A Thesis Submitted to the Sylhet Engineering College for the Degree of
Bachelor of Science in **Computer Science and Engineering**

Mobile-Based Mosquito Larvae Classification System Using Deep Learning

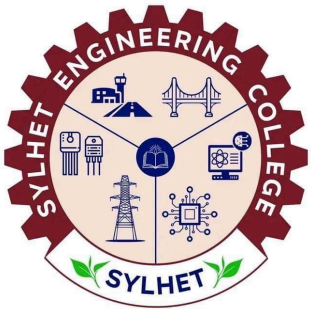
Submitted by

Md. Bodrul Islam
Reg.no: 2019331507
4th Year 2nd Semester
Department of Computer Science and Engineering
Sylhet Engineering College

MD. Fuad Khan
Reg.no: 2019331513
4th Year 2nd Semester
Department of Computer Science and Engineering
Sylhet Engineering College

Supervised by

Md Lysuzzaman
Lecturer
Department of Computer Science and Engineering
Sylhet Engineering College



Department of Computer Science and Engineering
Sylhet Engineering College, Sylhet

Affiliated with

Shahjalal University of Science and Technology

Sylhet, Bangladesh
22th July 2025

Candidates' Declaration

This is to certify that the work presented in this thesis, titled **"Mobile-Based Mosquito Larvae Classification System Using Deep Learning"**, is the outcome of the investigation and research carried out by us under the supervision of Md Lysuzzaman.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree or diploma.

Md. Bodrul Islam
Reg No: 2019331507

MD. Fuad Khan
Reg No: 2019331513

Sylhet
Bangladesh
22 July 2025

Recommendation Letter from Thesis Supervisor

It is with great satisfaction that I confirm the successful completion and final submission of the thesis titled **"Mobile-Based Mosquito Larvae Classification System Using Deep Learning"**, carried out by the following group of students in fulfillment of the requirements for the Bachelor of Science in Computer Science and Engineering degree, submitted in July 2025..

Group Members:

1) **Md. Bodrul Islam**
2) **MD. Fuad Khan**

The group has exhibited outstanding dedication, technical proficiency, and problem-solving skills throughout the entire project lifecycle — from conceptualization to the final deployment of a fully functional mobile-friendly web application for mosquito larvae classification. Their work involved original data collection, meticulous annotation, advanced preprocessing, and the training and evaluation of multiple deep learning models including EfficientNet, MobileNet, ResNet, Vision Transformer, and YOLOv8. They also incorporated CLIP-based open-set filtering to enhance robustness and deployed the application on Streamlit Cloud, ensuring accessibility and real-world applicability.

Their research presents a meaningful contribution to public health and vector surveillance by enabling accurate, real-time larval identification using AI, which can assist in mosquito-borne disease prevention. The system's practical deployment and strong experimental results further reflect their capability for real-world innovation.

Given the significance of their work and their commitment to quality and impact, I wholeheartedly recommend their project for academic and professional recognition.

Supervisor:

_____

Md Lysuzzaman

Lecturer

Department of Computer Science and Engineering
Sylhet Engineering College

Certificate of Acceptance

The thesis is titled **"Mobile-Based Mosquito Larvae Classification System Using Deep Learning"** submitted by **Md. Bodrul Islam** and **MD. Fuad Khan**; Student ID. **2019331507** and **2019331513**; Session **2019-20**, to the Department of Computer Science and Engineering , Sylhet Engineering College, has been accepted as satisfactory in partial fulfillment of the requirement for the Degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents.

BOARD OF EXAMINERS

Internal
Nayan Kumar Nath
Lecturer
Department of Computer Science and Engineering
Sylhet Engineering College, Sylhet

Internal
Md Lysuzzaman
Lecturer
Department of Computer Science and Engineering
Sylhet Engineering College, Sylhet

Internal
Md. Rasel Ahmed
Assistant Professor
Department of Computer Science and Engineering
Sylhet Engineering College, Sylhet

Internal
Md. Nagrul Islam
Assistant Professor
Department of Computer Science and Engineering
Sylhet Engineering College, Sylhet

Chairman
Md. Abu Naser Mojumder
Head
Department of Computer Science and Engineering
Sylhet Engineering College, Sylhet

Member (External)
Dr. Mohammad Shahidur Rahman
Professor
Department of Computer Science and Engineering
Shahjalal University of Science and Technology

Acknowledgement

We sincerely express our gratitude to the Department of Computer Science and Engineering at Sylhet Engineering College for providing the necessary support and resources to initiate and advance our research.

We are especially thankful to our supervisor, Md Lysuzzaman, Lecturer, Department of CSE, Sylhet Engineering College, for his invaluable guidance, continuous encouragement, and dedicated supervision throughout our research journey. His insightful feedback and expertise have been crucial in shaping the progress of our work thus far.

Additionally, we acknowledge the contributions of previous researchers and scholars, whose work has provided a strong foundation and valuable insights for our study. Their research has been an essential source of inspiration and reference.

As we continue refining our work through further data processing, model training, and analysis, we remain grateful for the support and encouragement from our families, friends, and peers, who have been with us throughout this journey.

