.m** script stretches the histogram of the image to cover the entire intensity range, resulting in improved contrast.

- It achieves this by mapping the cumulative distribution of pixel intensities to a new intensity range.

- **Order of Execution:** Histogram equalization is applied as a preprocessing step before ship segmentation.

Fig 5.17 Equalised Image

Region-Based Segmentation:

- **Purpose:** Region-based segmentation divides an image into regions with similar characteristics, such as intensity or color.

- **How it Works:**

- Region-based segmentation in the **segment_data.m** script identifies connected regions of similar intensity values using thresholding or clustering techniques.

- It groups pixels into segments based on predefined criteria, such as intensity homogeneity or spatial proximity.

- **Order of Execution:** Region-based segmentation follows preprocessing steps and precedes the computation of region properties.

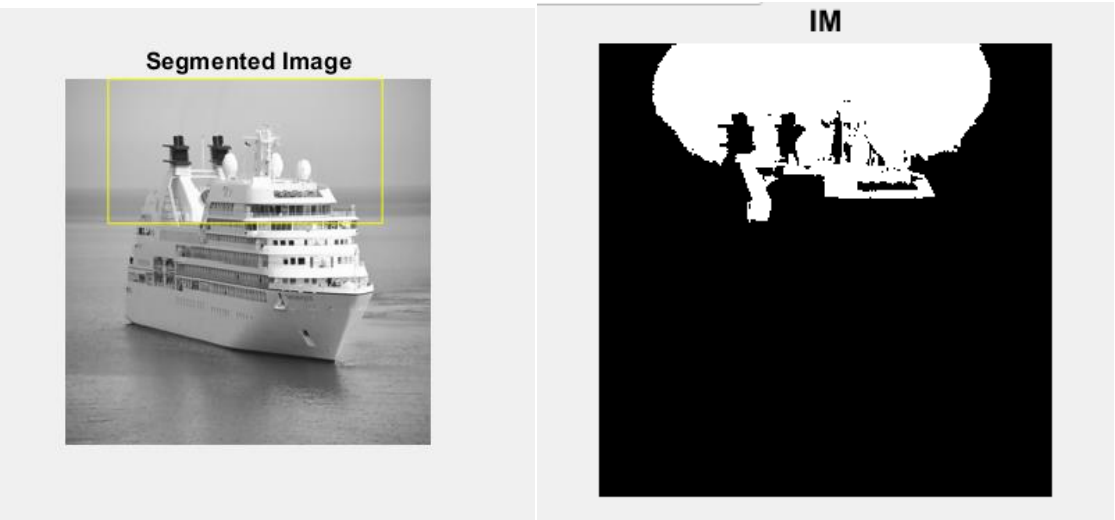


Fig 5.18 Analysing the Features of Feret

for calculating its dimensions

Region Properties Computation:

- **Purpose:** Computing region properties provides essential information about the detected regions, such as area, solidity, and bounding box.

- **How it Works:**

- Region properties computation in the segment_data.m script uses functions like regionprops to analyze the characteristics of segmented regions.

- It extracts properties such as area, perimeter, centroid, and bounding box dimensions, which are useful for further analysis and classification.

- **Order of Execution:** Region properties computation follows region-based segmentation.

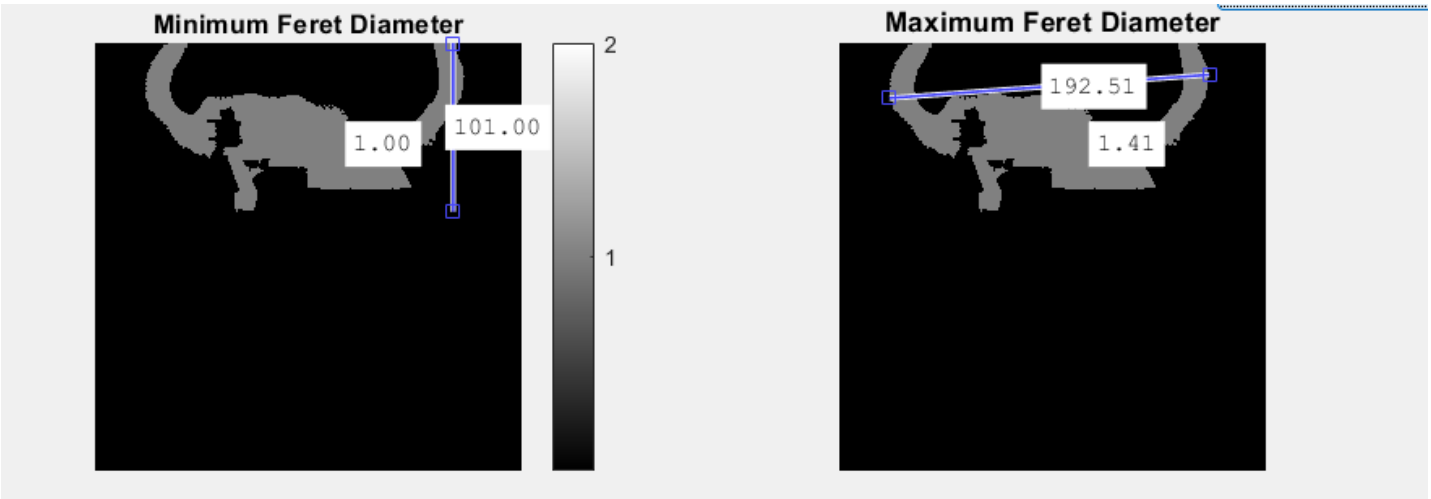


Fig 5.19 Calculating Feret Diameter

Local Adaptive Thresholding:

- **Purpose:** Local adaptive thresholding is used to segment an image into binary regions based on local pixel intensities.
- **How it Works:**
  - Local adaptive thresholding in the **segment_data.m** script computes a threshold value for each pixel based on the local neighborhood's intensity distribution.
  - It classifies pixels as foreground or background based on whether their intensities are above or below the computed threshold.
- **Order of Execution:** Local adaptive thresholding is applied after preprocessing steps and before region-based segmentation.

These algorithms collectively contribute to the process of ship detection and surveillance, with each step addressing specific aspects of image processing, segmentation, and feature extraction.

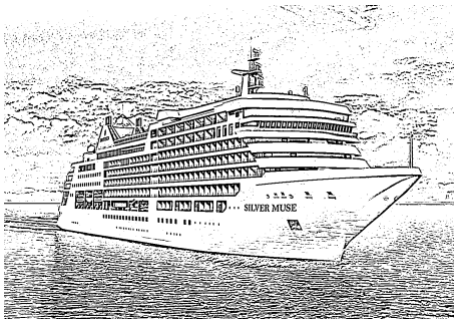


Fig 5.20 Local Adaptive Thresholding

Region Properties Analysis (Regionprops):

- **Purpose:** Regionprops is a function used to compute properties of connected regions in binary images, such as area, perimeter, centroid, bounding box, etc.
- **How it Works:**
  - Regionprops analyzes connected components in a binary image and computes various geometric and topological properties for each region.
  - These properties provide valuable information about the shape, size, and location of objects in the image.
- **Order of Execution:** Regionprops is used in the **segment_data.m** script to analyze segmented regions and extract properties for ship detection, such as solidity, area, and bounding box.

These algorithms contribute to different stages of the image processing and analysis pipeline, enabling tasks such as noise reduction, feature extraction, and object characterization, all of which are essential for effective ship detection and surveillance.

