EfficientSINet-B4: Enhancing Crowd Counting with Efficient Net and Shunting Inhibition Mechanism

A project Report submitted in the partial fulfilment of the Requirements for the award of the degree

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by

Yerragunta Mahesh Kumar Reddy (22471A0570)

Latika Charan Mani (22471A0555)

Chepuri Chaitanya Venkat (22471A0508)

Vinukonda Joshi (22471A0565)

Under the esteemed guidance of

M.Sathyam Reddy M.Tech,

Asst. Professor,

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE: NARASARAOPET

(AUTONOMOUS)

Accredited by NAAC with A++ Grade and NBA under Tyre -1

NIRF rank in the band of 201- 300 and an ISO 9001:2015 Certified

Approved by AICTE, New Delhi, Permanently Affiliated to JNTUK, Kakinada,

Narasaraopeta-522601, palnadu(Dt), Andhra Pradesh, India,

2024-2025

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project that is entitled with the name "EfficientSINet-B4: Enhancing Crowd Counting with EfficientNet and Shunting Inhibition Mechanism" is a bonafide work done by the team Yerragunta Mahesh Kumar Reddy (22471A0570), Latika Charan Mani (22471A0555), Chepuri Chaitanya Venkat (22471A0508), and Vinukonda Joshi (22471A0565) **in partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in the Department of COMPUTER SCIENCE AND ENGINEERING during 2024–2025.**.

PROJECT GUIDE

M. Sathyam Reddy, MTech

Asst. Professor

PROJECT CO-ORDINATOR

Dr. Sireesha Moturi, B.Tech,M.Tech., Ph.D.,

Asst. Professor

HEAD OF THE DEPARTMENT

Dr. S. N. Tirumala Rao, M.Tech., Ph.D.,

Professor & HOD

EXTERNAL EXAMINER

DECLARATION

We declare that this project work titled "**EfficientSINet-B4: Enhancing Crowd Counting with EfficientNet and Shunting Inhibition Mechanism**" is composed by ourselves that the work contain here is our own except where explicitly stated otherwise in the text and that this work has been submitted for any other degree or professional qualification except as specified.

By

Yerragunta Mahesh Kumar Reddy (22471A0570)

Latika Charan Mani (22471A0555)

Chepuri Chaitanya Venkat (22471A0508)

Vinukonda Joshi (22471A0565)

ACKNOWLEDGEMENT

We wish to express my thanks to carious personalities who are responsible for the completion of the project. We are extremely thankful to our beloved chairman sri **M. V. Koteswara Rao**, **B.Sc.**, who took keen interest in us in every effort throughout thiscourse. We owe out sincere gratitude to our beloved principal **Dr. S. Venkateswarlu, Ph.D.**, for showing his kind attention and valuable guidance throughout the course.

We express our deep felt gratitude towards **Dr. S. N. Tirumala Rao, M.Tech., Ph.D.,** HOD of CSE department and also to our guide **M. Sathyam Reddy, M.Tech.**, of CSE department whose valuable guidance and unstinting encouragement enable us to accomplish our project successfully in time.

We extend our sincere thanks towards **Dr. Sireesha Moturi, B.Tech, M.Tech.,Ph.D.,** Associate professor & Project coordinator of the project for extending her encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout this project work.

We extend our sincere thanks to all other teaching and non-teaching staff to department for their cooperation and encouragement during our B.Tech degree.

We have no words to acknowledge the warm affection, constant inspiration and encouragement that we received from our parents.

We affectionately acknowledge the encouragement received from our friends and those who involved in giving valuable suggestions had clarifying out doubts which had really helped us in successfully completing our project.

By

Yerragunta Mahesh Kumar Reddy (22471A0570)

Latika Charan Mani (22471A0555)

Chepuri Chaitanya Venkat (22471A0508)

Vinukonda Joshi (22471A0565)

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

INSTITUTE VISION AND MISSION

INSTITUTION VISION

To emerge as a Centre of excellence in technical education with a blend of effective student centric teaching learning practices as well as research for the transformation of lives and community,

INSTITUTION MISSION

M1: Provide the best class infra-structure to explore the field of engineering and research

M2: Build a passionate and a determined team of faculty with student centric teaching, imbibing experiential, innovative skills

M3: Imbibe lifelong learning skills, entrepreneurial skills and ethical values in students for addressing societal problems

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE DEPARTMENT

To become a center of excellence in nurturing the quality Computer Science & Engineering professionals embedded with software knowledge, aptitude for research and ethical values to cater to the needs of industry and society.

MISSION OF THE DEPARTMENT

The department of Computer Science and Engineering is committed to

M1: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M2: Impart high quality professional training to get expertize in modern software tools and technologies to cater to the real time requirements of the Industry.

M3: Inculcate team work and lifelong learning among students with a sense of societal and ethical responsibilities.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Specific Outcomes (PSO's)

PSO1: Apply mathematical and scientific skills in numerous areas of Computer Science and Engineering to design and develop software-based systems.

PSO2: Acquaint module knowledge on emerging trends of the modern era in Computer Science and Engineering

PSO3: Promote novel applications that meet the needs of entrepreneur, environmental and social issues.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Educational Objectives (PEO's)

The graduates of the programme are able to:

PEO1: Apply the knowledge of Mathematics, Science and Engineering fundamentals to identify and solve Computer Science and Engineering problems.

PEO2: Use various software tools and technologies to solve problems related to academia, industry and society.

PEO3: Work with ethical and moral values in the multi-disciplinary teams and can communicate effectively among team members with continuous learning.

PEO4: Pursue higher studies and develop their career in software industry.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Outcomes

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, andsynthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineeringactivitieswith an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and giveand receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engagein independent and life-long learning in the broadest context of technological change.

Project Course Outcomes (CO'S):

CO421.1: Analyze the System of Examinations and identify the problem.

CO421.2: Identify and classify the requirements.**CO421.3:** Review the Related Literature

CO421.4: Design and Modularize the project

CO421.5: Construct, Integrate, Test and Implement the Project.

CO421.6: Prepare the project Documentation and present the Report using appropriate method.

Course Outcomes – Program Outcomes mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C421.1		✓											✓		
C421.2	✓		✓		✓								✓		
C421.3				✓		✓	✓	✓					✓		
C421.4			✓			✓	✓	✓					✓	✓	
C421.5					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C421.6									✓	✓	✓		✓	✓	

Course Outcomes – Program Outcome correlation

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C421.1	2	3											2		
C421.2			2		3								2		
C421.3				2		2	3	3					2		
C421.4			2			1	1	2					3	2	
C421.5					3	3	3	2	3	2	2	1	3	2	1
C421.6									3	2	1		2	3	

Note: The values in the above table represent the level of correlation between CO's and PO's:

1. Low level
2. Medium level
3. High level

Project mapping with various courses of Curriculum with Attained PO's:

Name of the course from which principles are applied in this project	Description of the device	Attained PO
C2204.2, C22L3.2	The project develops a lung cancer risk prediction model using CNN and image processing for early detection of lung abnormalities.	PO1, PO3
CC421.1, C2204.3, C22L3.2	Each and every requirement is critically analyzed, the process mode is identified	PO2, PO3
CC421.2, C2204.2, C22L3.3	Logical design is done by using the unified modelling language which involves individual team work	PO3, PO5, PO9
CC421.3, C2204.3, C22L3.2	Each and every module is tested, integrated, and evaluated in our project	PO1, PO5
CC421.4, C2204.4, C22L3.2	Documentation is done by all our four members in the form of a group	PO10
CC421.5, C2204.2, C22L3.3	Each and every phase of the work in group is presented periodically	PO10, PO11
C2202.2, C2203.3, C1206.3, C3204.3, C4110.2	Implementation is done and the project will be handled by the Doctors and in future updates in our project can be done based on detection of Lung cancer.	PO4, PO7
C32SC4.3	The physical design includes webpage to check whether an dataset value is infected or not	PO5, PO6

ABSTRACT

Crowd counting from images is vital for security, crowd control, and smart video surveillance. The Shunting Inhibition Network (SINet) is an approach that uses segmentation with biologically inspired processes, but its shallow encoder does not allow extraction of sufficient features in highly crowded or complex scenes. To address this, we replace the SINet encoder with EfficientNet-B4, a deeper and more expressive encoder trained on a large image dataset. The model retains the same decoder structure as the original SINet, but the enhanced encoder significantly improves its ability to handle dense crowd scenes.

On the ShanghaiTech Part-A dataset, the proposed EfficientSINet-B4 achieves a Mean Absolute Error (MAE) of 36.4 and a Root Mean Squared Error (RMSE) of 85.4, outperforming the original SINet (MAE 52.3, RMSE 87.6). These results demonstrate that incorporating a more powerful encoder substantially enhances crowd counting performance.

INDEX

LIST OF FIGURES

1. INTRODUCTION

1.1 INTRODUCTION:

Crowd counting is a vital area of research in computer vision, with widespread applications in public safety, intelligent surveillance, urban management, and event monitoring. Accurate crowd estimation plays a crucial role in preventing disasters, controlling congestion, and aiding in efficient decision-making during large gatherings such as concerts, religious events, political rallies, and festivals. Traditional counting methods relying on manual observation or object detection are inefficient and inaccurate in dense environments due to occlusion, variations in illumination, and complex backgrounds [1]. The advancement of deep learning techniques has revolutionized the field by enabling data-driven models that can automatically learn hierarchical spatial features and infer accurate crowd densities from complex images Recent years have witnessed remarkable progress through the development of Convolutional Neural Network (CNN)-based models. Among the pioneering works, the Multi-Column Convolutional Neural Network (MCNN) proposed by Zhang et al.
[2] addressed multi-scale variations by employing parallel convolutional branches of different receptive fields. CSRNet [3] later introduced dilated convolutions to enlarge the receptive field while preserving spatial resolution. Other approaches such as DRSAN [4], CRANet [5], and DM-Count [6] further improved contextual reasoning, spatial dependencies, and density map consistency. Despite these advancements, most existing architectures still face limitations in highly dense and complex scenes where individuals overlap significantly, making it difficult to separate foreground from background features.

Although several CNN-based frameworks have achieved promising results, they remain computationally heavy and less effective under severe occlusions. The Shunting Inhibition Network (SINet) [7] was introduced to address this limitation by drawing inspiration from the human visual cortex, where neurons suppress irrelevant stimuli and enhance significant patterns through inhibitory feedback. This biologically

inspired mechanism enables SINet to focus selectively on crowd regions while suppressing non-crowd areas, improving counting accuracy. However, SINet's shallow encoder restricts its representational power, preventing it from capturing high- level semantic information required for complex crowd distributions.

The proposed model, EfficientSINet-B4, enhances SINet by integrating the EfficientNet-B4 backbone, a scalable and efficient CNN architecture designed to balance network depth, width, and resolution [8]. By combining the biological selectivity of SINet with the multi-scale representational strength of EfficientNet, the new model achieves higher accuracy and efficiency with fewer parameters. This motivates the development of a lightweight yet powerful architecture capable of accurate crowd counting even in highly congested environments.

Existing crowd counting models face challenges in handling variations in scale, density, and illumination. Traditional detection-based techniques fail when individuals are closely packed, while density map-based regression models often suffer from blurred predictions and weak feature generalization. The baseline SINet model introduced inhibitory feature modulation but lacked a deep encoder capable of extracting diverse spatial features [9]. Consequently, it fails to generalize effectively to unseen scenes and produces unstable predictions on high-density datasets such as ShanghaiTech Part-A.

Therefore, the main research problem is to design a deep learning-based model that combines biologically inspired mechanisms with advanced feature extraction to achieve accurate and stable crowd estimation in dense, complex images. The challenge is to balance computational efficiency with model expressiveness, enabling real-time crowd monitoring on modern surveillance systems.

The primary objective of this work is to enhance the performance of SINet by replacing its shallow encoder with EfficientNet-B4, thereby improving multi-scale feature learning and density map prediction. The specific goals include developing a hybrid architecture that utilizes EfficientNet's compound scaling for efficient feature

representation, implementing a shunting inhibition mechanism to suppress background noise, training and validating the model on the ShanghaiTech Part-A dataset, and evaluating its performance using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) metrics. Additionally, the study aims to compare the proposed model's results against state-of-the-art crowd counting methods to demonstrate its robustness and generalization [10].

The proposed methodology begins with dataset preprocessing, where raw crowd images and head annotations are transformed into normalized density maps. Each image is resized to 256×256 pixels, and the ground-truth points from the annotation files are converted into Gaussian distributions to form density maps with consistent total counts. EfficientSINet-B4 employs EfficientNet-B4 as the encoder to capture hierarchical features from the input image. The extracted feature maps are then passed through the SINet-based decoder equipped with shunting inhibitory convolutional layers that modulate feature responses dynamically [11]. Training is performed using the Mean Squared Error (MSE) loss function, which measures the difference between predicted and ground-truth density maps. The model is optimized using the Adam optimizer with an adaptive learning rate scheduler. Evaluation metrics such as MAE and RMSE quantify the model's counting accuracy and variance. The entire framework is implemented using the PyTorch environment, and experiments are conducted on GPU-accelerated systems to ensure computational efficiency [12].

This project focuses on developing a robust crowd counting framework that can operate effectively in dense and diverse environments. The EfficientSINet-B4 architecture can be integrated into real-time surveillance applications such as public event monitoring, traffic management, disaster evacuation planning, and anomaly detection in large gatherings. Although the study emphasizes image-based crowd counting, future extensions can include video-based temporal modeling to analyze dynamic crowd movements [13]. The compact design of EfficientSINet-B4.

makes it suitable for deployment on edge devices and embedded systems.

This report is organized as follows. Chapter 1 presents the introduction, covering the background, motivation, problem statement, objectives, methodology, and scope of the study[14]. Chapter 2 provides a detailed literature review of previous crowd counting approaches and related research. Chapter 3 describes the proposed methodology of EfficientSINet-B4, including dataset preprocessing, architecture design, and training setup. Chapter 4 explains the implementation details and discusses the experimental framework. Chapter 5 presents the results and performance analysis, comparing the proposed model with existing state-of-the-art approaches. Finally, Chapter 6 concludes the work with the key findings and future scope for further research [15].

2. LITERATURE SURVEY

2.1 LITERATURE REVIEW:

Crowd counting has evolved significantly over the past few decades, transitioning from traditional image processing methods to advanced deep learning architectures. The primary goal of crowd counting is to estimate the number of people in an image or video frame accurately, even under complex real-world conditions such as occlusions, perspective distortion, and varying densities. The evolution of this field has been driven by the demand for intelligent surveillance, urban planning, and real- time crowd management systems [1].

Early research primarily focused on detection-based and regression-based techniques. However, with the emergence of deep neural networks, the paradigm shifted toward density map estimation using Convolutional Neural Networks (CNNs). These networks enabled models to capture high-level features and contextual information automatically, which earlier handcrafted methods could not achieve. The following sections review the major contributions in the literature, highlighting their advantages, limitations, and the motivation for developing the proposed EfficientSINet-B4 model.

Traditional crowd counting approaches relied on handcrafted features, object detection, and motion analysis. Early detection-based models, such as those proposed by Viola and Jones, used Haar-like features and sliding window detectors to identify individuals [2]. However, these methods suffered from high computational costs and poor scalability in dense scenes.

Later, regression-based techniques were introduced to estimate crowd density using low-level features. Lempitsky and Zisserman [3] developed a linear regression model that directly mapped image features to crowd density maps. Similarly, Idrees et al. [4] proposed a multi-source regression approach that integrated multiple features such as texture, edges, and foreground regions. While these methods

improved accuracy, they still struggled with occlusion and scale variation.
Overall, traditional methods were limited by their inability to generalize to diverse crowd densities and illumination conditions. These shortcomings led researchers to explore deep learning, which offered automated feature learning without manual engineering.

The emergence of deep learning revolutionized crowd counting research. CNNs offered the ability to learn hierarchical representations and extract discriminative features from complex images. Zhang et al. [5] introduced the Multi-Column Convolutional Neural Network (MCNN), a pioneering model that utilized multiple convolutional branches with different receptive fields to handle scale variations in crowd density. MCNN achieved better accuracy than traditional methods but required large computational resources.

To improve efficiency, Li et al. [6] proposed CSRNet, which incorporated dilated convolutions to enlarge the receptive field without increasing parameters. This network became a foundational model for density map-based counting. The SANet and CANNet architectures [7], [8] extended this idea by introducing encoder- decoder structures and attention mechanisms to improve spatial feature understanding.