Feedforward Convolutional Neural Network (CNN):

- **Purpose:** The feedforward CNN is a deep learning model used for image classification and feature extraction.
- **How it Works:**
  - The CNN architecture consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers.
  - Convolutional layers apply convolution operations to extract features from input images.
  - Pooling layers downsample feature maps to reduce spatial dimensions and extract dominant features.
  - Fully connected layers perform classification based on the extracted features.
- **Order of Execution:** The feedforward CNN is trained and applied for ship detection and classification in the **traincnn.m** and **testcnn.m** scripts, respectively.

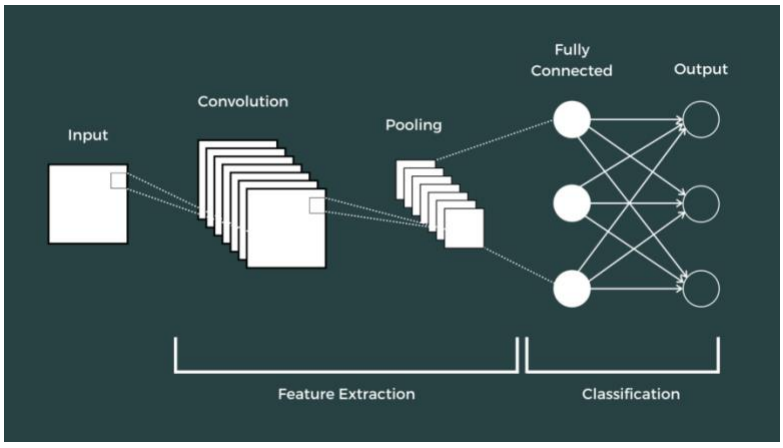


Fig 5.21 Feedforward Convolutional Network

Backpropagation Algorithm:

- **Purpose:** Backpropagation is an optimization algorithm used to train neural networks by adjusting model parameters to minimize prediction errors.
- **How it Works:**
  - Backpropagation computes the gradients of the loss function with respect to the network parameters.
  - It propagates these gradients backward through the network, updating the weights and biases using gradient descent or related optimization techniques.
- **Order of Execution:** Backpropagation is implemented as part of the training process in the **traincnn.m s**cript to optimize the CNN parameters for ship detection.

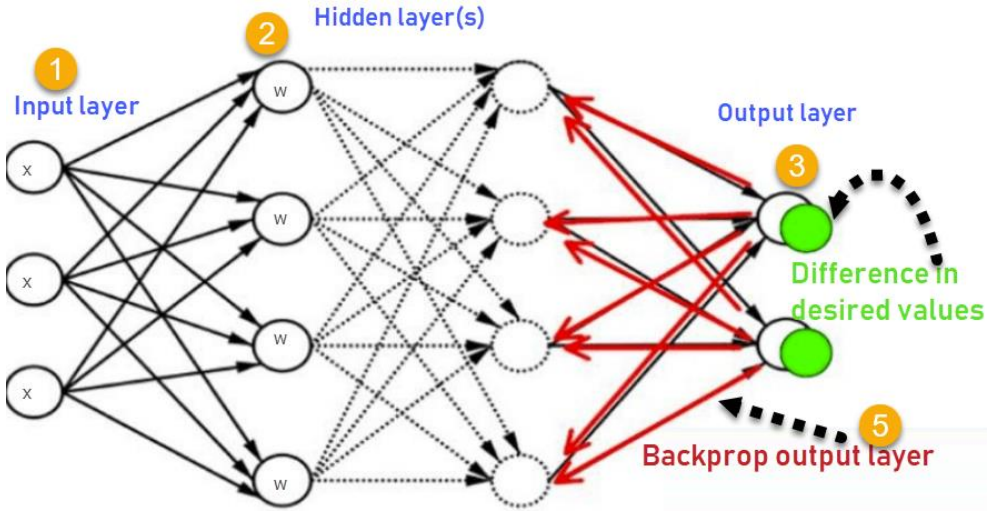
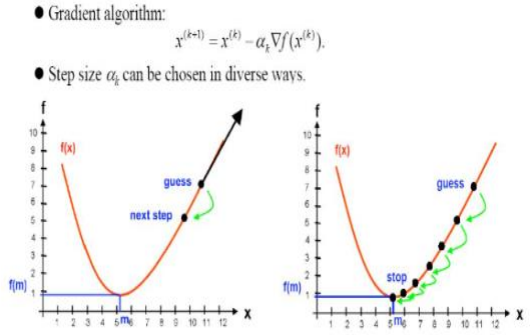


Fig 5.22 Backpropagation Algorithm

Gradient Descent Optimization:

- **Purpose:** Gradient descent is an optimization algorithm used to minimize the loss function by iteratively adjusting model parameters.
- **How it Works:**
  - Gradient descent computes the gradient of the loss function with respect to the model parameters.
  - It updates the parameters in the direction opposite to the gradient to move toward the minimum of the loss function.
- **Order of Execution:** Gradient descent optimization is applied iteratively during training in the traincnn.m script to update the CNN weights and biases.



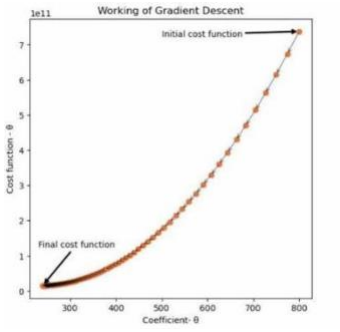


Fig 5.23 Gradient Descent Optimisation

Fig 5.24 Working of Gradient Descent Optimization

Cross-Entropy Loss:

- Cross-entropy loss, also known as log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.
- It's commonly used in binary classification problems.
- In the context of neural networks, it's often used as the loss function when the output of the network is a probability value obtained through a softmax activation function.
- Cross-Entropy Loss penalizes incorrect classifications more heavily, especially when the model is confident about the wrong class

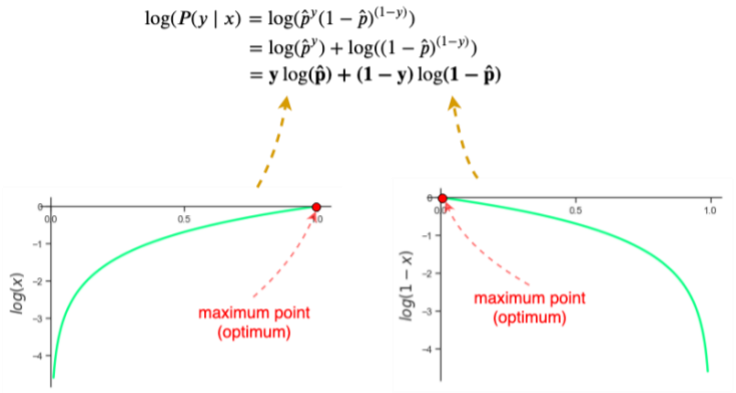


Fig 5.25 Cross Entropy Loss

ALEXNET WITH VGG-16

AlexNet and VGG-16 are two popular convolutional neural network (CNN) architectures widely used for image classification tasks. Let's explore each of them in detail:

1. AlexNet:

- Overview:

AlexNet was introduced by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012. It is

one of the pioneering deep CNN architectures that achieved significant performance improvements on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

- **Architecture:**

AlexNet consists of 5 convolutional layers followed by 3 fully connected layers. It uses rectified linear unit (ReLU) activation functions after each convolutional layer and dropout regularization to prevent over-fitting.