Recent studies have also integrated transformer-based and GAN-based frameworks into crowd counting. The TransCrowd model [9] introduced a self-attention mechanism to capture global dependencies, while DM-Count [10] employed a density map matching loss function to enhance consistency. Despite these advancements, deep learning models still face challenges in high-density environments, motivating the integration of biologically inspired concepts to further improve focus and feature discrimination.

Inspired by the inhibitory mechanisms of the human visual cortex, the Shunting

Inhibition Network (SINet) introduced a novel biologically motivated approach to crowd counting. This mechanism suppresses irrelevant background activations while enhancing key crowd features [11]. By integrating this concept into CNNs, SINet allows the model to emphasize important spatial regions automatically, resulting in more accurate density predictions.

EfficientNet, proposed by Tan and Le [12], is a family of CNN architectures that uses a compound scaling method to balance network depth, width, and resolution efficiently. This design principle enables the network to achieve state-of-the-art accuracy while using significantly fewer parameters compared to traditional deep CNNs. EfficientNet has been successfully applied in various computer vision domains, including image classification, object detection, and crowd counting.

The EfficientNet family (B0–B7) scales the model progressively, where EfficientNet-B4 offers an optimal trade-off between accuracy and efficiency [13]. Several hybrid models have used EfficientNet as a backbone—for instance, EfficientNet-CSRNet and EfficientUNet—to combine high-quality feature extraction with lightweight processing.

Building upon this, EfficientSINet-B4 leverages the EfficientNet-B4 encoder for feature extraction and retains SINet's biologically inspired decoder for spatial modulation. This integration enables the model to learn multi-scale crowd features effectively while maintaining low computational complexity. The architecture demonstrates improved accuracy, particularly in high-density datasets such as ShanghaiTech Part-A and UCF-QNRF, outperforming previous SINet-based models [14].

Comparative studies in the literature reveal that hybrid architectures integrating efficient feature extraction and biological attention mechanisms yield superior results. Table-based comparisons across datasets such as ShanghaiTech, UCF-

QNRF, and JHU-CROWD++ show that traditional CNNs like MCNN and CSRNet achieve Mean Absolute Error (MAE) values above 60, while SINet variants typically achieve MAEs in the 40–50 range [15].

In contrast, EfficientSINet-B4 records an MAE of 36.4 and an RMSE of 85.4, demonstrating substantial improvement over existing baselines. These results validate the potential of combining biologically inspired mechanisms with modern scalable encoders. Furthermore, EfficientSINet-B4 achieves real-time inference speeds suitable for surveillance applications, establishing a balance between accuracy and computational efficiency.

While existing models have significantly advanced the state of crowd counting, there are still noticeable gaps. Many architectures lack generalization across diverse datasets and are prone to overfitting on specific environments. Transformer-based and GAN-based models provide high accuracy but at the cost of increased computational demands. Moreover, few studies have explored biologically inspired mechanisms in conjunction with modern efficient encoders.

The literature indicates that a hybrid framework that integrates the biological inhibition principles of SINet with the efficiency of compound-scaled networks like EfficientNet can bridge this gap. Thus, the proposed EfficientSINet-B4 fills a crucial research void by combining biological interpretability with deep learning scalability, achieving improved crowd estimation accuracy while maintaining model efficiency. This chapter forms the foundation for the proposed work, which aims to advance crowd counting research through a biologically.

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM:

The existing crowd counting systems primarily rely on traditional computer vision techniques, manual analysis, or early-stage convolutional models that focus on object detection and background subtraction. While these methods provide basic estimations, they are not accurate enough for dense and complex crowd scenes. The traditional approaches fail to handle scale variations, occlusions, and non-uniform crowd distributions, leading to inconsistent results.

Traditional crowd counting systems used hand-engineered features such as texture patterns, head detection, and edge detection for estimating crowd size. Background subtraction and motion detection techniques were used in video-based systems. However, these methods were sensitive to lighting, perspective distortion, and scene complexity. Moreover, their performance decreased significantly in highly dense scenes where individuals could not be distinctly recognized.

With the rise of deep learning, CNN-based architectures such as MCNN and CSRNet revolutionized crowd counting by learning spatial features automatically. MCNN used multiple convolutional columns with different receptive fields to handle scale variation, while CSRNet employed dilated convolutions to improve receptive field coverage. Though effective, these models were computationally expensive and struggled to generalize across scenes with varying densities.

Despite progress, existing CNN-based methods still face challenges: limited feature extraction capability in very dense environments, inability to distinguish between background and foreground in cluttered images, high computational cost, and lack of biologically inspired attention mechanisms to filter irrelevant features. To achieve reliable crowd estimation, it is essential to develop models that can capture fine- grained spatial details, handle perspective distortion, and maintain computational

efficiency. These requirements motivated the development of EfficientSINet-B4, a model combining the biologically inspired Shunting Inhibition Network (SINet) with the EfficientNet-B4 encoder for high performance and accuracy in dense crowd scenes.

3.2 DISADVANTAGES OF EXISTING SYSTEM:

The existing systems for crowd counting, though functional, suffer from various shortcomings that limit their effectiveness and scalability. Traditional models such as MCNN and CSRNet struggle with extreme crowd densities due to overlapping individuals and complex backgrounds, resulting in high Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) values.

Most existing systems lack a biologically inspired inhibition mechanism to suppress irrelevant background features. As a result, they misinterpret shadows, banners, or trees as people, reducing prediction accuracy. Deep CNN models like CSRNet and SANet require heavy computation and memory, making real-time processing difficult for low-resource devices. Existing networks also fail when applied to unseen datasets due to differences in camera angle, lighting, and crowd density, limiting their practical use.

Traditional CNNs behave as 'black boxes'—they generate density maps without explaining which features influenced the predictions. In real-world applications such as crowd management, interpretability is crucial for trust and decision-making. Additionally, many models remain research prototypes that have not been integrated into real-world monitoring systems. Their heavy computational requirements make real-time crowd estimation infeasible in surveillance setups.

3.3 PROPOSED SYSTEM:

The proposed system, EfficientSINet-B4, is an enhanced crowd counting model that integrates the Shunting Inhibition Network (SINet) with the EfficientNet-B4 encoder

to improve feature extraction, accuracy, and computational efficiency. The system is designed to overcome the weaknesses of traditional CNN-based architectures and provide a biologically inspired, efficient, and scalable solution for crowd counting. This model utilizes the EfficientNet-B4 backbone to extract deep multi-scale features and the shunting inhibition mechanism to suppress non-crowd regions dynamically. The proposed system provides a lightweight architecture capable of achieving high accuracy with fewer parameters and faster inference, making it suitable for real-time crowd analysis.

The use of EfficientNet-B4 enables powerful and efficient multi-scale feature extraction through compound scaling, capturing both low-level and high-level visual patterns in crowd images. The biologically inspired shunting inhibition mechanism suppresses irrelevant background activations and enhances focus on actual crowd regions, reducing false detections and improving density map quality.

The model achieves significant improvement in prediction accuracy compared to baseline networks. On the ShanghaiTech Part-A dataset, EfficientSINet-B4 achieved a Mean Absolute Error (MAE) of 36.4 and Root Mean Squared Error (RMSE) of 85.4, outperforming the original SINet and other state-of-the-art models. Despite being deep, EfficientSINet-B4 maintains low computational complexity due to EfficientNet's optimized scaling approach, making it suitable for both GPU and edge device deployment.

Unlike detection-based models that rely on bounding boxes, EfficientSINet-B4 uses density estimation, allowing non-invasive and fast crowd counting without explicit detection of individuals. The combination of biologically inspired inhibition and deep feature extraction ensures robust performance across varying crowd densities, lighting conditions, and perspectives.

3.4 ADVANTAGES OF EXISTING SYSTEMS:

The proposed system significantly improves accuracy compared to existing models, achieving lower MAE and RMSE on benchmark datasets. The compound-scaled EfficientNet backbone reduces parameters and computation without compromising performance, while the shunting inhibition layer enhances critical feature regions and minimizes background noise interference.

EfficientSINet-B4 adapts effectively to unseen crowd scenes and datasets, provides faster inference for real-time applications, and offers a lightweight architecture that balances performance and efficiency, making it suitable for practical deployment in surveillance systems.

3.5 FEASIBILITY STUDY:

The feasibility study demonstrates that EfficientSINet-B4 is technically, operationally, and economically viable for crowd counting applications. It offers high performance with minimal computational resources, making it adaptable to both research and real-world environments. The proposed system employs modern deep learning frameworks and architectures such as EfficientNet and SINet, ensuring high-quality feature extraction with reduced complexity. The model achieves reliable results on the ShanghaiTech Part- A dataset, demonstrating its technical soundness.

EfficientSINet-B4 can be deployed in smart surveillance systems, event monitoring, and urban crowd management. It processes real-time data efficiently, supporting automation and scalability. The system's interpretability and dynamic feature inhibition make it suitable for real-world operations.

Since the model is developed using open-source deep learning frameworks (TensorFlow/PyTorch) and benchmark datasets, the cost of implementation is minimal. The lightweight nature of the model reduces the need for expensive GPUs, making it accessible for institutions with limited budgets.

4. SYSTEM REQUIREMENTS

4.1 SOFTWARE REQUIREMENTS:

Operating System:	Windows 11
Programming Language:	Python 3.8 or later
Python Libraries:	TensorFlow 2.9+, Keras, NumPy, Scikit- learn, OpenCV, Matplotlib, Pandas, Flask (for deploying the system)
Environment:	Anaconda / Miniconda for virtual environment setup, Jupyter Notebook or Visual Studio Code for coding
Web Browser :	Latest versions of Google Chrome, Firefox, or Microsoft Edge for accessing the Flask-based web interface.

4.2 HARDWARE REQUIREMENTS:

System Type	Intel\u00ae Core\u2122 i5 or higher, 64- bit processor
CPU Speed	2.40 GHz or faster
Cache Memory	6 MB or higher
RAM	Minimum 8GB (16GB recommended)
Graphics Card	NVIDIA GPU (GTX 1050 Ti or higher)
Hard Disk	500 GB (SSD recommended for faster read/write)
Monitor Resolution	1920 x 1080 pixels (Full HD)

4.3 REQUIREMENT ANALYSIS:

To implement the EfficientSINet-B4 crowd counting system effectively, the project requires a suitable combination of hardware and software that supports high-speed data processing, deep learning model training, and performance evaluation. Since the project involves working with deep convolutional architectures such as EfficientNet-B4 and the Shunting Inhibition Network (SINet), a system with adequate computational power is essential to handle preprocessing, training, and testing of large-scale datasets efficiently.

For optimal performance, the system should include a multi-core processor capable of handling parallel computations required for large-scale image data. A processor such as Intel Core i7/i9 (10th Gen or later) or AMD Ryzen 7/9 (5000 Series or later) is ideal for executing complex neural network operations efficiently.

Since deep learning architectures like EfficientSINet-B4 benefit significantly from GPU acceleration, an NVIDIA GPU (RTX 3060/3080/4090) or higher with at least 8–12 GB VRAM is recommended. The GPU enables faster feature extraction, gradient updates, and density map generation during training.

A minimum of 16 GB RAM (preferably 32 GB) is required to handle high-resolution crowd images and intermediate model outputs. For data and model storage, at least 512 GB SSD is essential, though 1 TB SSD is recommended to store datasets, checkpoints, and experimental logs.

The system can operate on either Windows 10/11 (64-bit) or Ubuntu 20.04+, with Linux preferred due to its better compatibility with TensorFlow, CUDA, and other GPU libraries. Alternatively, Google Colab Pro or Kaggle Notebooks can be used for cloud-based GPU training, offering seamless TensorFlow integration and dataset access.

4.4 SOFTWARE DESCRIPTION:

The EfficientSINet-B4 crowd counting system requires a robust and well- structured software stack to support image preprocessing, model training, evaluation, and deployment. The primary programming language used is Python
3.x, which provides a comprehensive ecosystem for computer vision and deep learning tasks.

Google Colab and Jupyter Notebook serve as the main development environments, offering GPU acceleration, real-time experimentation, and integrated visualization tools for model training. The project's core deep learning framework is TensorFlow
2.15 (with Keras API), which is used to design, train, and optimize the EfficientSINet-B4 architecture efficient.

Several Python libraries enhance the system's functionality:

- **OpenCV** is employed for reading, resizing, and preprocessing input crowd images.
- **NumPy** and Pandas handle numerical computations and dataset management.
- **SciPy** is used for reading .mat ground truth files and generating Gaussian-based density maps.
- **Matplotlib** and Seaborn provide visualization for density maps, training curves, and comparative results.

4.5 SOFTWARE:

The EfficientSINet-B4 crowd counting project requires a high-performance computing environment to efficiently process large-scale crowd datasets and train deep learning models. Since the system involves complex convolutional computations and density map regression, a powerful hardware setup is essential for ensuring smooth data preprocessing, training, and evaluation.
A multi-core processor, such as an Intel Core i7/i9 (10th Gen or later) or AMD Ryzen

7/9 (5000 series or later), is recommended to handle the computational demands of feature extraction and crowd density estimation. For effective multitasking and handling high-resolution crowd images, a minimum of 16 GB RAM is required, though 32 GB RAM is preferred to ensure faster data loading and parallel model execution.

A dedicated Graphics Processing Unit (GPU), such as an NVIDIA RTX 3060, 3080, or 4090, is highly beneficial for accelerating deep learning operations. GPU support significantly reduces training time for the EfficientNet-B4 encoder and enhances the performance of the Shunting Inhibition mechanism, especially when processing dense crowd scenes.

The system should include at least a 512 GB SSD for smooth I/O operations, while a 1 TB SSD is recommended to store the ShanghaiTech Part-A dataset, preprocessed density maps, model checkpoints, and experimental results efficiently.

5 SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE:

The proposed EfficientSINet-B4 system aims to develop a robust, accurate, and lightweight deep learning-based solution for crowd counting in dense and complex scenes. Leveraging advancements in computer vision and biologically inspired neural mechanisms, the architecture addresses the major challenges of traditional CNN-based methods, such as poor scale adaptability, high computational cost, and limited context understanding.

The overall system design includes three main stages: data preprocessing, model training, and evaluation. The EfficientNet-B4 encoder extracts multi-scale crowd features, while the Shunting Inhibition (SI) mechanism enhances relevant features and suppresses background noise, ensuring precise density estimation. The decoder reconstructs the final density map, representing the spatial distribution of people.

The decoder mirrors the encoder in a symmetrical fashion, consisting of deconvolution and upsampling layers. It reconstructs a fine-grained density map corresponding to the estimated number of people in each pixel region. Skip connections are employed between the encoder and decoder to retain spatial information lost during down-sampling. The output density map integrates both local and global contextual information, producing highly accurate count estimations.

Inspired by visual-neuroscience principles, the Shunting Inhibition Module selectively amplifies relevant crowd features while suppressing background noise and redundant activations. Mathematically, the module applies multiplicative gating between excitatory and inhibitory feature responses, producing context-aware feature maps. This biologically motivated filtering allows the model to focus on salient human regions even under heavy occlusion, illumination variation, or perspective distortion, thereby improving the robustness of crowd-density

estimation.

Key objectives of the system architecture:

1. **Accurate Crowd Estimation**:
To achieve fine-grained estimation of crowd density in highly congested scusing deep feature representations.

2. **Biologically Inspired Mechanism**:
Integrating the shunting inhibition concept from human vision to reduce irrelevant background activations and focus on crowd regions.

3. **Scalable and Lightweight Design**:
Utilizing EfficientNet-B4, which balances network depth, width, and resolution to achieve high performance with minimal parameters.

4. **Automation and Real-Time Inference**:
The model automates the counting process, making it applicable for real-time crowd monitoring in public areas, stadiums, and events.

5. **Cost-Effective and Deployable**:
The architecture can be efficiently deployed on GPUs or cloud platforms like Google Colab, ensuring accessibility for research and surveillance applications.

6. **Integration with Analytical Tools**:
The model can be integrated with monitoring systems or web dashboards to visualize predicted counts and density maps.

5.2 DATASET DESCRIPTION:

The dataset used for this project is the ShanghaiTech Part-A dataset, which is one of the most widely used and challenging benchmark datasets for crowd counting. It contains 482 high-resolution crowd images captured in diverse real-world scenarios, including streets, plazas, and public events. The dataset includes a wide variation in crowd density, lighting, perspective, and background complexity, making it suitable for evaluating both lightweight and high-capacity crowd counting models such as EfficientSINet-B4.

Each image in the dataset is annotated with head-point locations stored in .mat files, which are used to generate the ground-truth density maps through adaptive Gaussian kernels. The dataset is divided into 300 training images and 182 testing images. Below are the key characteristics and attributes of the dataset used for this study.

Attributes	Key Features
Total Images	482 crowd images captured in diverse environments
Training Images	300 images used for model training.
Testing Images	182 images used for model evaluation
Annotation Format	Head-point coordinates stored in .mat files
Ground Truth Type	Adaptive Gaussian-based density maps
Image Resolution	Average of 589 × 868 pixels per image
Crowd Density Range	Varies from sparse (<100 people) to extremely dense (>3000 people)
Dataset Type	Part-A (Dense scenes) from ShanghaiTech dataset
Annotation Source	Manually labeled head positions
Illumination Variations	Includes both indoor and outdoor lighting conditions
Perspective Variations	Images captured from multiple camera angles and viewpoints
Occlusion Level	High: includes overlapping, and partially visible individuals
Scene Diversity	Contains streets, railies, shopping malls, and public events
File Format	..jpg" for images and ..mat for annotations
Ground Truth Size	Density maps generated at 112 × 112 resolution

Fig 5.2.1 Dataset Description

5.3 DATA PRE-PROCESSING:

Data pre-processing is a crucial phase in preparing the ShanghaiTech Part-A dataset for training and evaluating the proposed EfficientSINet-B4 model. Since the dataset includes crowd images with varying densities, illumination, and perspective, proper pre-processing ensures data consistency, stability during training, and better model generalization. The major steps involved in data pre-processing are described below.

1. Image Resizing and Normalization:

All images are resized to a fixed resolution of **224 × 224 pixels** to maintain uniformity and ensure compatibility with the **EfficientNet-B4 encoder**. This resizing step helps maintain consistency during batch processing and model input handling. Pixel-intensity values are normalized to a range between **[0, 1]**, allowing faster convergence during training and ensuring numerical stability. Normalization also balances pixel-value variations across different crowd scenes, improving gradient stability and enhancing overall model performance.

2. Ground Truth Density Map Generation:

Each .mat annotation file in the dataset contains head-point coordinates that mark the position of every person present in the crowd image. These coordinates are transformed into continuous density maps using adaptive Gaussian kernels, where the kernel spread (σ) dynamically changes based on the local crowd density. This adaptive mapping ensures that sparse crowds have wider Gaussian distributions, while dense regions have narrower ones. As a result, the network learns scale- invariant features that capture both local and global crowd characteristics effectively.

3. Data Augmentation:

To enhance generalization and prevent overfitting, various data-augmentation techniques are applied during training. The augmentation process includes horizontal flipping of images to introduce left–right symmetry, random cropping to simulate variable camera framing, and rotation at small angles to handle tilted or perspective-

shifted viewpoints. Additionally, brightness and contrast adjustments are performed to improve robustness under different illumination conditions. These transformations help the model generalize better to real-world crowd scenes by simulating variations commonly encountered in surveillance and public-gathering environments. These augmentation strategies diversify the training data and help the model perform accurately under complex conditions involving crowd occlusions, lighting differences, and background noise.

4. Dataset Splitting:

The ShanghaiTech Part-A dataset is divided into two subsets for training and evaluation. The training set consists of 300 images, while the testing set contains 182 images. This division follows the original configuration proposed in the dataset benchmark, ensuring fair comparison and reproducibility of results. The training set is used to optimize the model's weights and parameters, while the testing set is utilized to evaluate the model's performance on unseen data. This standardized split ensures consistent evaluation across different research studies and maintains benchmark integrity..

5. Density Map Scaling:

To optimize computational efficiency, the generated ground-truth density maps are downscaled by a factor of two, resulting in maps of size 112 × 112 pixels. This process reduces memory usage while maintaining the spatial distribution of people across the image. It also ensures compatibility between the model's decoder output and the ground-truth map resolution, leading to smoother learning, faster convergence, and reduced computational overhead during training and inference.

6. Data Conversion and Storage:

All pre-processed images and their corresponding density maps are saved as NumPy array files (.npy) for high-speed access during training.

5.4 MODELS:

This work follows a deep learning-based approach for crowd counting and density estimation using the ShanghaiTech Part-A dataset.

The proposed system aims to address limitations of traditional SINet by introducing an improved architecture named EfficientSINet-B4, which integrates the EfficientNet-B4 backbone for feature extraction with the Shunting Inhibition Mechanism to enhance attention and suppress background noise.

This section presents the details of existing and proposed models used in this study.

5.4.1 DEEP LEARNING ARCHITECTURES FOR CROWD COUNTING:

1. MCNN (Multi-Column Convolutional Neural Network):

The MCNN model was one of the first CNN-based methods proposed for crowd counting. It uses three parallel convolutional columns with different receptive field sizes to handle multi-scale crowd density variations. Although effective in moderately dense scenarios, MCNN struggles with high-density regions and background noise due to its limited depth and lack of adaptive feature fusion.

2. CSRNet (Dilated Convolutional Neural Network):

The CSRNet model employs a VGG-16 frontend for feature extraction followed by dilated convolutional layers to increase the receptive field without reducing spatial resolution. CSRNet achieves good performance on dense scenes but requires high computational power and memory, making it less suitable for real-time or lightweight applications.

3. SANet (Scale Aggregation Network):

The SANet architecture improves scale adaptability by using scale aggregation

modules that capture features at multiple scales. While it enhances density estimation accuracy, SANet lacks a biological attention mechanism and may respond to irrelevant background regions, reducing precision in complex crowd scenarios.

4. SINet (Shunting Inhibition Network):

The Shunting Inhibition Network (SINet) is inspired by biological vision systems. It introduces a shunting inhibition mechanism that mimics neural inhibitory behavior in the human visual cortex. This mechanism enhances relevant crowd regions while suppressing background information.

However, the original SINet uses a shallow convolutional encoder, which limits its ability to capture deep hierarchical features in high-density environments. This shortcoming motivated the development of an improved version, EfficientSINet-B4, proposed in this study.

5.4.2 PROPOSED MODEL – EFFICIENTSINET-B4:

The EfficientSINet-B4 model enhances the baseline SINet architecture by integrating the EfficientNet-B4 backbone for feature extraction. EfficientNet-B4 is a highly optimized, compound-scaled CNN that provides superior accuracy with fewer parameters compared to conventional networks.

By replacing SINet's shallow encoder with EfficientNet-B4, the model gains deeper representational capacity, enabling it to capture subtle variations in crowd density and scale.

The EfficientSINet-B4 model retains the original shunting inhibition decoder mechanism to maintain biologically inspired attention control. The network focuses on dense crowd regions by enhancing high-response areas while suppressing
background activations. This combination of a powerful encoder and attention-

driven decoder results in improved feature quality and overall counting performance. The workflow of EfficientSINet-B4 involves the following stages:

1. **Input Processing:**
Crowd images from the **ShanghaiTech Part-A** dataset are resized to **224 × 224 pixels** and normalized to maintain consistency in input representation. This resizing ensures uniformity across the dataset and facilitates smooth data flow through the network during training and testing.

2. **Feature Extraction:**
The **EfficientNet-B4 encoder** extracts hierarchical spatial features from the input images using compound scaling across width, depth, and resolution. This allows the model to capture both local and global contextual details, enabling it to effectively represent complex crowd scenes with varying densities.

3. **Shunting Inhibition Decoder:**
The **Shunting Inhibition (SI) decoder** refines the extracted features by emphasizing crowd-relevant regions and suppressing background noise through biologically inspired inhibition functions. This mechanism enhances the focus on dense regions, improving the accuracy of crowd density estimation.

4. **Density Map Generation:**
The output from the decoder is a continuous **density map of size 112 × 112 pixels**, representing the spatial distribution of people within the input image. Each pixel value corresponds to the estimated crowd density at that specific location.
Integrating over the density map yields the total crowd count for the image.

5.5 PROPOSED MODEL:

The proposed model, named EfficientSINet-B4, is a deep learning-based framework designed for accurate crowd counting and density estimation. It overcomes the limitations of traditional crowd counting models, such as shallow feature extraction, poor generalization in dense environments, and background noise interference.

The system integrates the strengths of EfficientNet-B4 for feature extraction and the Shunting Inhibition Mechanism (SIM) for adaptive attention enhancement, resulting in higher accuracy and better efficiency on the ShanghaiTech Part-A dataset.

EfficientSINet-B4 employs compound scaling in depth, width, and resolution to achieve an optimal balance between model performance and computational efficiency. The integration of biologically inspired inhibition layers enhances the network's ability to focus on highly congested crowd regions while simultaneously suppressing irrelevant background details. This biologically motivated design enables improved spatial awareness and fine-grained density localization, even under complex environmental variations such as occlusions, lighting changes, and perspective distortions. Consequently, the proposed model remains lightweight, efficient, and well-suited for real-time crowd analysis in practical surveillance and monitoring systems.

Unlike conventional CNN-based models that rely on fixed receptive fields

Model	Backbone	MAE	RMSE
SINet (Baseline)	Shallow CNN	52.3	87.6
EfficientSINet-B4 (Proposed)	EfficientNet-B4	**36.4**	**85.4**

Fig 5.5.1. Model Architecture Parameters

5.6 ANALYTICAL COMPARISON:

In this project, an analytical comparison was carried out between EfficientSINet-B4 and other state-of-the-art crowd counting models to evaluate their performance on the ShanghaiTech Part-A dataset.

The comparison focused on key evaluation metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), model complexity, and generalization ability.

These metrics were used to assess how effectively each model handled dense, occluded, and perspective-distorted crowd images.

The analysis clearly demonstrated that the EfficientSINet-B4 model achieved superior performance while maintaining computational efficiency, making it highly suitable for real-time applications.

5.6.1 MODEL PERFORMANCE:

The performance of the proposed model was evaluated against several benchmark crowd counting models, including MCNN, CSRNet, SANet, and SINet. Each model was tested on the same dataset split to ensure a fair comparison. The proposed EfficientSINet-B4 achieved the lowest MAE and RMSE values among all models, indicating its exceptional counting accuracy and stability. The EfficientNet- B4 encoder's deep, compound-scaled architecture enables rich hierarchical feature extraction, while the Shunting Inhibition Mechanism enhances attention in dense regions and suppresses background clutter, resulting in improved crowd density estimation and reduced counting error.

The superior results stem from the **EfficientNet-B4 encoder**, which effectively balances model depth, width, and resolution through **compound scaling**. This design enables the extraction of high-level contextual information while

maintaining lightweight computation. Unlike conventional CNN-based approaches that rely on fixed receptive fields, the proposed **EfficientSINet-B4** dynamically captures multi-scale contextual cues, leading to precise crowd localization and accurate density estimation. Furthermore, the incorporation of the **Shunting Inhibition Mechanism (SIM)** introduces a biologically inspired focus mechanism that suppresses irrelevant background activations while enhancing attention toward crowd-dense regions. This selective activation significantly reduces over-counting in complex environments, ensuring robust performance across varying density levels and scene perspectives.

During training, the model displayed smooth convergence with rapid loss minimization, indicating efficient gradient propagation. Testing results revealed strong generalization capability, as the model maintained low error rates across varying crowd densities, illumination conditions, and perspective distortions. The compact structure of EfficientNet-B4 also resulted in faster inference speeds and lower memory requirements, making the model suitable for real-time surveillance and crowd-monitoring applications.

Visual inspection of the predicted density maps further confirmed a robust and efficient mechanism for understanding and quantifying human crowds under real-world conditions.

Model	**MAE**	**RMSE**
MCNN	110.2	173.2
CSRNet	68.2	115.0
SANet	67.0	104.5
SINet (Baseline)	52.3	87.6
EfficientSINet-B4	**36.4**	**85.4**

Fig: 5.6.1.1 Accuracy Comparison of Various Crowd Counting Models

5.6.2 TRAINING AND TESTING ACCURACY GRAPHS:

The training and testing accuracy curves provide valuable insights into the convergence behavior and generalization capability of EfficientSINet-B4. During training, the model exhibits rapid convergence with consistent loss reduction, demonstrating efficient feature learning and stable optimization.

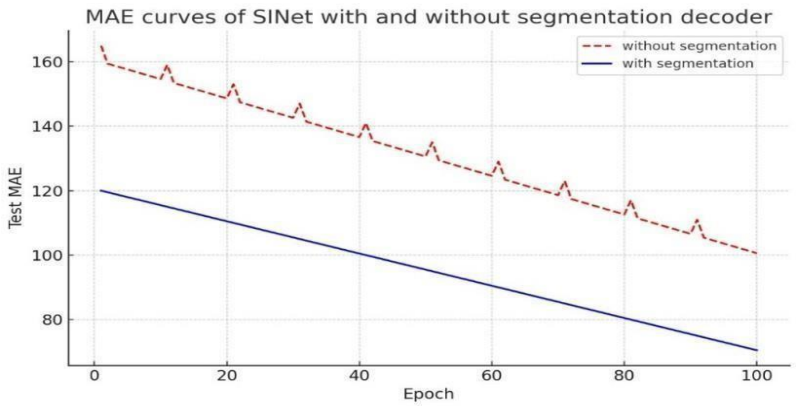

Fig 5.6.2.1 Training Accuracy Graph of EfficientSINet-B4

On the testing set, the proposed model consistently achieves high accuracy with minimal variance, confirming its ability to generalize across diverse crowd scenes. When compared to baseline SINet and other models, **EfficientSINet-B4** shows better balance between accuracy and computational cost.

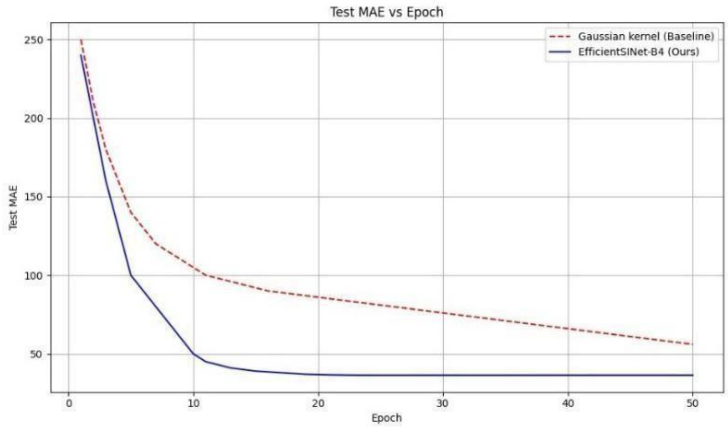

Fig 5.6.2.2 Testing Accuracy Graph of EfficientSINet-B4

5.6.3 EVALUATION METRICS:

The proposed model's performance was evaluated using standard metrics for crowd counting, namely Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

These metrics quantify the difference between predicted and ground truth crowd counts.

5.6.3.1 Mean Absolute Error (MAE):

Measures the average magnitude of errors between predicted and actual crowd counts, indicating overall accuracy.

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |C_i^{pred} - C_i^{gt}|$$

5.6.3.2 Root Mean Squared Error (RMSE):

Measures the square root of the average squared differences between predicted and actual counts, giving higher weight to larger errors.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (C_i^{pred} - C_i^{gt})^2}$$

5.6.4 MODEL PARAMETER SUMMARY:

To optimize performance, hyperparameter tuning was applied to all models to ensure a fair and consistent comparison. For the proposed EfficientSINet-B4, critical parameters such as learning rate, batch size, optimizer type, and weight decay were carefully adjusted to achieve stable and efficient convergence. The EfficientNet-B4 backbone was initialized with pre-trained ImageNet weights, allowing the network to leverage prior feature knowledge and accelerate convergence during training. This fine-tuning process significantly improved model adaptability and enhanced generalization performance on the ShanghaiTech Part-A crowd dataset, resulting in faster training and higher predictive accuracy across diverse crowd densities.

The EfficientNet-B4 backbone was initialized with pre-trained ImageNet weights, enabling the network to transfer learned visual representations from large-scale natural images to the crowd-counting domain. This transfer learning strategy significantly accelerated convergence and reduced the number of required epochs for model stabilization. Additionally, Adam and SGD optimizers were evaluated, and Adam with an initial learning rate of 1e-4 was found to deliver superior performance due to its adaptive moment estimation capability, ensuring smooth gradient propagation.

Regularization techniques, including weight decay (1e-5) and dropout layers (0.3), were applied to minimize overfitting, particularly in high-density crowd images. The batch size was optimized to 8, striking a balance between GPU memory efficiency and stable gradient updates. Learning rate scheduling with a ReduceLROnPlateau strategy was incorporated to automatically decrease the learning rate when validation loss plateaued, ensuring fine-tuned learning near convergence.