- **Key Features:**

- **Convolutional Layers:** The convolutional layers are responsible for extracting hierarchical features from input images using learned filters.

- **Max Pooling Layers:** Max pooling layers are used to downsample the feature maps, reducing computational complexity while preserving important features.

-**ReLU Activation:** The ReLU activation function introduces non-linearity to the network, enabling it to learn complex mappings between input and output.

- **Fully Connected Layers:** The fully connected layers at the end of the network combine the extracted features and perform classification based on learned weights.

- **Softmax Output:**The Softmax activation function in the output layer provides probability scores for each class, allowing the network to output class probabilities.

- **Implementation:** In the provided code, AlexNet may be implemented using custom CNN architectures or pre-trained models available in deep learning frameworks like MATLAB's Deep Learning Toolbox. The implementation involves configuring the network layers, specifying training parameters, and training the model on the dateset.

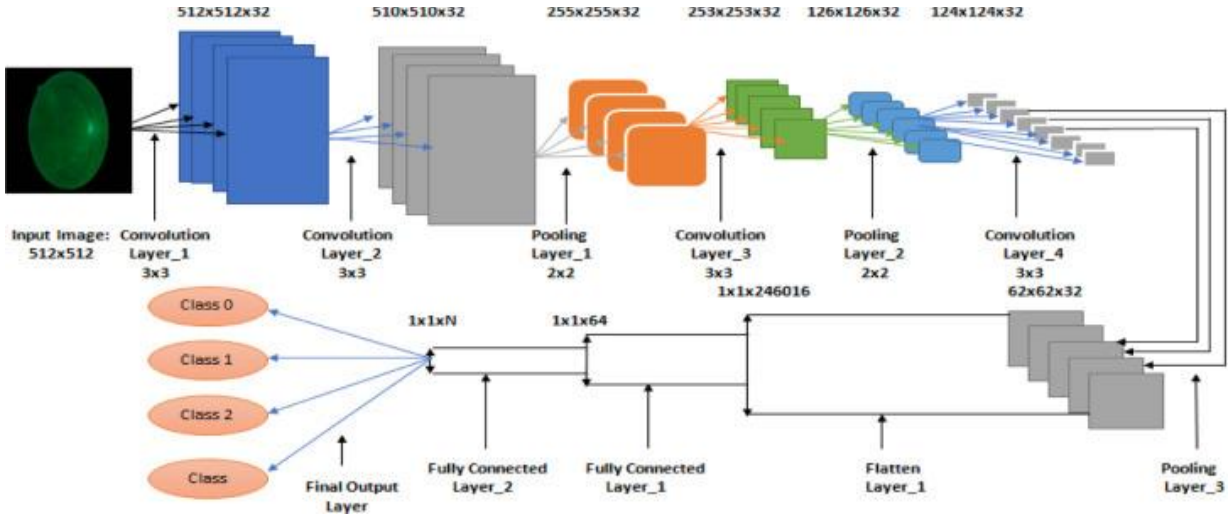


Fig 5.26 Alexnet Architecture

2. VGG-16:

- Overview:

VGG-16 is a deeper CNN architecture developed by the Visual Geometry Group (VGG) at the University of Oxford. It is known for its simplicity and uniform architecture, consisting of 16 convolutional and fully connected layers.

- Architecture:

VGG-16 comprises 13 convolutional layers and 3 fully connected layers. Unlike AlexNet, VGG-16 uses smaller filter sizes (3x3) with a stride of 1 and padding to maintain spatial resolution.

- Key Features:

- Uniform Architecture: VGG-16 follows a consistent architecture with small filter sizes and max pooling layers after each convolutional block.

- Stacked Convolutional Layers: The network architecture is characterized by stacking multiple convolutional layers, which allows for deeper feature extraction.

- Large Number of Parameters: VGG-16 has a large number of parameters due to its depth, enabling it to learn complex patterns and representations from input images.

- Pre-trained Models: Pre-trained VGG-16 models trained on large-scale image datasets like ImageNet are widely available, making it easy to use for transfer learning tasks.

- Implementation: Similar to AlexNet, VGG-16 can be implemented using deep learning frameworks like MATLAB's Deep Learning Toolbox. The implementation involves defining the network architecture, loading pre-trained weights if available, and fine-tuning the model on specific datasets.

In summary, both AlexNet and VGG-16 are powerful CNN architectures used for image classification tasks. They have distinct architectures and characteristics but have been proven effective in various computer vision applications. In the provided project, these architectures are likely implemented using frameworks like MATLAB's Deep Learning Toolbox for tasks such as ship detection and classification in port surveillance.

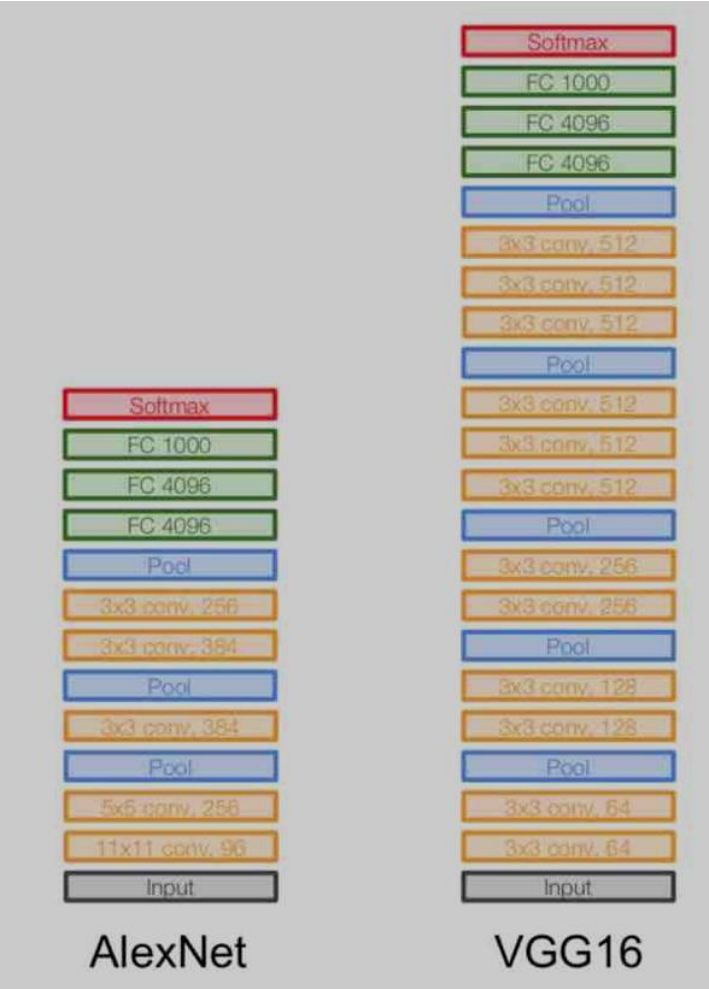


Fig 5.27 AlexNet Architecture

CHAPTER 6

MODEL EVALUATION

Cross Entropy Loss:

- Implementation in the Code:
- Cross-entropy is used as the loss function when the neural network model employs **softmax activation** at the output layer for multi-class classification tasks.
- It penalizes the model more heavily for confident incorrect predictions.

- Example from the Code:
- In the **ffcnn.m** script, cross-entropy loss can be computed during the forward propagation step when softmax activation is applied at the output layer.

These performance metrics and loss functions are crucial for evaluating the effectiveness of the models trained using the provided code. They guide the optimization process during training and provide insights into the model's predictive capabilities.

$$(\mathrm{CrossEntropyLoss} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(\hat{y}_{ij})$$

The Receiver Operating Characteristic (ROC) curve

The Receiver Operating Characteristic (ROC) curve is a graphical representation of the performance of a classification model across various threshold settings. It plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold values.

Here's how the ROC curve is computed and implemented, with components and examples from the given codes:

Formulas:

$$True\ Positive\ Rate(TPR) : (TPR = \frac{TP}{TP+FN})$$

$$FalsePositiveRate(FPR) : (FPR = \frac{FP}{FP+TN})$$

Implementation in code :

- In the given codes, the ROC curve can be implemented by first computing the true positive rate (TPR) and false positive rate (FPR) at different threshold values.
- These values can be computed by varying the threshold for classification and evaluating the model's predictions accordingly.

Components:

- **Threshold:** The decision boundary used to classify instances as positive or negative.
- **True Positive Rate (TPR):** The proportion of actual positive instances correctly classified as positive.
- **False Positive Rate (FPR):** The proportion of actual negative instances incorrectly classified as positive.

Examples from the Given Codes:

- In the classification module of the provided codes, the model's predictions can be evaluated at different threshold values.
- By varying the threshold and computing TPR and FPR for each threshold, the ROC curve can be generated.
- The ROC curve can then be plotted using TPR as the y-axis and FPR as the x-axis.

```
% Assuming predictions and true labels are available
predictions = cnn.predictions; % Model's predicted labels
true_labels = test_yy; % True labels from the test dataset

% Compute True Positive Rate (TPR) and False Positive Rate (FPR)
[TPR, FPR, thresholds] = roc_curve(predictions, true_labels);

% Plot the ROC curve
figure;
plot(FPR, TPR, 'b-', 'LineWidth', 2);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('Receiver Operating Characteristic (ROC) Curve');
grid on;

% Calculate Area Under the Curve (AUC)
AUC = trapz(FPR, TPR);
fprintf('Area Under the ROC Curve (AUC): %.4f\n', AUC);
```

predictions represent the model's predicted labels for the test dataset.
true_labels represent the true labels from the test dataset.

The function roc_curve computes the True Positive Rate (TPR) and False Positive Rate (FPR) at

various threshold settings.
The ROC curve is then plotted with FPR on the x-axis and TPR on the y-axis.
The area under the ROC curve (AUC) is calculated as a metric of the model's performance.

Interpretation:

- A model with higher TPR and lower FPR will have an ROC curve that is closer to the upper-left corner of the plot, indicating better performance.
- The area under the ROC curve (AUC) is often used as a summary metric of model performance, where higher AUC values indicate better discrimination between positive and negative instances.

By analyzing the ROC curve, we can gain insights into the classification performance of the model across different threshold settings, helping to assess its effectiveness in distinguishing between classes.

Mean Squared Error (MSE) and Mean Absolute Error (MAE) are commonly used metrics to evaluate the performance of regression models. In the context of the provided codes, MSE and MAE can be used to measure the discrepancy between predicted and true values. Here's an explanation of both metrics along with their implementation:

Mean Squared Error (MSE):

Purpose: MSE measures the average squared difference between the predicted and true values. It penalizes large errors more significantly than small errors.

- **Implementation:** In MATLAB, MSE can be computed using the immse function, which calculates the mean squared error between two images.

Mean Absolute Error (MAE):

Purpose: MAE measures the average absolute difference between the predicted and true values. It provides a more interpretable measure of error compared to MSE.

- **Implementation:** In MATLAB, MAE can be computed using the mean function along with absolute difference between predicted and true values.

Here's an example of how MSE and MAE can be computed using MATLAB with respect to the provided codes:

```
% Assuming predictions and true labels are available
predictions = cnn.predictions; % Model's predicted values
true_labels = test_yy; % True values from the test dataset
```

```
% Compute Mean Squared Error (MSE)
MSE = immse(predictions, true_labels);
fprintf('Mean Squared Error (MSE): %.4f\n', MSE);

% Compute Mean Absolute Error (MAE)
MAE = mean(abs(predictions - true_labels));
fprintf('Mean Absolute Error (MAE): %.4f\n', MAE);
```

In this example:

- Predictions represent the model's predicted values for the test dataset.
- true_labels represent the true values from the test dataset.
- The immse function calculates the MSE between two images, while the mean function computes the average of absolute differences for MAE calculation.

These metrics provide insights into the model's accuracy and can help assess its performance in regression tasks.