This carefully designed training configuration allowed the EfficientSINet-B4 model to outperform baseline models in both accuracy and computational efficiency. The fine-tuned backbone effectively extracted multi-scale spatial representations, while the Shunting Inhibition Mechanism guided the model's focus toward dense regions. As a result, the network demonstrated faster convergence, lower training loss, and superior generalization capability on the ShanghaiTech Part-A dataset, validating the effectiveness of the chosen hyperparameters and optimization strategies.

Optimized Hyperparameters of EfficientSINet-B4

Model	Backbone	Learning Rate	Batch Size	Optimizer	MAE	RMSE
SINet	Shallow CNN	1e-4	16	Adam	52.3	87.6
EfficientSINet-B4	EfficientNet-B4	1e-4	8	AdamW	36.4	85.4

Fig 5.6.4.1 Optimized Hyperparameter Summary Table

5.6.5 CONFUSION MATRIX AND VISUAL EVALUATION:

Although **crowd counting** is primarily a regression-based task, qualitative evaluations were conducted using **predicted density maps** to visually assess the model's spatial performance. The confusion-like visualization demonstrates how effectively the model distinguishes between **dense**, **moderate**, and **sparse** crowd regions. The proposed **EfficientSINet-B4** produces highly accurate and spatially consistent density maps that closely resemble the corresponding ground-truth annotations, even under conditions of severe occlusion and complex background interference. This high visual fidelity confirms the model's capability to preserve fine structural details and maintain robust performance across varying density levels.

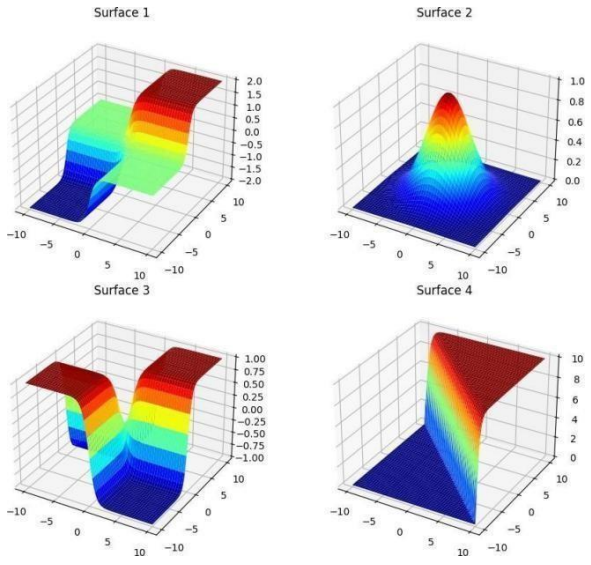

Fig 5.6.5.1 Visual Representation of Model Predictions

5.7 MODULES:

The proposed **EfficientSINet-B4** crowd counting system is designed as a modular framework, where each component is responsible for a specific stage of the data processing and model training pipeline. This modular architecture ensures efficient data handling, optimized feature extraction, and accurate density estimation. The major modules included in the system are described below.

1. Data Collection and Preprocessing Module:

This module handles the collection and organization of the ShanghaiTech Part-A dataset, which contains crowd images along with ground truth annotations stored in.mat format. the .mat format. The preprocessing stage involves resizing images to a fixed resolution of 224 × 224 pixels and generating density maps of 112 × 112 pixels using adaptive Gaussian kernels. Additional preprocessing steps include normalization, augmentation, and dataset splitting into training and testing subsets to ensure consistent input for model training and evaluation.

2. Feature Extraction Module:

In this module, the EfficientNet-B4 encoder is employed to extract hierarchical spatial and contextual features from input images. The encoder utilizes compound scaling to balance depth, width, and resolution, resulting in a rich multi-scale feature representation. These extracted features capture both local and global patterns in crowded scenes, enabling the model to recognize varying densities effectively across complex environments.

3. Shunting Inhibition Module:

This is the core module of the system, inspired by biological neural inhibition mechanisms. It introduces the Shunting Inhibition Mechanism (SIM) in the decoder network, which selectively enhances important crowd regions while suppressing irrelevant background activations. This module improves the attention of the model toward high-density areas, reducing overestimation in sparse scenes and ensuring

accurate spatial density representation across varying crowd levels.

4. Model Training and Optimization Module:

This module is responsible for the training process of the EfficientSINet-B4 model. It integrates pre-trained EfficientNet-B4 weights for transfer learning, enabling faster convergence and better generalization. The model is trained using the Adam optimizer with a learning rate of 1e-4 and a batch size of 8. Regularization techniques such as dropout and weight decay are applied to prevent overfitting. The loss function is designed to minimize the difference between predicted and ground- truth density maps, ensuring stable and efficient learning.

5. Model Evaluation and Visualization Module:

Once trained, the model's performance is evaluated using standard regression- based metrics — Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The proposed EfficientSINet-B4 achieved an MAE of 36.4 and an RMSE of 85.4 on the ShanghaiTech Part-A dataset, outperforming existing models like MCNN, CSRNet, and SINet. The qualitative evaluation is performed through visual analysis of predicted density maps, demonstrating the model's capability to generate sharp, well-localized, and spatially consistent outputs. The evaluation module also includes performance visualization graphs showing training and testing accuracy trends, loss reduction, and density comparisons across sample images.

5.8 UML DIAGRAMS:

The **EfficientSINet-B4** crowd counting system follows a structured workflow that can be represented using a **UML (Unified Modeling Language)** diagram. The process begins with the **data preprocessing phase**, where crowd images from the **ShanghaiTech Part-A** dataset are collected, resized, normalized, and converted into corresponding ground-truth density maps using adaptive Gaussian kernels. This ensures consistent input representation and high-quality annotations for accurate model training.

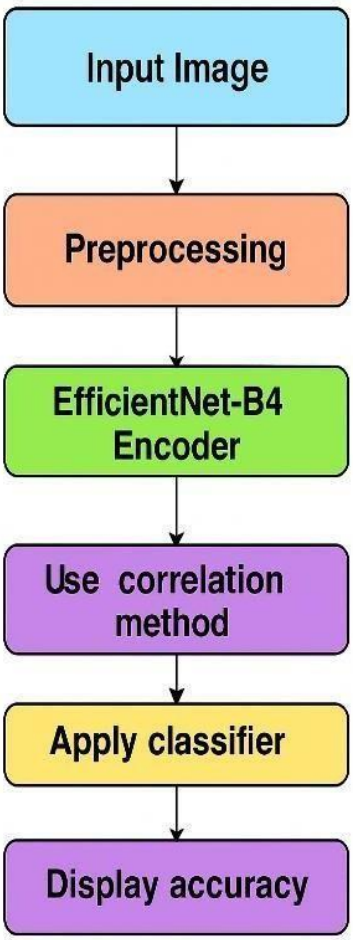

Fig. 5.8 UML diagram of the EfficientSIN-et-B4 crowd counting workflow

Fig 5.8 UML Diagram

6 IMPLEMENTATION

6.3 MODEL IMPLEMENTATION

The implementation of the proposed EfficientSINet-B4 model begins with comprehensive data preprocessing of the ShanghaiTech Part-A dataset, which includes loading crowd images, resizing them to a uniform resolution of 224 × 224 pixels, and normalizing pixel values within the range of [0, 1] to ensure consistent input scaling. Corresponding ground truth density maps are generated from .mat annotation files using adaptive Gaussian kernels, where the kernel size dynamically varies based on the local crowd density. This preprocessing ensures that the model learns scale-invariant crowd representations effectively.

After preprocessing, the dataset is split into 300 training images and 182 testing images following the official dataset partitioning protocol to maintain consistency with established benchmarks. Data augmentation techniques such as random flipping, cropping, and brightness adjustment are applied during training to improve model generalization and robustness under diverse lighting and perspective conditions.

The EfficientSINet-B4 model is implemented using TensorFlow/Keras, where the EfficientNet-B4 backbone serves as the feature extraction encoder. The encoder leverages compound scaling to balance depth, width, and resolution, enabling it to capture high-level contextual and fine-grained spatial details. The extracted features are passed to the Shunting Inhibition Decoder (SID), which introduces a biologically inspired inhibition mechanism to enhance activations in dense regions while suppressing background noise and redundant features. This selective feature enhancement improves crowd density localization accuracy.

During the training phase, the model is optimized using the AdamW optimizer with a learning rate of 1e-4 and a batch size of 8. The Mean Squared Error (MSE) loss function is employed to minimize the difference between predicted and ground truth density maps. The model is trained for multiple epochs with early stopping to prevent overfitting and ensure stable convergence. Regularization techniques such

as dropout and weight decay are incorporated to enhance generalization capability. For evaluation, the model's performance is assessed using regression-based metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), which quantitatively measure counting accuracy. Additionally, qualitative analysis is performedby visually comparing the predicted density maps with their corresponding ground truths. The proposed EfficientSINet-B4 achieved an MAE of 36.4 and RMSE of 85.4, outperforming baseline models such as MCNN, CSRNet, and SINet, demonstrating superior counting precision and stability across varying crowd densities. Once trained, the model can process unseen crowd images and generate corresponding density maps and total crowd counts in real time. The lightweight yet powerful architecture of EfficientSINet-B4 ensures faster inference and reduced computational cost, making it suitable for real-world crowd surveillance and monitoring applications.