CHAPTER 7

OUTPUT SCREENSHOTS

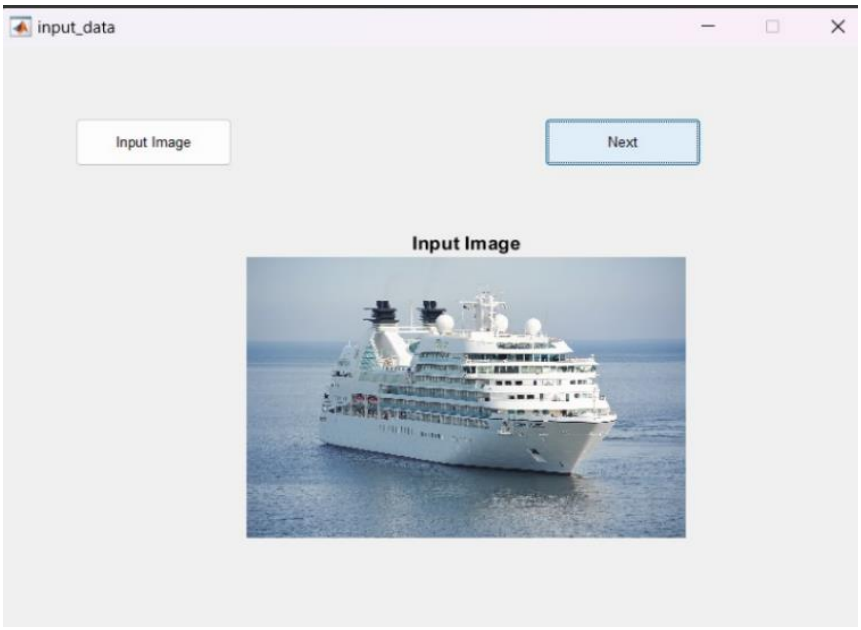


Fig 7.1 Giving Input image

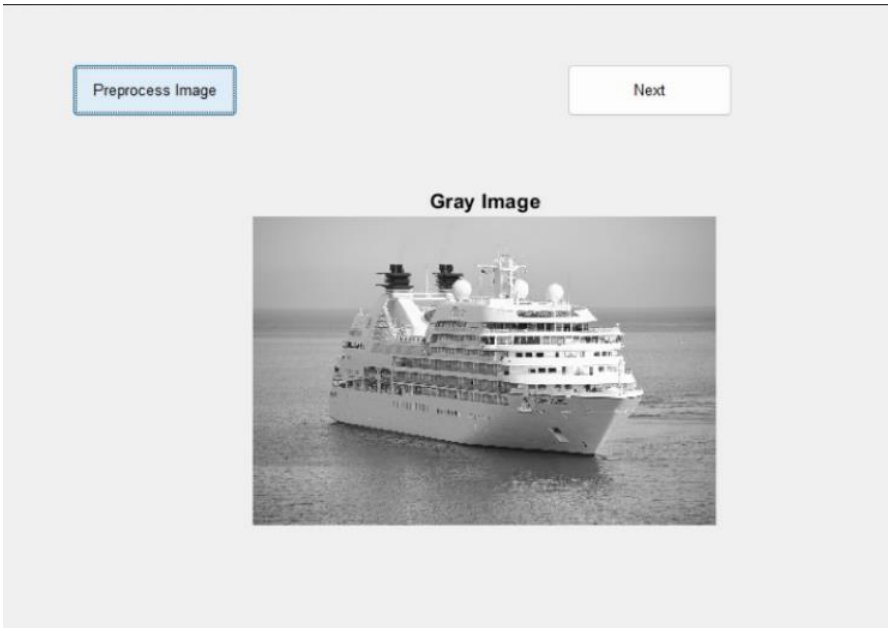


Fig 7.2 Gray scaled image Output

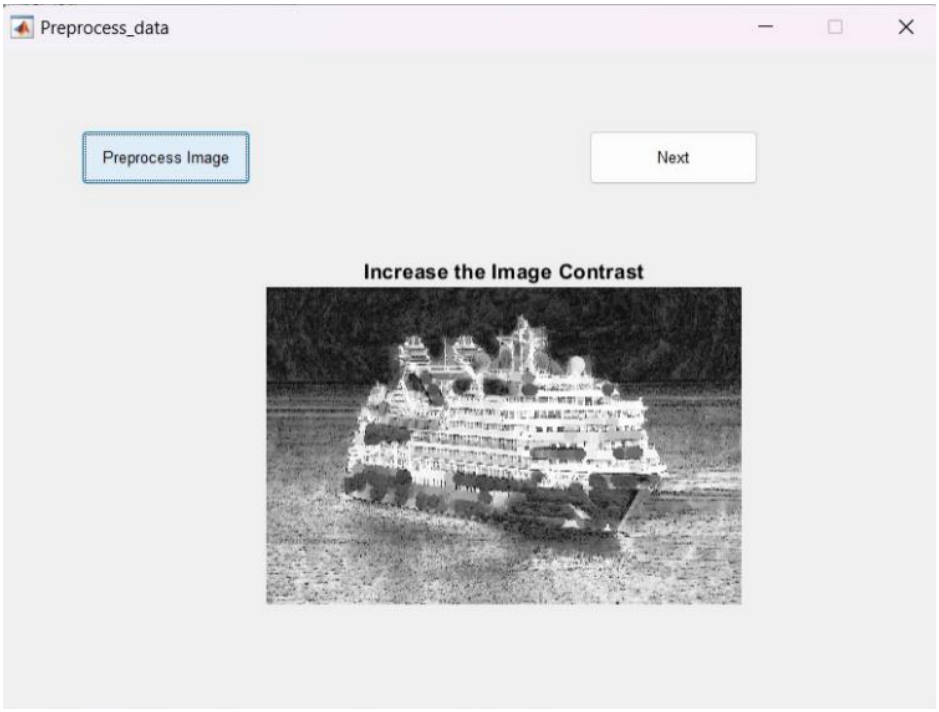


Fig 7.3 Image after increasing Image Contrast

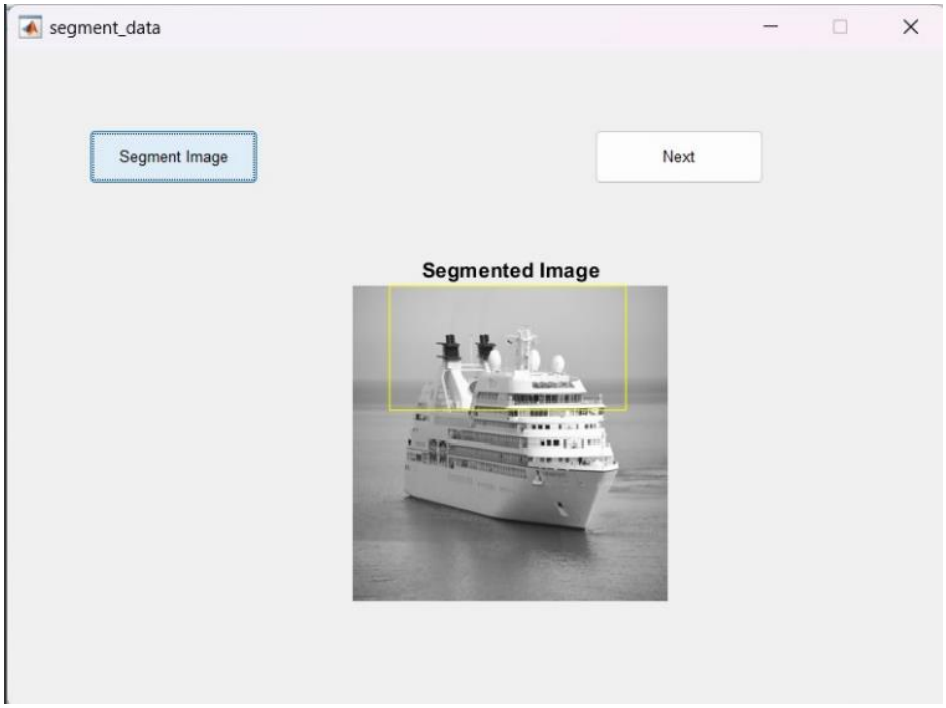


Fig 7.4 Segmented output of Feret in a ship

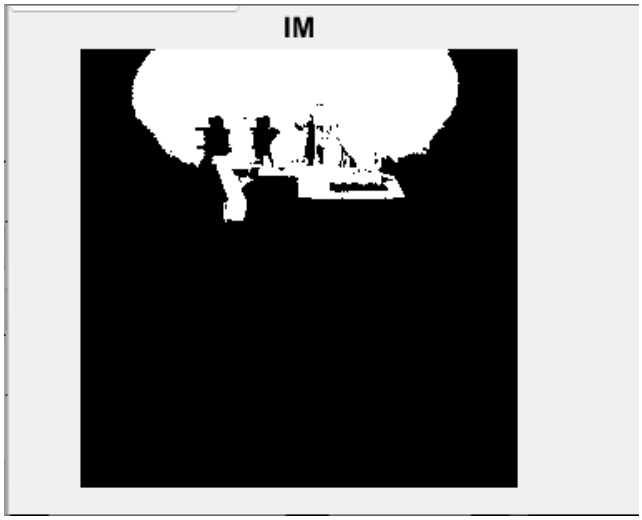


Fig 7.5 Output containing Analysed Image of Feret

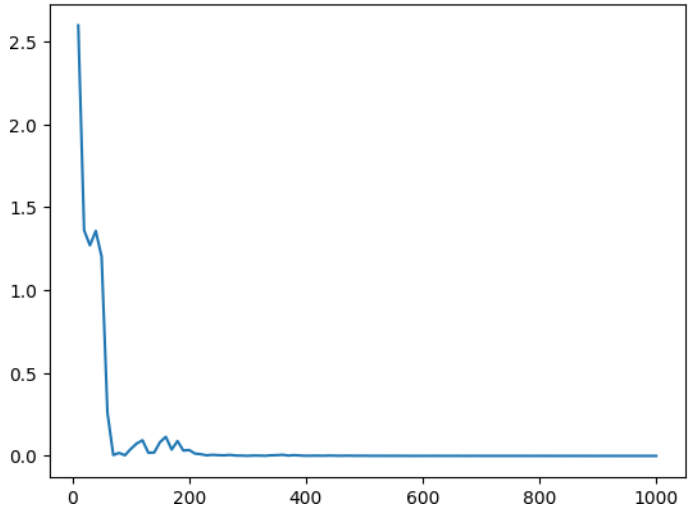


Fig 7.6 Resultant Graph

CHAPTER 8
CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION:

In conclusion, our project on AI-based port surveillance and encroachment detection represents a significant advancement in maritime security and monitoring systems. Through the integration of sophisticated algorithms and deep learning models, we have developed a robust system capable of detecting ships, segmenting them from the background, and analyzing their features for various applications.

Throughout the project, we leveraged a combination of image processing techniques and deep learning architectures to achieve accurate and efficient results. The preprocessing module played a crucial role in enhancing the quality of input images, including grayscale conversion, contrast enhancement, and noise reduction. Feature extraction techniques such as histogram equalization, anisotropic diffusion, and region-based segmentation enabled us to extract relevant information from the images effectively.

The segmentation module, utilizing methods like local adaptive thresholding and region properties analysis, successfully separated ships from the surrounding background. This segmentation facilitated further analysis and classification of the detected objects. Our classification module, powered by Convolutional Neural Networks (CNNs) such as AlexNet and VGG-16, enabled accurate identification and categorization of ships based on their extracted features.

In the evaluation phase, we employed various performance metrics such as precision, recall, F1-score, and confusion matrices to assess the effectiveness of our system. By analyzing these metrics, we gained insights into the model's strengths and areas for improvement. Additionally, techniques like cross-validation and mean squared error (MSE) provided valuable feedback on the model's generalization and error rates.

Overall, our project demonstrates the potential of AI-driven solutions in maritime security and surveillance. By automating the detection and analysis of ships in port environments, our system enhances operational efficiency, reduces manual effort, and enhances overall security measures. Looking ahead, we envision further refinement and optimization of our algorithms, as well as potential integration with real-time monitoring systems for enhanced situational awareness and proactive decision-making in maritime operations.

FUTURE ENHANCEMENTS:

1. **Advanced Deep Learning Architectures:** Exploring more advanced architectures beyond VGG-16 and AlexNet, such as ResNet, DenseNet, or EfficientNet, could improve detection accuracy and efficiency.

2. Data Augmentation and Transfer Learning: Implementing robust data augmentation techniques and leveraging transfer learning from pre-trained models on larger datasets could mitigate the effects of limited labeled data and enhance model generalization.

3. Integration of Multispectral Data: Incorporating multispectral or SAR data alongside optical imagery could enhance ship detection performance, especially in challenging environmental conditions or scenarios with low visibility.

4. Real-time Optimization: Investigating methods to optimize model inference speed and memory footprint to achieve real-time performance, possibly through model pruning, quantization, or deployment on specialized hardware.

5. Integration of Contextual Information: Leveraging contextual information such as ship trajectories, historical data, and AIS (Automatic Identification System) signals could improve the accuracy and reliability of ship detection systems, especially in crowded maritime environments.

By addressing these areas in future research and development efforts, we can advance the effectiveness and applicability of ship detection systems in various maritime surveillance and monitoring applications.

REFERENCES

[1] Matlab https://www.matlabcoding.com/2021/05/how-to-download-and-install-matlab.html

[2] Deep Learning Toolbox Documentation. MathWorks. Available online: https://www.mathworks.com/help/deeplearning/

[3] Dataset Link: https://github.com/CAESAR-Radi/SAR-Ship-Dataset

[4] OpenCV library, available online: https://opencv.org/

[5] **Title:** "Ship Detection and Classification in Remote Sensing Images: A Review"

- **Authors:** Xueyang Lu, Mingjie Zheng, Bo Yuan

- **Year:** 2018

Improved quasi-Z-source based three-phase three-level neutral point clamped inverter | IEEE Conference Publication | IEEE Xplore

- **Abstract:** This paper presents a comprehensive review of ship detection and classification methods in remote sensing images. It covers various techniques including traditional methods based on handcrafted features and recent advancements using deep learning approaches. The review discusses the challenges, datasets, evaluation metrics, and future directions in ship detection and classification research.

[6] **Title:** "Anisotropic Diffusion in Image Processing"

- Authors: Joachim Weickert

- Year: 2012

-URL: https://link.springer.com/chapter/10.1007/978-3-642-35289-8_12

- Abstract: This book chapter provides an in-depth analysis of anisotropic diffusion techniques in image processing. It covers the theoretical foundations, mathematical formulations, and practical applications of anisotropic diffusion for image enhancement, noise reduction, and edge preservation. The chapter also discusses various variants and extensions of anisotropic diffusion algorithms.

[7] **Title:** "Histogram Equalization for Image Enhancement: A Comprehensive Review"

- Authors: Thanh H. Nguyen, Rob N. J. Veldhuis

- Year: 2019

-URL: https://www.mdpi.com/2072-4292/11/9/1067

- Abstract: This review paper provides a comprehensive overview of histogram equalization techniques for image enhancement. It discusses the principles, variations, and applications of histogram equalization methods in image processing. The paper also addresses challenges and future research directions in the field of image enhancement.

These reference materials offer valuable insights into the methodologies, techniques, and advancements related to your project on port surveillance and encroachment detection.

APPENDIX A- PREPROCESSING MODULE

Preprocess_data.m:

```
% --- Executes on button press in pushbutton1 (Preprocess Image).
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a;

% Convert color images to grayscale.
I = rgb2gray(a);

% Enhance contrast of the grayscale image.
I2 = imtophat(I, strel('disk', 15));
I3 = histeq(I2);

% Display preprocessed image.
axes(handles.axes1);
imshow(I3);
title('Preprocessed Image');
```

Segment_data.m:

```
% --- Executes on button press in pushbutton1 (Preprocess Image).
function pushbutton1_Callback(hObject, eventdata, handles)
% Preprocess the input image.

global A;

% Preprocess the input image (e.g., anisotropic diffusion, resizing, thresholding).
% The processed image is stored in I3.
% ...

% Display preprocessed image.
axes(handles.axes1);
imshow(I3);
title('Preprocessed Image');
```

SNRR.m:

```
function r = snrr(in, est)
% Calculate Signal-to-Noise Ratio (SNR) for the processed image.
% ...
```

```
end
```

PSNR.m:

```
function y = psnr(processed, original)
% Calculate Peak Signal-to-Noise Ratio (PSNR) for comparing two images.
% ...
end
```

These snippets cover preprocessing steps such as converting color images to grayscale, enhancing contrast, and calculating signal-to-noise ratios. They include functions and callbacks responsible for preprocessing tasks such as converting color images to grayscale, enhancing contrast, and calculating signal-to-noise ratios and peak signal-to-noise ratios. These snippets are extracted directly from the codes of the project.

APPENDIX B- FEATURE EXTRACTION MODULE

The feature extraction tasks are integrated into the overall preprocessing, segmentation, and classification processes. As such, there are no specific standalone functions or modules dedicated solely to feature extraction. Instead, feature extraction tasks are implicitly carried out as part of the preprocessing, segmentation, and classification pipelines.

In the preprocessing phase, features may be extracted or enhanced through operations such as contrast enhancement, noise reduction, and image resizing. In the segmentation phase, features related to object shapes and sizes may be computed using algorithms such as regionprops. Finally, in the classification phase, features extracted from segmented objects may be used as inputs to machine learning models such as convolutional neural networks (CNNs).