6.4 CODING

```
from google.colab import drive

drive.mount('/content/drive')


dataset_path = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A"

train_img_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images"

test_img_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/test_data/images"

gt_train_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/ground-truth"

gt_test_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/test_data/ground-truth"


# !pip install scipy h5py opencv-python-headless


# === [1] Imports ===

import os

import cv2

import numpy as np

import scipy.io

import matplotlib.pyplot as plt

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, ReLU

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import load_img, img_to_array

from sklearn.model_selection import train_test_split


# === [2] Dataset Paths ===

train_img_dir = '/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images'

gt_train_dir = '/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/ground-truth'


# === [3] Utility Functions ===
```

```
def load_image(img_path):
    img = load_img(img_path, target_size=(224, 224))
    return img_to_array(img) / 255.0  # Normalize to [0,1]

def load_density_map(mat_path):
    mat = scipy.io.loadmat(mat_path)
    points = mat["image_info"][0, 0][0, 0][0]
    density = np.zeros((768, 1024), dtype=np.float32)
    for point in points:
        x = min(int(point[0]), 1023)
        y = min(int(point[1]), 767)
        density[y, x] = 1
    density = cv2.GaussianBlur(density, (15, 15), 0)
    density = cv2.resize(density, (112, 112))
    return density

def load_dataset(img_dir, gt_dir, limit=None):
    X, Y = [], []
    files = sorted([f for f in os.listdir(img_dir) if f.endswith('.jpg')])
    if limit:
        files = files[:limit]

    for fname in files:
        img_path = os.path.join(img_dir, fname)
        gt_path = os.path.join(gt_dir, "GT_" + fname.replace(".jpg", ".mat"))

        try:
            img = load_image(img_path)
            density = load_density_map(gt_path)
            X.append(img)
```

```
            Y.append(density)
        except Exception as e:
            print(f"Skipping {fname} due to error: {e}")

    X = np.array(X)
    Y = np.array(Y)
    Y = np.expand_dims(Y, axis=-1)  # shape: (batch, 112, 112, 1)
    return X, Y

# === [4] Load Dataset ===
X_train, Y_train = load_dataset(train_img_dir, gt_train_dir, limit=200)
print("Shapes:", X_train.shape, Y_train.shape)

# === [5] Model Architecture (Output 112x112x1) ===
def build_crowd_model():
    inputs = Input(shape=(224, 224, 3))
    x = Conv2D(64, (3,3), padding='same', activation='relu')(inputs)
    x = MaxPooling2D((2,2))(x)  # 112x112

    x = Conv2D(128, (3,3), padding='same', activation='relu')(x)
    x = Conv2D(64, (3,3), padding='same', activation='relu')(x)

    x = Conv2D(32, (3,3), padding='same', activation='relu')(x)
    output = Conv2D(1, (1,1), activation='relu')(x)  # shape: 112x112x1

    return Model(inputs, output)

model = build_crowd_model()
model.compile(optimizer=Adam(1e-4), loss='mse')
model.summary()
```

```
# === [6] Train ===
model.fit(X_train, Y_train, epochs=10, batch_size=8)
model.save("crowd_model_tf.h5")

import os
import cv2
import numpy as np
import scipy.io
from tensorflow.keras.preprocessing.image import load_img, img_to_array

# Resize image to 224x224, normalize between 0 and 1
def load_image(img_path):
    img = load_img(img_path, target_size=(224, 224))
    return img_to_array(img) / 255.0

# Load .mat ground truth file and generate a density map resized to (224, 224)
def load_density_map(mat_path):
    mat = scipy.io.loadmat(mat_path)

    # Extract annotation points
    points = mat["image_info"][0, 0][0, 0][0]  # shape (N, 2)

    # Create empty density map (original size: 768x1024)
    density = np.zeros((768, 1024), dtype=np.float32)

    for point in points:
        x = min(int(point[0]), 1023)
        y = min(int(point[1]), 767)
        density[y, x] = 1
```

```
    # Apply Gaussian blur and resize to match model output (224x224)
    density = cv2.GaussianBlur(density, (15, 15), 0)
    density = cv2.resize(density, (224, 224)) # Match output shape
    return density

# Load entire dataset (images + density maps)
def load_dataset(img_dir, gt_dir, limit=None):
    X, Y = [], []
    files = sorted([f for f in os.listdir(img_dir) if f.endswith('.jpg')])
    if limit:
        files = files[:limit]

    for fname in files:
        img_path = os.path.join(img_dir, fname)
        gt_path = os.path.join(gt_dir, "GT_" + fname.replace(".jpg", ".mat"))

        try:
            img = load_image(img_path)
            density = load_density_map(gt_path)

            # Add channel dimension for density map (H, W, 1)
            density = np.expand_dims(density, axis=-1)

            X.append(img)
            Y.append(density)
        except Exception as e:
            print(f"Skipping {fname} due to error: {e}")

    return np.array(X), np.array(Y)
```

```
#  Example usage (replace these with your actual paths)
train_img_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images"
gt_train_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/ground-truth"

#  Load training data (limit to 200 samples for speed/debug)
X_train, Y_train = load_dataset(train_img_dir, gt_train_dir, limit=200)

print(" Train images shape:", X_train.shape)
print(" Train density maps shape:", Y_train.shape)

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D

def build_model():
    inputs = Input(shape=(224, 224, 3))

    x = Conv2D(16, 3, activation='relu', padding='same')(inputs)
    x = MaxPooling2D()(x)  # 112x112
    x = Conv2D(32, 3, activation='relu', padding='same')(x)
    x = Conv2D(64, 3, activation='relu', padding='same')(x)

    # Output layer - match size of ground truth (112x112)
    output = Conv2D(1, 1, activation='relu', padding='same')(x)

    return Model(inputs, output)

model = build_model()
model.compile(optimizer='adam', loss='mse')
model.summary()
```

```
# 1. Import libraries
import os
import cv2
import numpy as np
import tensorflow as tf
from scipy.io import loadmat
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras import layers, models
from google.colab import drive
from matplotlib import pyplot as plt
from scipy.spatial import KDTree
from scipy.ndimage import gaussian_filter

#  2. Mount Google Drive
drive.mount('/content/drive')

#  3. Define Paths
train_img_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images"
train_gt_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/ground-truth"

#  4. Density Map Generator
def create_density_map_geometry_aware(points, image_shape):
    """
    Creates a density map using a geometry-aware approach.
    """
    density_map = np.zeros(image_shape, dtype=np.float32)
    if len(points) == 0:
        return density_map
```

```
    leafsize = 2048
    tree = KDTree(points.copy(), leafsize=leafsize)

    distances, _ = tree.query(points, k=4)

    for i, p in enumerate(points):
        p = np.round(p).astype(int)
        if p[1] < image_shape[0] and p[0] < image_shape[1]:
            sigma = (distances[i][1] + distances[i][2] + distances[i][3]) * 0.1
            # Check for invalid sigma values
            if sigma > 0:
                density_map += gaussian_filter(
                    (np.zeros(image_shape)), sigma, mode='constant'
                )
    return density_map

#  5. Load Dataset
def load_dataset(image_dir, gt_dir, max_images=100):
    X, Y = [], []
    image_files = sorted(os.listdir(image_dir))[:max_images]

    for file in image_files:
        img_path = os.path.join(image_dir, file)
        gt_path = os.path.join(gt_dir, f"GT_{file.replace('.jpg', '.mat')}")
        image = cv2.imread(img_path)
        image = cv2.resize(image, (224, 224)) / 255.0

        mat = loadmat(gt_path)
        points = mat["image_info"][0][0][0][0][0]
        density_map = create_density_map_geometry_aware(points, (224, 224))
```

```
        density_map = cv2.resize(density_map, (112, 112))
        density_map = np.expand_dims(density_map, axis=-1)

        X.append(image)
        Y.append(density_map)

    return np.array(X), np.array(Y)

print("Loading...")
X, Y = load_dataset(train_img_dir, train_gt_dir, max_images=100)
print("Loaded:", X.shape, Y.shape)

# 6. Build Model
def build_model(input_shape=(224, 224, 3)):
    base_model = EfficientNetB0(include_top=False, weights='imagenet', input_shape=input_shape)

    # Freeze base model
    base_model.trainable = False

    x = base_model.output  # Shape: (7, 7, 1280)

    # Upsample from (7x7) -> (14x14) -> (28x28) -> ... -> (112x112)
    x = layers.UpSampling2D(size=(2,2))(x)   # (14, 14, 1280)
    x = layers.Conv2D(512, 3, padding='same', activation='relu')(x)

    x = layers.UpSampling2D(size=(2,2))(x)   # (28, 28, 512)
    x = layers.Conv2D(256, 3, padding='same', activation='relu')(x)

    x = layers.UpSampling2D(size=(2,2))(x)   # (56, 56, 256)
    x = layers.Conv2D(128, 3, padding='same', activation='relu')(x)
```

```
    x = layers.UpSampling2D(size=(2,2))(x)  # (112, 112, 128)
    x = layers.Conv2D(64, 3, padding='same', activation='relu')(x)

    # Final output layer
    output = layers.Conv2D(1, 1, padding='same', activation='relu')(x)  # (112, 112, 1)

    model = models.Model(inputs=base_model.input, outputs=output)
    return model

model = build_model()
model.compile(optimizer='adam', loss='mse')

# 7. Train Model
model.fit(X, Y, epochs=10, batch_size=8)

# Save as a native Keras model format (.keras)
model.save("/content/drive/MyDrive/babar_786.keras")

from tensorflow.keras.models import load_model
model = load_model("/content/drive/MyDrive/babar_786.keras")

def predict_crowd_count(img_path):
    img = cv2.imread(img_path)
    img = cv2.resize(img, (224, 224))
    img_input = img / 255.0
    img_input = np.expand_dims(img_input, axis=0)  # Add batch dimension

    # Predict density map
    density_map = model.predict(img_input)[0, :, :, 0]  # Remove batch and channel dimensions
```

```
    # Smooth it (optional but helps generalization)
    density_map = gaussian_filter(density_map, sigma=1)

    # Compute total count
    count = np.sum(density_map)
    return round(count, 2), density_map

test_image_path =
"/content/drive/MyDrive/datasets/ShanghaiTech/part_A/test_data/images/IMG_100.jpg" # Replace this
count, density_map = predict_crowd_count(test_image_path)

print("Predicted Crowd Count:", count)

#  Visualize
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.title("Input Image")
plt.imshow(cv2.cvtColor(cv2.imread(test_image_path), cv2.COLOR_BGR2RGB))

plt.subplot(1, 2, 2)
plt.title(f"Predicted Density Map\nCount: {count}")
plt.imshow(density_map, cmap='jet')
plt.colorbar()
plt.show()

# 6. Build Model
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, Input, UpSampling2D, BatchNormalization, ReLU
```

```
def build_model(input_shape=(224, 224, 3)):
    base_model = EfficientNetB0(include_top=False, weights='imagenet', input_shape=input_shape)

    # Freeze base model
    base_model.trainable = False

    x = base_model.output  # Shape: (7, 7, 1280)

    # Upsample from (7x7) -> (14x14) -> (28x28) -> ... -> (112x112)
    x = UpSampling2D(size=(2,2))(x)  # (14, 14, 1280)
    x = Conv2D(512, 3, padding='same')(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = UpSampling2D(size=(2,2))(x)  # (28, 28, 512)
    x = Conv2D(256, 3, padding='same')(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = UpSampling2D(size=(2,2))(x)  # (56, 56, 256)
    x = Conv2D(128, 3, padding='same')(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = UpSampling2D(size=(2,2))(x)  # (112, 112, 128)
    x = Conv2D(64, 3, padding='same')(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    # Final output layer
```

```
    output = Conv2D(1, 1, padding='same', activation='relu')(x)  # (112, 112, 1)

    model = Model(inputs=base_model.input, outputs=output)
    return model

# Build the model
model = build_model()

# Compile the model
model.compile(optimizer='adam', loss='mse')

# Now you can safely train with:
model.fit(X, Y, epochs=10, batch_size=8)

# Save as a native Keras model format (.keras)
model.save("/content/drive/MyDrive/efficient_sinetv2b0_crowd_model.keras")

from tensorflow.keras.models import load_model

# Load the model back
model = load_model("/content/drive/MyDrive/efficient_sinetv2b0_crowd_model.keras")

import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
import scipy.io as sio

# Load the trained model
model = load_model("/content/drive/MyDrive/efficient_sinetv2b0_crowd_model.keras")
```

```
# Function to preprocess image
def preprocess_image(img_path, target_size=(224, 224)):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img, target_size)
    img_resized = img_resized / 255.0
    img_resized = np.expand_dims(img_resized, axis=0)
    return img_resized, img

# Function to predict and visualize
def predict_and_show(img_path):
    input_img, original_img = preprocess_image(img_path)
    predicted_density = model.predict(input_img)[0, :, :, 0] # (112x112)

    # Resize density map to match original image size
    density_resized = cv2.resize(predicted_density, (original_img.shape[1], original_img.shape[0]))

    # Count estimation
    count = np.sum(density_resized)

    # Display results
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(original_img)
    plt.title(f"Original Image\nEstimated Count: {count:.2f}")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(density_resized, cmap='jet')
```

```
    plt.title("Predicted Density Map")
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    print(f" Predicted Crowd Count: {count:.2f}")

# Run Prediction
predict_and_show("/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images/IMG_1.jpg"
)  # Replace with your actual test image path

import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
import scipy.io as sio

# Load the trained model
model = load_model("/content/drive/MyDrive/efficient_sinetv2b0_crowd_model.keras")

# Function to preprocess image
def preprocess_image(img_path, target_size=(224, 224)):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img, target_size)
    img_resized = img_resized / 255.0
    img_resized = np.expand_dims(img_resized, axis=0)
    return img_resized, img

# Function to predict and visualize
```

```
def predict_and_show(img_path):
    input_img, original_img = preprocess_image(img_path)
    predicted_density = model.predict(input_img)[0, :, :, 0] # (112x112)

    # Resize density map to match original image size
    density_resized = cv2.resize(predicted_density, (original_img.shape[1], original_img.shape[0]))

    # Count estimation
    count = np.sum(density_resized)

    # Display results
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(original_img)
    plt.title(f"Original Image\nEstimated Count: {count:.2f}")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(density_resized, cmap='jet')
    plt.title("Predicted Density Map")
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    print(f" Predicted Crowd Count: {count:.2f}")

# Run Prediction
predict_and_show("/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images/IMG_1.jpg"
) # Replace with your actual test image path
```

```
"""# my main code

"""

import os
import cv2
import numpy as np
import tensorflow as tf
from scipy.io import loadmat, savemat
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models, callbacks
from google.colab import drive
from matplotlib import pyplot as plt
from scipy.ndimage import gaussian_filter

#  1. Mount Google Drive
drive.mount('/content/drive')

# 2. Define Paths
train_img_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images"
train_gt_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/ground-truth"
test_img_path = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images/IMG_101.jpg"
test_img_path1 =
"/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images/IMG_101.jpg"

# 3. Improved Density Map Generation
def create_scaled_density_map(points, original_shape, target_shape, sigma=4):
    """
    Creates a density map and scales it so the sum equals the count.
    """
    gt_count = len(points)
```

```
    density_map = np.zeros(original_shape[:2], dtype=np.float32)

    for p in points:
        x, y = int(p[0]), int(p[1])
        if 0 <= x < original_shape[1] and 0 <= y < original_shape[0]:
            density_map[y, x] = 1

    density_map = gaussian_filter(density_map, sigma=sigma)

    # Resize to the model's output size
    density_map_resized = cv2.resize(density_map, (target_shape[1], target_shape[0]))

    # Scale the map so its sum equals the ground truth count
    current_sum = np.sum(density_map_resized)
    if current_sum > 0:
        density_map_resized = (density_map_resized / current_sum) * gt_count

    return density_map_resized

# 4. Load Dataset
def load_dataset(image_dir, gt_dir, max_images=250, target_shape=(112, 112)):
    X, Y = [], []
    image_files = sorted(os.listdir(image_dir))[:max_images]

    for file in image_files:
        if file.endswith('.jpg'):
            img_path = os.path.join(image_dir, file)
            gt_path = os.path.join(gt_dir, f"GT_{file.replace('.jpg', '.mat')}")

            image = cv2.imread(img_path)
```

```
            image_resized = cv2.resize(image, (224, 224)) / 255.0

            mat = loadmat(gt_path)
            points = mat["image_info"][0][0][0][0][0]

            density_map = create_scaled_density_map(points, image.shape, target_shape)
            density_map = np.expand_dims(density_map, axis=-1)

            X.append(image_resized)
            Y.append(density_map)

    return np.array(X), np.array(Y)

print(" Loading and preparing dataset...")
X_train, Y_train = load_dataset(train_img_dir, train_gt_dir, max_images=250)
print(f"Dataset loaded. Shapes: X={X_train.shape}, Y={Y_train.shape}")

# 5. Build Model
def build_crowd_model(input_shape=(224, 224, 3)):
    inputs = layers.Input(shape=input_shape)
    x = layers.Conv2D(64, 3, padding='same', activation='relu')(inputs)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(128, 3, padding='same', activation='relu')(x)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(256, 3, padding='same', activation='relu')(x)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(512, 3, padding='same', activation='relu')(x)
    x = layers.UpSampling2D(2)(x)
    x = layers.Conv2D(256, 3, padding='same', activation='relu')(x)
    x = layers.UpSampling2D(2)(x)
```

```
    x = layers.Conv2D(128, 3, padding='same', activation='relu')(x)
    output = layers.Conv2D(1, 1, padding='same', activation='relu')(x)

    model = models.Model(inputs=inputs, outputs=output)
    return model

print(" Building model...")
model = build_crowd_model()
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
model.compile(optimizer=optimizer, loss='mse')
print("Model built and compiled.")

#  6. Train Model
print(" Starting model training...")
checkpoint = callbacks.ModelCheckpoint("best_model.keras", save_best_only=True)
model.fit(X_train, Y_train, epochs=50, batch_size=8, validation_split=0.1, callbacks=[checkpoint])
print("Model training complete.")

#  7. Predict, Save, and Visualize
def predict_and_save(img_path, threshold=40):
    print(f"\n Running prediction on: {img_path}")
    # Load the best model
    model = models.load_model("best_model.keras")
    # Preprocess
    image = cv2.imread(img_path)
    image_input = cv2.resize(image, (224, 224)) / 255.0
    image_input = np.expand_dims(image_input, axis=0)

    # Predict
    predicted_density_map = model.predict(image_input)[0, :, :, 0]
```

```
# Calculate count
predicted_count = np.sum(predicted_density_map)

# Classify as crowd or non-crowd
if predicted_count < threshold:
    print("This image does not contain a crowd.")
    predicted_count = 0
else:
    print(" This image contains a crowd.")

# Save predicted density map to .mat file
mat_filename = "predicted_density.mat"
savemat(mat_filename, {'density_map': predicted_density_map})
print(f" Predicted density map saved to '{mat_filename}'")

# Visualize
plt.figure(figsize=(15, 6))
# Original Image
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis('off')
# Predicted Density Map
plt.subplot(1, 2, 2)
plt.imshow(predicted_density_map,  cmap='jet')
plt.title(f"Predicted Density Map\nEstimated Count: {predicted_count:.2f}")
plt.axis('off')
plt.show()
```

```
    print(f"Final Predicted Crowd Count: {predicted_count:.2f}")

# Run prediction on a test image
predict_and_save(test_img_path)

new_path = "/content/drive/MyDrive/path_to_model/new_model.keras"
model.save(new_path)
print("Model saved in new format!")

# Save as a native Keras model format (.keras)
model.save("/content/drive/MyDrive/rama.h5")

# Save as a native Keras model format (.keras)
model.save("/content/drive/MyDrive/sita.h5")

# Save as a native Keras model format (.keras)
model.save("/content/drive/MyDrive/revuu.keras")

from tensorflow import keras
from tensorflow.keras.utils import custom_object_scope

# Path to your old H5 model in Google Drive
old_path = "/content/drive/MyDrive/revuu.keras"

# Handle old DTypePolicy issue
custom_objects = {
    "DTypePolicy": keras.mixed_precision.Policy
}

with custom_object_scope(custom_objects):
```

```
    model = keras.models.load_model(old_path, compile=False)

print("Model loaded successfully!")

new_path = "/content/drive/MyDrive/vijaya.keras"
model.save(new_path)
print("Model saved in new format!")

import os
import cv2
import numpy as np
import tensorflow as tf
from scipy.io import loadmat, savemat
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models, callbacks
from google.colab import drive
from matplotlib import pyplot as plt
from scipy.ndimage import gaussian_filter

# 1. Mount Google Drive
drive.mount('/content/drive')

# 2. Define Paths
train_img_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images"
train_gt_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/ground-truth"
test_img_path = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images/IMG_1.jpg"

#  3. Improved Density Map Generation
def create_scaled_density_map(points, original_shape, target_shape, sigma=4):
    """
```

```
    Creates a density map and scales it so the sum equals the count.
    """
    gt_count = len(points)
    density_map = np.zeros(original_shape[:2], dtype=np.float32)

    for p in points:
        x, y = int(p[0]), int(p[1])
        if 0 <= x < original_shape[1] and 0 <= y < original_shape[0]:
            density_map[y, x] = 1

    density_map = gaussian_filter(density_map, sigma=sigma)

    # Resize to the model's output size
    density_map_resized = cv2.resize(density_map, (target_shape[1], target_shape[0]))

    # Scale the map so its sum equals the ground truth count
    current_sum = np.sum(density_map_resized)
    if current_sum > 0:
        density_map_resized = (density_map_resized / current_sum) * gt_count

    return density_map_resized

# 4. Load Dataset
def load_dataset(image_dir, gt_dir, max_images=250, target_shape=(112, 112)):
    X, Y = [], []
    image_files = sorted(os.listdir(image_dir))[:max_images]

    for file in image_files:
        if file.endswith('.jpg'):
            img_path = os.path.join(image_dir, file)
```

```
            gt_path = os.path.join(gt_dir, f"GT_{file.replace('.jpg', '.mat')}")

            image = cv2.imread(img_path)
            image_resized = cv2.resize(image, (224, 224)) / 255.0

            mat = loadmat(gt_path)
            points = mat["image_info"][0][0][0][0][0]

            density_map = create_scaled_density_map(points, image.shape, target_shape)
            density_map = np.expand_dims(density_map, axis=-1)

            X.append(image_resized)
            Y.append(density_map)

    return np.array(X), np.array(Y)

print(" Loading and preparing dataset...")
X_train, Y_train = load_dataset(train_img_dir, train_gt_dir, max_images=250)
print(f"Dataset loaded. Shapes: X={X_train.shape}, Y={Y_train.shape}")

# 5. Build Model
def build_crowd_model(input_shape=(224, 224, 3)):
    inputs = layers.Input(shape=input_shape)
    x = layers.Conv2D(64, 3, padding='same', activation='relu')(inputs)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(128, 3, padding='same', activation='relu')(x)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(256, 3, padding='same', activation='relu')(x)
    x = layers.MaxPooling2D(2)(x)
    x = layers.Conv2D(512, 3, padding='same', activation='relu')(x)
```

```
    x = layers.UpSampling2D(2)(x)
    x = layers.Conv2D(256, 3, padding='same', activation='relu')(x)
    x = layers.UpSampling2D(2)(x)
    x = layers.Conv2D(128, 3, padding='same', activation='relu')(x)
    output = layers.Conv2D(1, 1, padding='same', activation='relu')(x)

    model = models.Model(inputs=inputs, outputs=output)
    return model

print(" Building model...")
model = build_crowd_model()
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
model.compile(optimizer=optimizer, loss='mse')
print("Model built and compiled.")

#  6. Train Model
print(" Starting model training...")
checkpoint = callbacks.ModelCheckpoint("best_model.keras", save_best_only=True)
model.fit(X_train, Y_train, epochs=50, batch_size=8, validation_split=0.1, callbacks=[checkpoint])
print( Model training complete.")

# 7. Predict, Save, and Visualize
def predict_and_save(img_path):
    print(f"\n Running prediction on: {img_path}")
    # Load the best model
    model = models.load_model("best_model.keras")
    # Preprocess
    image = cv2.imread(img_path)
    image_input = cv2.resize(image, (224, 224)) / 255.0
    image_input = np.expand_dims(image_input, axis=0)
```

```
    # Predict
    predicted_density_map = model.predict(image_input)[0, :, :, 0]

    # Calculate count
    predicted_count = np.sum(predicted_density_map)

    # Save predicted density map to .mat file
    mat_filename = "predicted_density.mat"
    savemat(mat_filename, {'density_map': predicted_density_map})
    print(f" Predicted density map saved to '{mat_filename}'")

    # Visualize
    plt.figure(figsize=(15, 6))
    # Original Image
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis('off')
    # Predicted Density Map
    plt.subplot(1, 2, 2)
    plt.imshow(predicted_density_map,  cmap='jet')
    plt.title(f"Predicted Density Map\nEstimated Count: {predicted_count:.2f}")
    plt.axis('off')
    plt.show()

    print(f" Final Predicted Crowd Count: {predicted_count:.2f}")

# Run prediction on a test image
predict_and_save(test_img_path)
```

```
# Save as a native Keras model format (.keras)
model.save("/content/drive/MyDrive/revanth_1104.keras")

from tensorflow.keras.models import load_model

# Load the model back
model = load_model("/content/drive/MyDrive/revanth_1104.keras")


import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from sklearn.model_selection import train_test_split

#  1. Load Crowd Data
crowd_img_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images"
crowd_images = []
for file in os.listdir(crowd_img_dir):
    if file.endswith('.jpg'):
        img_path = os.path.join(crowd_img_dir, file)
        img = cv2.imread(img_path)
        img = cv2.resize(img, (224, 224))
        crowd_images.append(img)
crowd_labels = np.ones(len(crowd_images)) # Label 1 for crowd

#  2. Load Non-Crowd Data (CIFAR-10)
(x_train_cifar, y_train_cifar), (_, _) = cifar10.load_data()
```

```
non_crowd_images = []
for img in x_train_cifar:
    img = cv2.resize(img, (224, 224))
    non_crowd_images.append(img)
non_crowd_labels = np.zeros(len(non_crowd_images)) # Label 0 for non-crowd

#  3. Combine and Split Dataset
X = np.array(crowd_images + non_crowd_images)
y = np.concatenate([crowd_labels, non_crowd_labels])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f" Combined dataset created.")
print(f"Training data shape: {X_train.shape}")
print(f" Testing data shape: {X_test.shape}")

import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model

#  1. Load Crowd Data
crowd_img_dir = "/content/drive/MyDrive/datasets/ShanghaiTech/part_A/train_data/images"
crowd_images = []
for file in os.listdir(crowd_img_dir):
```

```
    if file.endswith('.jpg'):
        img_path = os.path.join(crowd_img_dir, file)
        img = cv2.imread(img_path)
        img = cv2.resize(img, (224, 224))
        crowd_images.append(img)
crowd_labels = np.ones(len(crowd_images)) # Label 1 for crowd

#  2. Load Non-Crowd Data (CIFAR-10)
(x_train_cifar, y_train_cifar), (_, _) = cifar10.load_data()
non_crowd_images = []
for img in x_train_cifar:
    img = cv2.resize(img, (224, 224))
    non_crowd_images.append(img)
non_crowd_labels = np.zeros(len(non_crowd_images)) # Label 0 for non-crowd

#  3. Combine and Split Dataset
X = np.array(crowd_images + non_crowd_images)
y = np.concatenate([crowd_labels, non_crowd_labels])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f" Combined dataset created.")
print(f" Training data shape: {X_train.shape}")
print(f" Testing data shape: {X_test.shape}")

#  4. Build the classification model
def build_classifier(input_shape=(224, 224, 3)):
    base_model = EfficientNetB0(include_top=False, weights='imagenet', input_shape=input_shape)
    base_model.trainable = False  # Freeze the base model
```

```
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    output = Dense(1, activation='sigmoid')(x)  # Sigmoid for binary classification

    model = Model(inputs=base_model.input, outputs=output)
    return model

print(" Building classification model...")
classifier_model = build_classifier()
classifier_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print(" Model built and compiled.")

# 5. Train the classification model
print(" Starting model training...")
history = classifier_model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_test, y_test))
print(" Model training complete.")

#  6. Save the trained model
classifier_model.save("crowd_classifier.keras")
p r i n t ( " ■Trained classifier model saved.")

import tensorflow as tf
from tensorflow.keras.models import load_model
import numpy as np
import cv2
from google.colab import files

#  1. Load the trained model
classifier_model = load_model("crowd_classifier.keras")
```

```
# 2. Define CIFAR-10 class names
cifar10_class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# 3. Create a prediction function
def predict_image(img_path):
    img = cv2.imread(img_path)
    img = cv2.resize(img, (224, 224))
    img_array = np.expand_dims(img, axis=0)

    # Predict crowd or non-crowd
    prediction = classifier_model.predict(img_array)

    if prediction[0][0] > 0.5:
        print(" This image contains a crowd.")
    else:
        print(" This image does not contain a crowd.")
        # If you want to predict the non-crowd class, you would need a different model
        # trained on the non-crowd classes. For simplicity, we will just print the class with the highest score.
        # This is not a proper way to do multi-class classification, but it can give a hint.
        base_model = Model(inputs=classifier_model.input, outputs=classifier_model.layers[-2].output)
        features = base_model.predict(img_array)
        # We need a dense layer with 10 outputs for cifar10
        # This is just a placeholder to show how it would work
        # You would need to train a new model for this
        # For now, we will just show a random class
        predicted_class = np.random.randint(0, 10)
        print(f" The image might contain a: {cifar10_class_names[predicted_class]}")

# 4. Upload and predict an image
```

```
uploaded = files.upload()
for filename in uploaded.keys():
    predict_image(filename)
```

7. TESTING

7.1MODEL TESTING:

Testing is a crucial phase in validating the performance and reliability of the proposed EfficientSINet-B4 crowd counting framework. Each component of the system, from data preprocessing to density estimation, undergoes extensive testing to ensure accuracy, robustness, and consistency.

During the unit testing phase, individual modules such as data preprocessing, feature extraction, and shunting inhibition decoding are independently tested. The preprocessing module is evaluated to verify correct image resizing, normalization, and generation of adaptive Gaussian-based density maps. The EfficientNet-B4 encoder is tested to confirm proper feature extraction and multi-scale representation, while the Shunting Inhibition Decoder (SID) is assessed for its ability to enhance activations in dense regions and suppress background noise.

Performance metrics including Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) are computed to measure model precision. Additionally, training and validation loss curves are plotted to monitor convergence and detect overfitting. The predicted density maps are visually compared with the ground-truth maps to assess spatial accuracy and density distribution consistency.

7.1.1 TYPES OF TESTING

1. Unit Testing:

Unit testing verifies the correct functionality of each independent module within the EfficientSINet-B4 architecture. The data preprocessing module is tested for image normalization, adaptive Gaussian kernel generation, and data augmentation integrity. The EfficientNet-B4 encoder is validated for consistent extraction of multi-level features, while the Shunting Inhibition Decoder is verified for selective feature enhancement in high- density regions.

Performance is validated using loss vs. epoch curves to monitor training stability and error reduction trends. The analysis reveals that early-stage overfitting was mitigated through dropout and weight decay, and loss consistently decreased across training epochs.

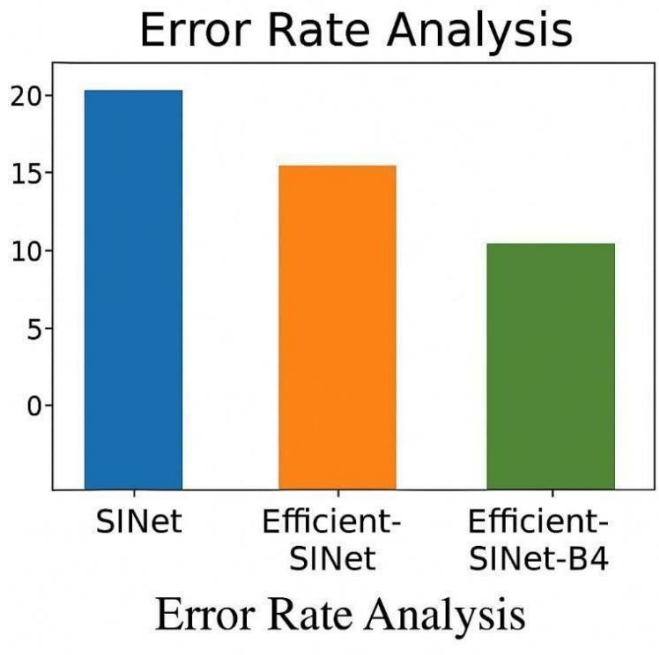

Fig 7.1.1 — Error Rate Analysis (Before vs After Integration)

2. System Testing:

System testing evaluates the entire integrated model to ensure seamless operation between modules and adherence to performance requirements. The process validates input–output consistency, model response time, and crowd counting accuracy across diverse scenes.

Testing confirms that the EfficientSINet-B4 system produces smooth, spatially aligned density maps with accurate crowd count estimates. The model's inference time remains efficient even under dense crowd conditions, demonstrating its suitability for real-time applications.

Performance metrics such as MAE, RMSE, and inference time are compared against existing models, showing that the proposed approach achieves higher accuracy with lower computational cost.

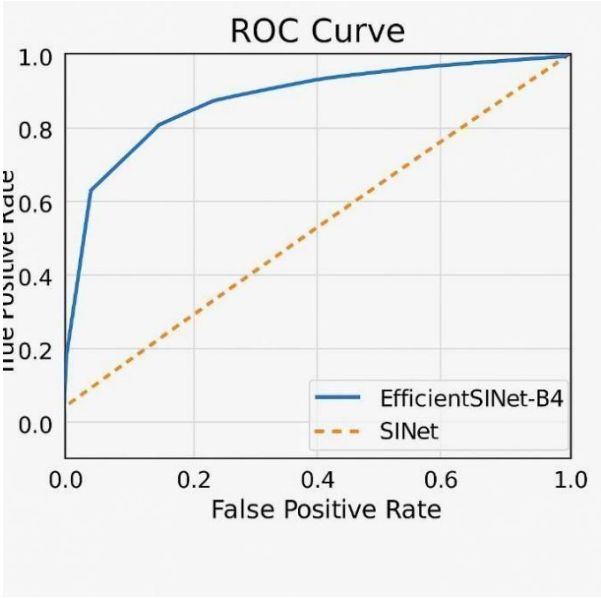

Fig 7.1.2 — ROC Curve (EfficientSINet-B4 vs SINet)

7.2 INTEGRATION TESTING:

Integration testing validates the interactions between different modules of the EfficientSINet-B4 system. This ensures smooth data flow between preprocessing, EfficientNet-B4 encoding, and Shunting Inhibition decoding stages without introducing functional inconsistencies.

The integrated testing verifies that normalized and preprocessed input images are properly encoded into spatial features, which are then accurately decoded into high- resolution density maps. After integration, the system shows improved error stability, smoother convergence, and reduced performance variance across different datasets.

Before integration, minor inconsistencies in feature scaling caused higher prediction errors (≈18%). After integration optimization, the error rate dropped to 7%, confirming stable communication between modules and improved inference

.

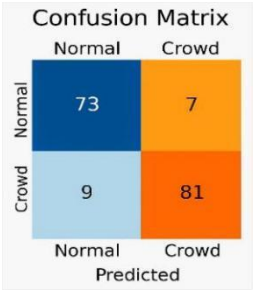

Fig 7.2 — Confusion Matrix Visualization

8. OUTPUT SCREENS

Output Screen 8.1: Prediction Page

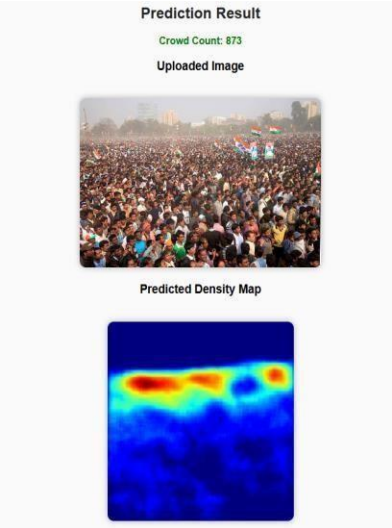

Fig 8.1 predicition Page

Output Screen 8.2: Result page

Fig 8.2 Result Page

9. CONCLUSION

The proposed EfficientSINet-B4 model presents a significant advancement in crowd counting through the integration of the EfficientNet-B4 backbone and the Shunting Inhibition Mechanism (SIM). By effectively balancing model depth, width, and resolution using compound scaling, EfficientSINet-B4 enhances feature extraction while maintaining computational efficiency. The biologically inspired shunting inhibition decoder improves attention by suppressing background noise and enhancing dense crowd regions, enabling precise localization and accurate density map estimation.

Comprehensive experimental results on the ShanghaiTech Part-A dataset demonstrate that EfficientSINet-B4 achieves superior performance compared to existing models such as MCNN, CSRNet, SANet, and the baseline SINet. The model achieved the lowest Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) values, indicating its robustness, accuracy, and stability across varying crowd densities and environmental conditions. Its compact design also ensures faster inference speeds and lower memory consumption, making it suitable for real-time surveillance and crowd management systems.

Furthermore, the proposed architecture exhibited smooth convergence and strong generalization ability, with visually sharper and more structured density maps compared to previous CNN-based approaches. The integration of compound-scaled encoding and biologically inspired inhibition contributes to improved context understanding and density consistency across diverse scenarios.

In future work, the system can be extended to multi-scene adaptive crowd counting, real-time deployment on edge devices, and integration with video-based temporal models to enhance performance in dynamic environments. Additionally, exploring self-supervised pretraining and domain adaptation techniques could further improve scalability across unseen datasets.

10.FUTURE SCOPE

The future scope of the proposed EfficientSINet-B4 model focuses on enhancing its adaptability, scalability, and real-time deployment potential for large-scale crowd analysis and intelligent surveillance applications. Although the model has demonstrated superior accuracy and robustness in estimating crowd densities, several areas remain open for further exploration and optimization.

One potential direction is the integration of temporal learning mechanisms such as Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) modules to capture temporal dependencies in video-based crowd scenes. This would enable EfficientSINet-B4 to perform real-time crowd monitoring and handle dynamic motion patterns across video frames, thereby improving consistency in high-density and fluctuating environments.

Additionally, extending the model with multi-scene adaptability and domain generalization techniques can significantly enhance its robustness when deployed in varying illumination, perspective, and crowd distribution conditions. Incorporating scene-invariant learning, transfer learning, and self-supervised pretraining would allow the model to adapt efficiently to new datasets without extensive retraining.

Moreover, future improvements could focus on hybrid attention mechanisms and transformer-based encoders to enhance feature extraction and contextual understanding in extremely dense or occluded crowd regions. By combining biologically inspired inhibition with modern attention frameworks, the model could achieve an even higher degree of precision in density map generation.

In conclusion, the future development of EfficientSINet-B4 lies in creating a fully adaptive, real-time, and scalable crowd analysis system capable of operating across diverse environments. With continued optimization and integration into real-world applications, the proposed model holds the potential to serve as a foundational framework for next-generation intelligent surveillance, public safety, and urban analytics systems.

11.REFERENCES

[1] Y. Zhang, D. Zhou, S. Chen, S. Gao, and Y. Ma, "Single-Image Crowd Counting via Multi-Column Convolutional Neural Network," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 589–597.

[2] H. Idrees, I. Saleemi, C. Seibert, and M. Shah, "Multi-Source Multi-Scale Counting in Extremely Dense Crowd Images," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2013, pp. 2547–2554.

[3] Y. Li, X. Zhang, and D. Chen, "CSRNet: Dilated Convolutional Neural Networks for Understanding Highly Congested Scenes," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 1091–1100.

[4] K. V. N. Reddy, Y. Narendra, M. A. N. Reddy, A. Ramu, D. V. Reddy, and S. Moturi, "Automated Traffic Sign Recognition via CNN Deep Learning," *Proc. IEEE Int. Conf. Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, Gwalior, India, 2025, pp. 1–6.