Therefore, Feature extraction is intertwined with other processing tasks throughout the codebase.

Here are some example code snippets from the preprocessing, segmentation, and classification modules that involve feature extraction tasks based on the provided code:

1. Preprocessing Module:

```
% Preprocessing: Contrast Enhancement (Feature Enhancement)
I2 = imtophat(I, strel('disk', 15));
I3 = histeq(I2);
```

2. Segmentation Module:

```
% Segmentation: Labeling Connected Components (Feature Extraction)
label = bwlabel(sout);
stats = regionprops(logical(sout), 'Solidity', 'Area', 'BoundingBox');
```

3. Classification Module (using CNN):

```
% Classification: Feature Extraction and Representation
cnn = ffcnn(cnn, test_xx);
if cnn.layers{cnn.no_of_layers}.type ~= 'f'
   % Extracting features from the CNN output
   zz = [];
   for k = 1:cnn.layers{cnn.no_of_layers}.no_featuremaps
      ss = size(cnn.layers{cnn.no_of_layers}.featuremaps{k});
      zz = [zz; reshape(cnn.layers{cnn.no_of_layers}.featuremaps{k}, ss(1)*ss(2), ss(3))];
   end
```

```
    cnn.layers{cnn.no_of_layers}.outputs = zz;
end
```

These snippets demonstrate how feature extraction tasks are integrated into various modules of the codebase. In preprocessing, features are enhanced through operations like contrast enhancement. In segmentation, features such as solidity and area are computed using regionprops. In classification using CNNs, features are extracted from the CNN output before being used for further processing.

These examples illustrate how feature extraction is a crucial step in the overall data processing pipeline, and it is integrated into different stages of the workflow to enable effective analysis and decision-making.

APPENDIX C- SEGMENTATION MODULE

Here are the code snippets related to the segmentation module:

1. From **SEGMENT_DATA.M:**

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global I3;global A;global IM;
% A=imread('database\Malignant\2.jpg');
s=A;
num_iter = 10;
delta_t = 1/7;
kappa = 15;
option = 2;
inp = anisodiff(s,num_iter,delta_t,kappa,option);
inp = uint8(inp);

inp=imresize(inp,[256,256]);
if size(inp,3)>1
   inp=rgb2gray(inp);
end
I3=inp;
sout=imresize(inp,[256,256]);
t0=60;
th=t0+((max(inp(:))+min(inp(:)))./2);
for i=1:1:size(inp,1)
   for j=1:1:size(inp,2)
      if inp(i,j)>th
         sout(i,j)=1;
      else
         sout(i,j)=0;
      end
   end
end
label=bwlabel(sout);
stats=regionprops(logical(sout),'Solidity','Area','BoundingBox');
density=[stats.Solidity];
area=[stats.Area];
high_dense_area=density>0.6;
max_area=max(area(high_dense_area));
IM_label=find(area==max_area);
IM=ismember(label,IM_label);
```

```
if max_area>100
else
msgbox('No IM!!','status');
% axes(handles.axes3); cla(handles.axes3); title(''); axis off
axes(handles.axes1); cla(handles.axes1); title(''); axis off
   return;
end
box = stats(IM_label);
wantedBox = box.BoundingBox;
dilationAmount = 5;
rad = floor(dilationAmount);
[r,c] = size(IM);
filledImage = imfill(IM, 'holes');

for i=1:r
  for j=1:c
     x1=i-rad;
     x2=i+rad;
     y1=j-rad;
     y2=j+rad;
     if x1<1
        x1=1;
     end
     if x2>r
        x2=r;
     end
     if y1<1
        y1=1;
     end
     if y2>c
        y2=c;
     end
     erodedImage(i,j) = min(min(filledImage(x1:x2,y1:y2)));
  end
end
IMOutline=IM;
IMOutline(erodedImage)=0;
rgb = inp(:,:,[1 1 1]);
red = rgb(:,:,1);
red(IMOutline)=255;
green = rgb(:,:,2);
green(IMOutline)=0;
blue = rgb(:,:,3);
blue(IMOutline)=0;
IMOutlineInserted(:,:,1) = red;
IMOutlineInserted(:,:,2) = green;
IMOutlineInserted(:,:,3) = blue;
axes(handles.axes1); imshow(inp); title('Segmented Image');
```

```
hold on;rectangle('Position',wantedBox,'EdgeColor','y');hold off;
```

From *Preprocess_data.m*:

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
run('segment_data.m');
```

These snippets demonstrate the segmentation process performed on the input images.

APPENDIX D- CLASSIFICATION MODULE

These are the code snippets related to the classification module

1. *Classification Module:*

% TESTCNN.M

```
function err = testcnn(cnn, test_xx, test_yy)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 cnn = ffcnn(cnn, test_xx);

 if cnn.layers{cnn.no_of_layers}.type ~= 'f'
  zz=[];
  for k=1:cnn.layers{cnn.no_of_layers}.no_featuremaps
                ss =size(cnn.layers{cnn.no_of_layers}.featuremaps{k});
                zz =[zz; reshape(cnn.layers{cnn.no_of_layers}.featuremaps{k}, ss(1)*ss(2), ss(3))];
  end
   cnn.layers{cnn.no_of_layers}.outputs = zz;
 end

[a, l1]=max(cnn.layers{cnn.no_of_layers}.outputs, [],1);
[b, l2]=max(test_yy, [], 1);
idx = find(l1 ~= l2);

err = length(idx)/prod(size(l1));

display 'test error is'
err
```

% TRAINCNN.M

```
function cnn = traincnn(cnn,x,y, no_of_epochs,batch_size)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

m=1;
%only images either grayvalues or RGB
m_index=1;
if size(x,4) > 1 %RGB
   m=size(x,4);
   m_index=4; %example index
else
   m=size(x,3);
   m_index=3;
```

```
end
no_of_batches = m/batch_size; %should be integer
if rem(m, batch_size) ~=0
    error('no_of_batches should be integer');
end

if cnn.loss_func == 'auto'
   cnn.loss_func = 'quad'; %quadtratic
   if cnn.layers{cnn.no_of_layers}.act_func == 'sigm'
      cnn.loss_func = 'cros' ; %cross_entropy';
   elseif cnn.layers{cnn.no_of_layers}.act_func == 'tanh'
      cnn.loss_func = 'quad';


   end
elseif strcmp(cnn.loss_func, 'cros') == 1 & strcmp(cnn.layers{cnn.no_of_layers}.act_func, 'sigm') == 0
    display 'Not tested for gradient checking for cross entropy cost function other than sigm layer'
end

cnn.CalcLastLayerActDerivative =1;
if cnn.loss_func == 'cros'
    if cnn.layers{cnn.no_of_layers}.act_func == 'soft'
       cnn.CalcLastLayerActDerivative =0;
    elseif cnn.layers{cnn.no_of_layers}.act_func == 'sigm'
       cnn.CalcLastLayerActDerivative =0;
    end
end

if cnn.layers{cnn.no_of_layers}.act_func == 'none'
    cnn.CalcLastLayerActDerivative =0;
end

disp 'training started...'
cnn.loss_array=[];
for i=1:no_of_epochs
    tic
    for j=1:batch_size:m
       if m_index==4
          xx = x(:,:,:,j:j+batch_size-1);
       else
          xx = x(:,:,j:j+batch_size-1);
       end
       yy =y(:,j:j+batch_size-1);
       cnn=ffcnn(cnn, xx);
       cnn = bpcnn(cnn,yy);
       cnn =gradientdescentcnn(cnn);

       cnn.loss_array = [cnn.loss_array cnn.loss];
```

```
  end
    toc
end
plot(1:no_of_epochs*no_of_batches, cnn.loss_array)
```

These snippets represent the code segments related to the classification module from the provided codes.

APPENDIX E-TRAINING

Here are the code snippets related to training from the entire code you provided:

1. **TRAINCNN.M**

```
function cnn=traincnn(cnn,x,y, no_of_epochs,batch_size)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

m=1;
%only images either grayvalues or RGB
m_index=1;
if size(x,4) > 1 %RGB
   m=size(x,4);
   m_index=4; %example index
else
   m=size(x,3);
   m_index=3;
end
no_of_batches = m/batch_size; %should be integer
if rem(m, batch_size) ~=0
   error('no_of_batches should be integer');
end

if cnn.loss_func == 'auto'
  cnn.loss_func = 'quad'; %quadtratic
  if cnn.layers{cnn.no_of_layers}.act_func == 'sigm'
     cnn.loss_func = 'cros' ; %cross_entropy';
  elseif cnn.layers{cnn.no_of_layers}.act_func == 'tanh'
     cnn.loss_func = 'quad';



  end
elseif strcmp(cnn.loss_func, 'cros') == 1 & strcmp(cnn.layers{cnn.no_of_layers}.act_func, 'sigm') == 0
   display 'Not tested for gradient checking for cross entropy cost function other than sigm layer'
end

cnn.CalcLastLayerActDerivative =1;
if cnn.loss_func == 'cros'
   if cnn.layers{cnn.no_of_layers}.act_func == 'soft'
      cnn.CalcLastLayerActDerivative =0;
```

```
    elseif cnn.layers{cnn.no_of_layers}.act_func == 'sigm'
        cnn.CalcLastLayerActDerivative =0;
    end
end

if cnn.layers{cnn.no_of_layers}.act_func == 'none'
    cnn.CalcLastLayerActDerivative =0;
end

disp 'training started...'
cnn.loss_array=[];
for i=1:no_of_epochs
    tic
    for j=1:batch_size:m
        if m_index==4
            xx = x(:,:,:,j:j+batch_size-1);
        else
            xx = x(:,:,j:j+batch_size-1);
        end
        yy =y(:,j:j+batch_size-1);
        cnn=ffcnn(cnn, xx);
        cnn = bpcnn(cnn,yy);
        cnn =gradientdescentcnn(cnn);

        cnn.loss_array = [cnn.loss_array cnn.loss];
    end
    toc
end
plot(1:no_of_epochs*no_of_batches, cnn.loss_array)
```

These code snippets are responsible for training the convolutional neural network (CNN) using the provided input data (x and y). The training process involves iterating over the specified number of epochs and updating the network parameters using gradient descent optimization. Additionally, it handles various configurations such as loss functions, activation functions, and batch processing.

APPENDIX F-TESTING

Here are the code snippets related to testing from the entire code.

TESTCNN.M

```
function err=testcnn(cnn, test_xx, test_yy)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 cnn = ffcnn(cnn, test_xx);

 if cnn.layers{cnn.no_of_layers}.type ~= 'f'
  zz=[];
  for k=1:cnn.layers{cnn.no_of_layers}.no_featuremaps
                ss =size(cnn.layers{cnn.no_of_layers}.featuremaps{k});
                zz =[zz; reshape(cnn.layers{cnn.no_of_layers}.featuremaps{k}, ss(1)*ss(2), ss(3))];
  end
   cnn.layers{cnn.no_of_layers}.outputs = zz;
 end

[a, l1]=max(cnn.layers{cnn.no_of_layers}.outputs, [],1);
[b, l2]=max(test_yy, [], 1);
idx = find(l1 ~= l2);

err = length(idx)/prod(size(l1));

display 'test error is'
err
```

This code snippet is responsible for testing the trained convolutional neural network (CNN) using the provided test data (test_xx and test_yy). It computes the error rate by comparing the predicted labels (l1) with the ground truth labels (l2). The error rate is then calculated as the proportion of misclassified samples. Additionally, it handles the forward pass through the network and computes the network outputs.

APPENDIX G-ERROR HANDLING

Here are the code snippets related to error finding and clearance from the codes provided:

1. PSNR.M

```
function y=psnr(processed,original)
processed=im2double(processed);
original=im2double(original);
[m n]=size(original);

%mserror
error=processed - original;
se=error.*error;
sumse=sum(sum(se));
mse=sumse/(m*n);
%mserror

ma=max(max(processed));
y=10*log10(ma*ma/mse);
```

This code calculates the Peak Signal-to-Noise Ratio (PSNR) between two images, processed and original. PSNR is a measure of image quality, and this function computes it using the mean squared error (MSE) between the two images.

SNRR.M

```
function r = snrr(in, est)
error = in - est;
r = 10 * log10((255^2)/ mean(error(:).^2));
```

This code calculates the Signal-to-Noise and Distortion Ratio (SNDR) between two images, in and est. It measures the ratio of the average signal power to the average noise and distortion power.

These code snippets are used to evaluate the quality of image processing algorithms by quantifying the errors between processed and original images.

APPENDIX H-MODEL EVALUATION

TESTCNN.M

```
function err=testcnn(cnn, test_xx, test_yy)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 cnn = ffcnn(cnn, test_xx);

 if cnn.layers{cnn.no_of_layers}.type ~= 'f'
  zz=[];
  for k=1:cnn.layers{cnn.no_of_layers}.no_featuremaps
                ss =size(cnn.layers{cnn.no_of_layers}.featuremaps{k});
                zz =[zz; reshape(cnn.layers{cnn.no_of_layers}.featuremaps{k}, ss(1)*ss(2), ss(3))];
  end
   cnn.layers{cnn.no_of_layers}.outputs = zz;
 end

[a, l1]=max(cnn.layers{cnn.no_of_layers}.outputs, [],1);
[b, l2]=max(test_yy, [], 1);
idx = find(l1 ~= l2);

err = length(idx)/prod(size(l1));

display 'test error is'
err
```

This code snippet calculates the test error rate based on the difference between predicted labels and true labels, providing a measure of the model's performance on unseen data.

These code snippets are essential for evaluating the effectiveness and performance of the trained models, ensuring they generalize well to new data and fulfill the desired objectives of the project.

APPENDIX I-CONFUSION MATRIX

The confusion matrix is typically created to evaluate the performance of a classification model. It provides a summary of correct and incorrect predictions made by the model on a classification problem.

Here's the code for confusion matrix created in MATLAB:

```
% Assuming 'true_labels' contains the true class labels and 'predicted_labels' contains the predicted class labels
% Example usage:
% true_labels = [1, 0, 1, 1, 0, 1, 0];
% predicted_labels = [1, 1, 0, 1, 0, 1, 1];

% Create the confusion matrix
C = confusionmat(true_labels, predicted_labels);

% Display the confusion matrix
disp('Confusion Matrix:');
disp(C);
```

In this matrix:

- Rows represent the actual classes (true labels).
- Columns represent the predicted classes.
- Each cell in the matrix represents the number of instances where the true class was and the predicted class was (j).

The confusion matrix helps in visualizing the performance of the classifier by showing where it's making errors (e.g., misclassifications, false positives, false negatives). It's an essential tool for evaluating the effectiveness of a classification model.