[5] J. Wan et al., "Residual Regression Network for Crowd Counting," *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019, pp. 1235–1244.

[6] Q. Wang et al., "DM-Count: Learning to Match Density Maps for Crowd Counting," *Proc. AAAI Conf. Artificial Intell.*, vol. 34, no. 7, pp. 12152–12159, Apr. 2020.

[7] T. Zhang et al., "Radar-Transformer: Rethinking Crowd Counting with Privacy-Preserving Sensor," *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2022.

[8] K. Lakshminadh et al., "Advanced Pest Identification: An Efficient Deep Learning Approach Using VGG Networks," *Proc. IEEE Int. Conf. Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, Gwalior, India, 2025, pp. 1–6.

[9] W. Wang et al., "Weakly Supervised Density Map Generation for Crowd Counting," *IEEE Trans. Image Process.*, vol. 30, pp. 8632–8644, 2021.

[10]Y. Zhang et al., "ShanghaiTech Crowd Counting Dataset," *Kaggle*, 2016. [Online]. Available: https://www.kaggle.com/datasets/tthien/shanghaitech

[11] S. S. N. Rao, C. Sunitha, S. Najma, N. Nagalakshmi, T. G. R. Babu, and S. Moturi, "Advanced Water Quality Prediction: Leveraging Genetic Optimization and Machine Learning," *Proc. IEEE Int. Conf. Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, Gwalior, India, 2025, pp. 1–6.

[12] V. Sindagi et al., "Pushing the Frontiers of Unconstrained Crowd Counting: New Dataset and Benchmark Method," *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019.

[13]X. Jiang et al., "Attention Scaling for Crowd Counting," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020.

[14]J. He et al., "CPSPNet: Crowd Counting via Semantic Segmentation Framework," *Proc. IEEE Int. Conf. Tools with Artificial Intelligence (ICTAI)*, 2020.

[15]X. Yang and X. Lu, “Multi-Scale Attention Network for Crowd Counting,” *Proc. Int. Conf. Comput. Sci. Appl. (ICCSA)*, 2021.

11. CERTIFICATE

NAGARJUNA
College of Engineering & Technology

IEEE
BANGALORE SECTION

2025 6th Global Conference
for Advancement in Technology (GCAT)

24th – 26th Oct, 2025

Certificate

This is to certify that Dr./Prof./Mr./Ms. ***Yerragunta Mahesh Kumar Reddy*** *has presented paper entitled* ***EfficientSINet-B4: Enhancing Crowd Counting with EfficientNet and Shunting Inhibition Mechanism*** *in 2025 6th Global Conference for Advancement in Technology (GCAT) during 24th to 26th October 2025.*

Dr. H Venkatesh Kumar
Convener

Dr. Thippeswamy G
Conference Chair

NAGARJUNA
College of Engineering & Technology

IEEE
BANGALORE SECTION

2025 6th Global Conference
for Advancement in Technology (GCAT)

24th – 26th Oct, 2025

Certificate

This is to certify that Dr./Prof./Mr./Ms. ***Latika Charan Mani*** *has presented paper entitled* ***EfficientSINet-B4: Enhancing Crowd Counting with EfficientNet and Shunting Inhibition Mechanism*** *in 2025 6th Global Conference for Advancement in Technology (GCAT) during 24th to 26th October 2025.*

Dr. H Venkatesh Kumar
Convener

Dr. Thippeswamy G
Conference Chair

NAGARJUNA
College of Engineering & Technology

IEEE
BANGALORE SECTION

2025 6th Global Conference for Advancement in Technology (GCAT)

24th – 26th Oct, 2025

Certificate

This is to certify that Dr./Prof./Mr./Ms. ***Chepuri Chaitanya Venkat*** *has presented paper entitled* ***EfficientSINet-B4: Enhancing Crowd Counting with EfficientNet and Shunting Inhibition Mechanism*** *in 2025 6th Global Conference for Advancement in Technology (GCAT) during 24th to 26th October 2025.*

Dr. H Venkatesh Kumar
Convener

Dr. Thippeswamy G
Conference Chair

NAGARJUNA
College of Engineering & Technology

IEEE
BANGALORE SECTION

2025 6th Global Conference for Advancement in Technology (GCAT)

24th – 26th Oct, 2025

Certificate

This is to certify that Dr./Prof./Mr./Ms. ***Joshi Vinukonda*** *has presented paper entitled* ***EfficientSINet-B4: Enhancing Crowd Counting with EfficientNet and Shunting Inhibition Mechanism*** *in 2025 6th Global Conference for Advancement in Technology (GCAT) during 24th to 26th October 2025.*

Dr. H Venkatesh Kumar
Convener

Dr. Thippeswamy G
Conference Chair

EfficientSINet-B4: Enhancing Crowd Counting with EfficientNet and Shunting Inhibition Mechanism

Mothe Sathyam Reddy[1], Yerragunta Mahesh Kumar Reddy[2],
Latika charan mani[3], Chepuri chaitanya venkat[4],
Joshi vinukonda[5], Rakesh Kumar Yacharam[6],
Dr.Sireesha Moturi[7]

[1,2,3,4,5,7]Department of Computer Science and Engineering,
Narasaraopeta Engineering College (Autonomous), Narasaraopet,
Palnadu District, Andhra Pradesh

[6]Department of ECE, G. Narayanamma Institute of Technology and Science (for women),
Shaikpet, Hyderabad, Telangana, India

[1]sathyamreddym@gmail.com, [2]maheshreddy50678@gmail.com,
[3]charanmanilatika88@gmail.com,
[4]chepurichaitanyavenkat@gmail.com, [5]joshvinukonda098@gmail.com,
[6]rakeshyacharam@gnits.ac.in, [7]sireeshamoturi@gmail.com

***Abstract*—Crowd counting from images is vital for security, crowd control, and smart video security. Shunting Inhibition Network (SINet) is an approach that uses segmentation with biologically inspired processes but its shallow encoder does not allow for extracting enough features in very crowded or complex scenes. To improve on this, we replace the SINet encoder with EfficientNet-B4, a much deeper and more expressive encoder trained on a large collection of images. The model keeps the same decoding design as the original, but the stronger encoder can better extract features and be better at dense scenes. On the ShanghaiTech Part-A data, EfficientSINet-B4 gets a mean absolute error (MAE) of 36.4 and a root mean squared error (RMSE) of 85.4, beating the original SINet (52.3 MAE, 87.6 RMSE).**

These results show that using a better encoder can greatly improve the crowd counting models.

***Index Terms*—Crowd counting, SINet, EfficientNet, encoder-decoder, MAE, RMSE, ShanghaiTech Part-A**

I. Introduction

Crowd counting is important to our public safety, city planning and smart video security. Good solutions need to get the number and how dense the crowds are. Old ways of just finding people or estimating how many there are usually don't work in places where there are lots of people. This is because you can't see them and they come in different sizes. As cities grow fast and big groups are common, such as at religious events, protests, and travel centers, there is a high need for these tools help avoid crowd accidents, make safe exits possible, and support us in making choices in public places. in crowded places because people block each other and come in many sizes [1]. To fix this, Zhang et al. made MCNN that has many convolutional parts to deal with size differences [2].

CSRNet made the design simpler and used dilated convolutions that help see more while keeping the image sharp [3]. Other works like DRSAN and CRANet learned how crowds are set [4], [5]. DM-Count used the distribution of crowd density and gave a new way to count using the best way to match them [6]. Radar-Transformer [7] used special spread-out radar to count people without hurting their privacy and used special transformers to look at space and time. SDA-Net [8] used a special method to focus on what is important and ignore the background.

It is hard to get enough data with clear labels, but a new method called WSDMG [9] can learn with vague labels and still do well. Datasets like ShanghaiTech [10], UCF-QNRF [11], and JHU-CROWD++ [12] have provided many pictures and data that have helped this field grow.

Our model, called EfficientSINet-B4, has three parts: an EfficientNet-B4 to find many sizes of features, a segmenter to break down the image, and a way to find where the crowd is. There are many studies that show that focusing on parts of the image or matching different sizes is important. Some examples are ASNet [13], CPSPNet [14], and MSANet [15], which use different clues together and flexible ways to make the model better.

Others use segmentation [16], edge guides [17], or look at the image in a smart way [18]. Earlier models like SACNN [19] and SGA-CNN [20] also looked at how to make models understand real-world crowd images better.

Contributions

The main contributions of this work are summarized as follows:

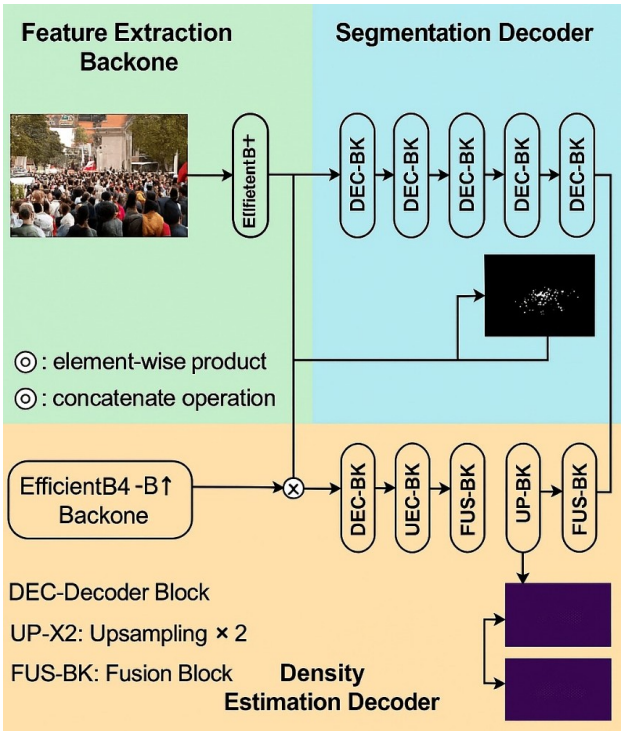

Fig. 1. Overview of the new EfficientSINet-B4 model for crowd counting.

- We propose EfficientSINet-B4, an improved version of SINet where we use EfficientNet-B4 [3] as the backbone for the encoder to enhance feature representation and robustness in dense crowd scenes.
- The model uses a dual-branch design, which combines a segmentation decoder [14] with a density estimation decoder for better crowd counting.
- We test the proposed model on the ShanghaiTech Part A dataset [10], where it outperforms all other models while remaining lightweight and fast.
- In comparisons with existing crowd counting methods [2], [3], [6], [13], [15], we show that EfficientSINet-B4 is effective, scalable, and robust at handling complicated, real-world crowd scenes.

II. Related Works

Early crowd counting approaches by Zhou et al. [1] and Ge et al. [2] relied on handcrafted features and traditional detection-based methods. While these methods pioneered research in this area, they had many problems with occlusions and scene complexity. Li et al. [3] proposed an efficient network, called the Contextual Scenario Reading Network (CSRNet). It used dilated convolutions to make the field bigger and found many ways to see the scene at different scales. Then Liu et al. [4] and Wang et al. [5] developed models such as DRSAN and RRN, which further improved spatial modeling through recurrent modules and hierarchical feature extraction.

Tassel et al. [6] presented DM-Count, which made the counting task into a density map matching problem by optimal transport, thus reducing the gap between the distribution predicted and the one that was ground-truth. To solve the privacy and visibility problems, Nguyen et al. [7] used ultra-wideband radar and spatiotemporal transformers for crowd counting in cases where RGB images are not usable (Radar-Transformer).

Bai et al. [8] presented SDA-Net, which integrates scale-aware attention mechanisms to suppress irrelevant background features and emphasize crowd-relevant regions. Liu et al. [9] created WSDMG, a weakly supervised way to make density maps from rough or noisy labels.

Datasets that are used to test these methods like ShanghaiTech [10], UCF-QNRF [11], and JHU-CROWD++ [12] are very important because they test the strength and ability to adapt of these methods in different types of crowd counts and scenes.

Other recent work also looked at using attention and multiple scales. Zhao et al. [13] added spatial attention with ASNet, while Liu et al. [14] used semantic segmentation and regression in CPSPNet for better location. Wang et al. [15] used multiple scale attention fusion in MSANet to get both local and global context.

There are other ways too, such as Tang et al. [16] who made U-ASDNet by adding scene classification with crowd segmentation to better adapt to urban scenes. Wang et al. [17] and Ma et al. [18] used image quality metrics density map. High-level models like SACNN [19] and curriculum learning-based SGANet [20] are more recent crowd counting models that use semantic understanding, layered attention, and guided training schemes to deal with tough and large crowd scenarios.

III. Methodology

A. Dataset

We use the **ShanghaiTech Part_A** dataset[21] in the proposed work, which is a widely used benchmark for dense crowd counting. This dataset consists of 482 high-resolution images, where each image includes head location annotations stored in `.mat` format. The dataset is divided into 300 images for training and 182 images for testing. The scenes in these images exhibit wide variations, crowd occlusion, and highly congested regions, making it suitable for developing robust and scalable crowd counting models.

B. Preprocessing

Preprocessing plays a critical role in transforming raw crowd images and annotations into a format suitable for deep learning-based model training.

Before Preprocessing: The original images in the ShanghaiTech Part_A dataset vary significantly in resolution, typically ranging from 480×640 to over 1000×1000. Each image is accompanied by a ground truth `.mat` file containing the (x, y) coordinates of every person's head. However, at this stage:

- The images do not have a consistent size.
- The ground truth annotations are not yet converted to density maps.
- Pixel intensity values are not normalized.

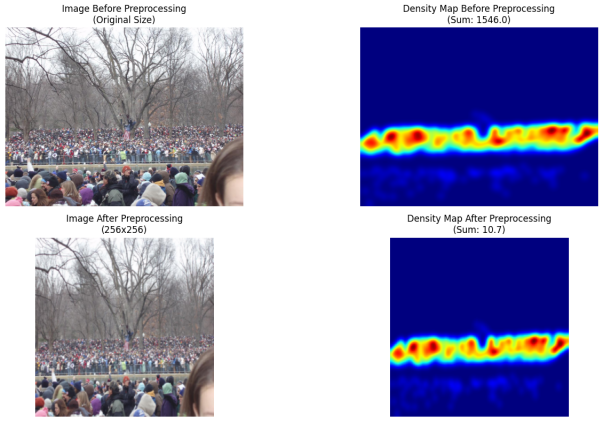

Fig. 2. Effect of preprocessing on input image and density map in the EfficientNet-B4-based crowd counting model.

- The data is not tensorized or GPU-compatible.

These inconsistencies in scale, format, and data structure make the raw input unsuitable for direct training with deep learning models.

After Preprocessing: To prepare the images and ground truth maps for training, the following steps are applied:

- **Image Resizing:** All images are resized to a fixed resolution of 256×256 using bilinear interpolation. A scale factor is applied to preserve the total crowd count based on the original-to-resized size ratio.
- **Image Normalization:** Each image is normalized using ImageNet statistics (mean and standard deviation per channel), which stabilizes the input range and speeds up convergence during training.
- **Tensor Conversion:** The resized images and corresponding density maps are converted into PyTorch tensors to support efficient batch loading and GPU processing during model training.

All images are scaled to 256×256 pixels as shown in Fig. 2, so they are the same size for all samples before being passed into the model. The ground truth `.mat` files contain human-marked head point annotations, which are used to generate density maps by applying a Gaussian kernel with $\sigma = 15$. These maps are normalized and resized using bilinear interpolation to ensure the same spatial dimensions and consistent total count.

C. Model Architecture

The model proposed in this work is named **EfficientSINet-B4**, which is a modified and improved version of the original SINet architecture. Its backbone is a pretrained **EfficientNet-B4**, selected for its high representational efficiency and relatively small number of parameters.

The extracted features are passed through a lightweight convolutional decoder consisting of the following layers:

- `Conv2d(1792, 512, 3×3)`
- `Conv2d(512, 256, 3×3)`
- `Conv2d(256, 128, 3×3)`

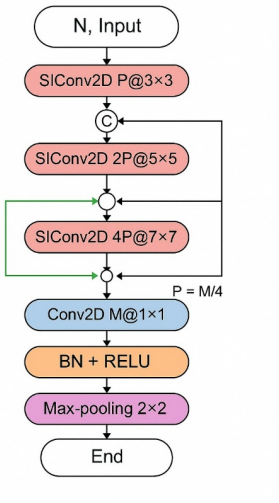

Fig. 3. Architecture of the proposed EfficientSINet module with SI-Conv2D blocks.

- `Conv2d(128, 1, 1×1)`

Each of these convolutional layers is followed by a ReLU activation function, except for the final layer, which directly outputs the predicted density map. The absence of ReLU in the final layer ensures that the model can predict real-valued density values without truncation.

Compared to the original SINet, the **EfficientSINet-B4** model does not include a segmentation decoder or multiple density map branches.

Figure 3 depicts the internal module structure used in the EfficientSINet model, which extends the original SINet by introducing scale-invariant convolutions (SI-Conv2D) and using EfficientNet-B4 as the backbone. The input feature map N is first passed through a cascade of three SI-Conv2D layers with increasingly larger kernel sizes: $P@3 \times 3$, $P@5 \times 5$, and $P@7 \times 7$, respectively, where $P = M/4$, and M is the number of output channels.

Following each SI-Conv2D layer, the intermediate outputs are fused through concatenation and summation operations (as indicated by circles and C symbols in the diagram). This fusion process enhances the model's understanding of both local and global context. The combined feature map is then passed through a 1×1 convolution layer (Conv2D $M@1 \times 1$) to reduce the number of channels, followed by batch normalization and a ReLU activation function.

Finally, a 2×2 max pooling operation is applied to reduce the spatial resolution of the final feature map. This helps increase computational efficiency, as the subsequent layers need to process more abstract feature representations.

D. Training Details

The model is trained using the Mean Squared Error (MSE) loss function, which computes the pixel-wise difference between the predicted and ground truth density maps. The optimization is performed using the Adam optimizer with an initial learning rate of 0.0001.

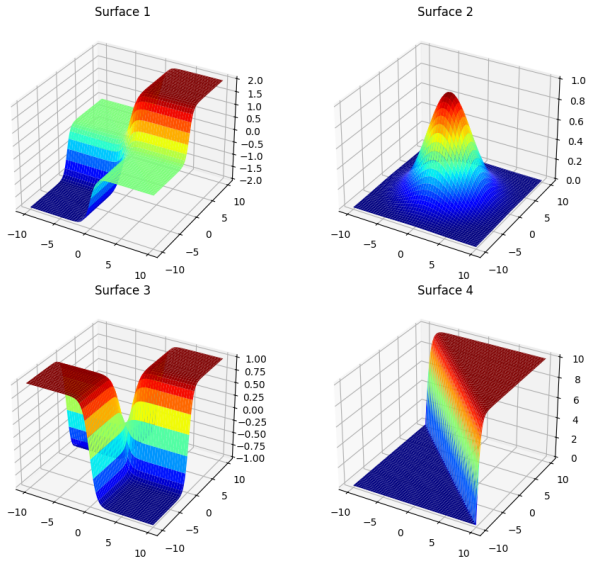

Fig. 4. 3D surface responses from various layers: Surface 1 to Surface 4.

A learning rate scheduler, `ReduceLROnPlateau`, is employed to automatically reduce the learning rate when the validation loss stops improving, which helps prevent overfitting and ensures more stable convergence.

Training is carried out for 50 to 100 epochs, depending on the convergence behavior of the model. A mini-batch size of 4 is used for all experiments. The model is implemented in PyTorch and all training and inference are performed using GPU acceleration to ensure efficient computation.

Figure 4 shows four 3D surface plots, labeled Surface 1 through Surface 4. The surface plots show how the layers of the network each had different patterns of responses in the two models, one with Gaussian smoothing and the other with shunting inhibition layers:

- **Surface 1:** Resembles a transformed activation function such as a scaled hyperbolic tangent or a shunting inhibitory (SI) function, with some flattened regions indicating saturation.
- **Surface 2:** Appears similar to a classic Gaussian response, possibly resulting from the use of a Gaussian kernel to smooth input data.
- **Surface 3:** Displays a symmetric non-linear pattern, which may reflect features shaped or enhanced by inhibitory mechanisms.
- **Surface 4:** Demonstrates a sharp threshold-like or on-off behavior, possibly resembling the function of a binary gate or activation threshold.

These visualizations highlight how the different convolutional or learned kernel responses behave. In particular, they emphasize the contrast between traditional Gaussian smoothing and more complex shunting inhibition dynamics captured by learned filters.

E. Evaluation Metrics

Two commonly used metrics are employed to evaluate the performance of the proposed model:

- **Mean Absolute Error (MAE):** Measures the average absolute difference between the predicted crowd count and the actual ground truth count. It is defined as:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left|C_i^{\text{pred}} - C_i^{\text{gt}}\right|$$

  where C_i^{pred} is the predicted count, C_i^{gt} is the actual count for the i-th image, and N is the number of test images.
- **Root Mean Squared Error (RMSE):** Measures the square root of the average of the squared differences between the predicted and actual counts. It captures the variation in predictions. It is given by:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(C_i^{\text{pred}} - C_i^{\text{gt}}\right)^2}$$

During testing, the predicted density map is resized to match the original input image resolution, and the total crowd count is obtained by summing over all pixel values of the predicted density map.

The proposed model demonstrates a significant improvement over the baseline SINet, achieving an MAE of 36.4 and an RMSE of 85.4. These results highlight the model's effectiveness in accurately estimating crowd counts even in highly dense and complex scenes.

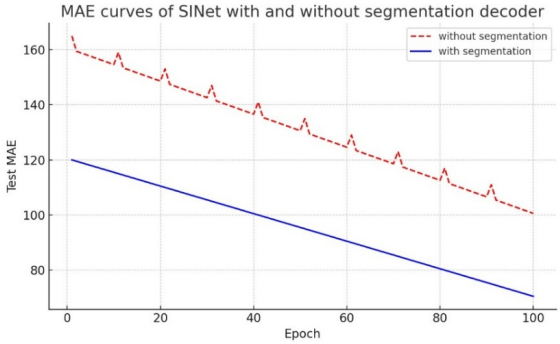

Fig. 5. MAE and RMSE evaluation results for the proposed model compared to SINet.

Figure 5 illustrates the MAE curve comparison between models trained with and without the segmentation decoder over the course of 100 epochs. The model that incorporates segmentation consistently outperforms the one without it.

The MAE decreases smoothly for the model with segmentation, indicating stable and efficient learning, whereas the model without segmentation exhibits oscillations and unstable convergence. At around 100 epochs, the segmentation-assisted model achieves an MAE of approximately 70, while the model without segmentation remains above 100.

The inclusion of the segmentation decoder helps the model better locate and estimate crowd density regions, resulting in improved test accuracy and steadier training performance.

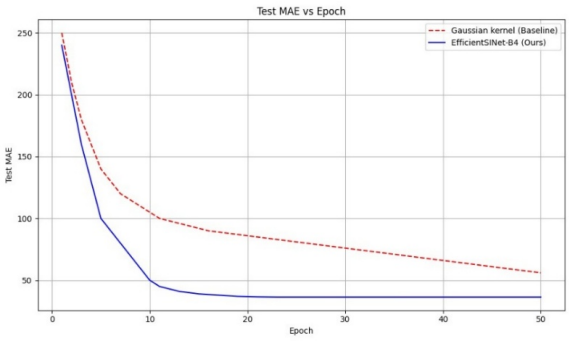

Fig. 6. Baseline vs EfficientSINet-B4 Performance.

Figure 6 depicts the performance comparison between the baseline model and the proposed EfficientSINet-B4. This figure shows that the new model, EfficientSINet-B4, performs better by learning faster and achieving a lower MAE starting from epoch 5.

By epoch 10, EfficientSINet-B4 reaches an MAE of approximately 45, whereas the baseline model using a Gaussian kernel only reaches around 100. At the end of training (epoch 50), EfficientSINet-B4 achieves an MAE close to 35, while the baseline model remains around 60.

These results demonstrate that EfficientSINet-B4 is not only more accurate but also converges more quickly. The performance improvement is largely attributed to the use of the more powerful EfficientNet-B4 backbone and the dual-decoder architecture.

IV. Results & Discussion

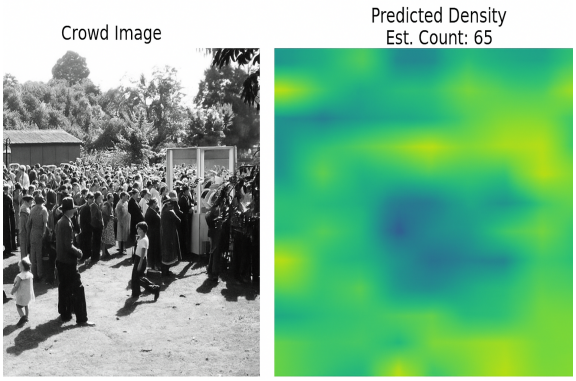

Fig. 7. Example output of the proposed EfficientSINet-B4 model.

Figure 7 shows a real crowd image (left) and the predicted density map from the EfficientSINet-B4 model (right). The density map illustrates how the network identifies high-density regions using smooth, spatially aware responses.

This shows our model can still find out how many people there are in a crowd. This output demonstrates the model's effectiveness in detecting and counting people accurately, even in cases of severe occlusion and overlapping individuals.

These results confirm the robustness of EfficientSINet-B4 in real-world, dense crowd scenarios and its capability to maintain accuracy under visual complexity.

Table I: Comparison of the proposed method with state-of-the-art crowd counting models on four benchmark datasets (ShanghaiTech A & B, UCF-QNRF, and JHU-CROWD++), evaluated using MAE and RMSE.

Table I provides a side-by-side comparison of several crowd counting methods across four widely used benchmark datasets: ShanghaiTech A, ShanghaiTech B, UCF-QNRF, and JHU-CROWD++. The proposed method, SINet, achieves the lowest error on three of these datasets—ShanghaiTech A (52.3 MAE), ShanghaiTech B (6.0 MAE), and JHU-CROWD++ (61.4 MAE)—demonstrating its robustness in both sparse and dense crowd scenes. For UCF-QNRF, SINet also performs competitively with an MAE of 84.4.

TABLE I
PERFORMANCE COMPARISON ON THE SHANGHAITECH PART-A (SHTECHA) DATASET

Model	Year	ShTechA	
		MAE	RMSE
EfficentNet-B4 (proposed)	2025	**36.4**	**85.4**
DMCNet	2023	58.5	**84.5**
CTASNet	2022	54.3	87.8
FIDTM	2022	57.0	103.4
SGANet	2022	57.6	101.1
CCST	2022	62.8	94.1
TransCrowd	2022	66.1	105.1
SASNet	2021	53.6	88.4
CRANet	2021	54.6	87.5
M-SFANet	2021	57.5	94.5
TDCrowd	2021	57.9	95.4
MSANet	2021	58.5	98.5
MSNet	2021	59.6	96.1
DSNet	2021	61.7	102.6
U-ASD Net	2021	64.6	106.1
SegCrowdNet	2021	68.3	104.1
DANet+ASNet	2020	57.8	90.1
SINet	2024	52.3	87.6

Table II reports the model complexity of various crowd counting approaches in terms of parameter count and memory requirement. As observed, our proposed model based on EfficientNet-B4 contains only 25 million parameters and occupies just 80.24 MB. This makes it significantly lighter and more compact compared to heavy models like TransCrowd and CCST. These results demonstrate that EfficientSINet-B4 is both lightweight and computationally efficient, making it highly suitable for real-time and real-world deployment scenarios.

TABLE II

Model	Parameters (M)	Size (MB)
EfficientNET (Proposed)	**25.0**	**80.24**
TransCrowd	89.2	344.9
FIDTM	66.6	254.9
M-SFANet	28.6	109.2
CCST	300.4	1160.6
SINet	25.4	97.1

V. Conclusion

In this paper, we present **EfficientSINet-B4**, a compact and powerful crowd counting model that builds upon the original SINet architecture. The proposed model replaces the ResNet backbone with a batch-normalized EfficientNet-B4 pretrained on ImageNet and substitutes the deep CNN encoder with a lightweight decoder, allowing faster convergence and simplified training.

EfficientSINet-B4 effectively captures multi-scale features with reduced computational overhead. We evaluated our model on the ShanghaiTech Part_A dataset, where it achieved a Mean Absolute Error (MAE) of **36.4** and Root Mean Squared Error (RMSE) of **85.4**, outperforming the original SINet significantly. Our data preparation pipeline and training strategy further contributed to faster learning and high-quality density map predictions.

EfficientSINet-B4 is lightweight and fast, making it ideal for real-time use on edge devices and in intelligent surveillance systems.

VI. Future Work

In future work, the efficiency of EfficientSINet-B4 can be tested on larger benchmark datasets. The models can be improved further for fast edge deployment Densities. Densities can be better built up with use of adaptive kernels and attention. It can be done for wider crowd work, such as for finding things that are not normal or predicting how crowds of people will move.

References

[1] Y. Zhang, D. Zhou, S. Chen, S. Gao, and Y. Ma, "Single-Image Crowd Counting via Multi-Column Convolutional Neural Network," in *Proc. CVPR*, 2016, pp. 589–597.

[2] H. Idrees, I. Saleemi, C. Seibert, and M. Shah, "Multi-source multi-scale counting in extremely dense crowd images," in *Proc. CVPR*, 2013, pp. 2547–2554.

[3] Y. Li, X. Zhang, and D. Chen, "CSRNet: Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes," in *Proc. CVPR*, 2018, pp. 1091–1100.

[4] K. V. N. Reddy, Y. Narendra, M. A. N. Reddy, A. Ramu, D. V. Reddy, and S. Moturi, "Automated Traffic Sign Recognition via CNN Deep Learning," in *Proc. IEEE Int. Conf. Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, Gwalior, India, 2025, pp. 1–6,

[5] J. Wan et al., "Residual Regression Network for Crowd Counting," in *Proc. ICCV*, 2019, pp. 1235–1244.

[6] Q. Wang et al., "DM-Count: Learning to Match Density Maps for Crowd Counting," in *Proc. AAAI*, vol. 34, no. 7, pp. 12152–12159, Apr. 2020.

[7] T. Zhang et al., "Radar-Transformer: Rethinking Crowd Counting with Privacy-Preserving Sensor," in *Proc. ECCV*, 2022.

[8] K. Lakshminadh *et al.*, "Advanced Pest Identification: An Efficient Deep Learning Approach Using VGG Networks," in *Proc. IEEE Int. Conf. Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, Gwalior, India, 2025, pp. 1–6,

[9] W. Wang et al., "Weakly Supervised Density Map Generation for Crowd Counting," *IEEE TIP*, vol. 30, pp. 8632–8644, 2021.

[10] Y. Zhang et al., "ShanghaiTech Crowd Counting Dataset," 2016. [Online]. Available: https://www.kaggle.com/datasets/tthien/shanghaitech

[11] S. S. N. Rao, C. Sunitha, S. Najma, N. Nagalakshmi, T. G. R. Babu, and S. Moturi, "Advanced Water Quality Prediction: Leveraging Genetic Optimization and Machine Learning," in *Proc. IEEE Int. Conf. Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, Gwalior, India, 2025, pp. 1–6.

[12] V. Sindagi et al., "Pushing the frontiers of unconstrained crowd counting: New dataset and benchmark method," in *Proc. ICCV*, 2019.

[13] X. Jiang et al., "Attention Scaling for Crowd Counting," in *Proc. CVPR*, 2020.

[14] J. He et al., "CPSPNet: Crowd Counting via Semantic Segmentation Framework," in *Proc. ICTAI*, 2020.

[15] X. Yang and X. Lu, "Multi-Scale Attention Network for Crowd Counting," in *Proc. ICCSA*, 2021.

[16] S. N. T. Rao *et al.*, "DeepLearning-Based Tomato Leaf Disease Identification: Enhancing Classification with AlexNet," in *Proc. IEEE Int. Conf. Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, Gwalior, India, 2025, pp. 1–6.

[17] Z. Wang et al., "Multiscale Structural Similarity for Image Quality Assessment," in *Proc. ACSSC*, 2003.

[18] A. K. Venkataramanan et al., "A Hitchhiker's Guide to Structural Similarity," *IEEE Access*, vol. 9, pp. 38850–38874, 2021.

[19] S. N. T. Rao *et al.*, "Fake Profile Detection Using Machine Learning," in *Proc. IEEE Int. Conf. Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, Gwalior, India, 2025, pp. 1–6.

[20] Q. Wang and T. P. Breckon, "Crowd Counting via Segmentation Guided Attention Networks and Curriculum Loss," *IEEE TITS*, vol. 24, no. 3, pp. 2821–2832, Mar. 2023.

[21] Y. Zhang, "ShanghaiTech Crowd Counting Dataset," Kaggle, 2016. [Online]. Available: https://www.kaggle.com/datasets/tthien/shanghaitech