Abstract

Dengue and other mosquito-borne diseases pose a severe public health threat, with Bangladesh experiencing a rising dengue-related death rate in recent years. Traditional mosquito larvae classification methods require expert supervision and laboratory analysis, making them time-consuming, resource-intensive, and impractical for large-scale monitoring. To address this, we propose a mobile-based mosquito larvae classification system that leverages deep learning models to classify larvae from mobile phone-captured images. Our system aims to automatically detect mosquito larvae in an image and classify them into Aedes, Anopheles and Culex aiding in early dengue outbreak classification and vector control efforts.

We employ a robust data collection and preprocessing pipeline to enhance model performance, integrating state-of-the-art deep learning architectures for accurate classification. This system is designed to be accessible, efficient, and scalable, enabling real-time monitoring on mobile devices, tablets, or laptops. By providing a cost-effective and rapid surveillance tool, our approach contributes to early intervention strategies, supporting public health authorities in Bangladesh and other dengue-endemic regions.

To automate the mosquito larvae classification we trained deep neural based models named **Vission Transformer(ViT), YOLOv8, MobileNetV3, ResNet50, EfficientNetB0, MobileViT and Ensemble(ResNet50+MobileViT)** along with noticeable accuracy. Finally we built a web app where we deploy our trained model so that users can easily use our models to classify mosquito larvae.

Contents

Chapter 1
INTRODUCTION

1.1: Background

Mosquito-borne diseases, particularly dengue fever, have emerged as major public health concerns in Bangladesh and many other tropical regions. Dengue outbreaks have shown an alarming upward trend in recent years, contributing to a rising number of infections and fatalities. A key strategy in controlling mosquito populations and preventing disease transmission is the early detection and classification of mosquito larvae. Traditionally, this process is conducted manually by entomologists under microscopes in laboratory environments—a process that is not only time-consuming and expensive but also dependent on expert availability and limited in scale.

With the increasing adoption of Artificial Intelligence (AI) and mobile technologies, the healthcare and environmental sectors are witnessing a shift from conventional systems to smarter, more automated approaches. In this context, developing an AI-driven, mobile-based solution for automatic mosquito larvae classification has the potential to significantly improve the efficiency and scalability of dengue prevention strategies.

1.2: Problem Statement

The existing mosquito larvae classification system relies heavily on manual inspection by trained professionals, making it impractical for rapid, large-scale deployment, especially in rural or under-resourced areas. This traditional approach is not only slow and labor-intensive but also susceptible to human error. Additionally, the lack of automation hinders timely intervention during outbreak seasons, resulting in delayed control measures and increased disease spread. Hence, there is a critical need for an automated, accessible, and accurate solution that can classify mosquito larvae directly from field-captured images—enabling proactive public health responses.

1.3: Motivation

Our motivation stems from the urgent need to contribute meaningfully to public health initiatives in Bangladesh, a country frequently hit by dengue epidemics. As computer science students and aspiring AI engineers, we sought to apply our technical skills to a problem that goes beyond conventional academic boundaries. By leveraging cutting-edge deep learning techniques and mobile accessibility, we aim to democratize mosquito larvae detection, making it available to anyone with a smartphone. Our approach not only addresses a pressing issue but also demonstrates how AI can be applied to solve real-world problems in resource-limited settings.

1.4: Research Area

This thesis lies at the intersection of computer vision, mobile health (mHealth), and public health informatics. It draws from domains such as deep learning, image classification, transfer learning, model deployment, and mobile application development. Our work particularly focuses on the application of state-of-the-art convolutional and transformer-based models for biological image classification tasks.

1.5: Research Aim

The aim of this research is to develop a mobile-based mosquito larvae classification system using deep learning techniques that can accurately identify Aedes, Anopheles, and Culex larvae from images captured by mobile devices. The system is intended to support early dengue intervention and vector surveillance efforts through fast, automated, and reliable identification.

1.6: Research Objectives

To fulfill the above aim, the specific objectives of our research are:

1. To collect and preprocess a high-quality dataset of mosquito larvae images, covering Aedes, Anopheles, and Culex species.

2. To train and evaluate various deep learning models, including CNNs (ResNet50, MobileNetV3, EfficientNetB0), Transformers (ViT, MobileViT), YOLOv8, and a hybrid ensemble model.

3. To analyze and compare model performance using evaluation metrics such as accuracy, precision, recall, F1-score, confusion matrix, and ROC-AUC curves.

4. To design and develop a user-friendly web interface for deploying the best-performing model.

5. To enable end-users to classify mosquito larvae images directly from mobile phones or web platforms, promoting real-time surveillance.

1.7: Thesis Organization

This thesis is structured into the following chapters:

- **Chapter 1: Introduction** – Presents the background, problem context, and motivation, followed by the research aim and objectives.
- **Chapter 2: Background Study –** Provides a detailed review of ML, deep learning techniques, computer vision and domain knowledge.
- **Chapter 3: Literature Review** – Discusses existing work and related research in the domain of mosquito larvae classification and deep learning-based biological detection systems.
- **Chapter 4: Methodology** – Details the data collection, preprocessing, model selection, training procedures, and evaluation metrics.
- **Chapter 5: Implementation:** – Presents the performance of the trained models and discusses findings from the experiments.
- **Chapter 6: Result-Discussion And Performance Analysis** – Describes the development of the web application and deployment of the classification system.
  Chapter 7: Conclusion – Summarizes key contributions and how the people have benefited from our work.
- **Chapter 8: Future Work –** Suggest directions for future research and propose the creation of a dedicated innovation hub or collaboration zone for research and AI enthusiasts to exchange ideas, build impactful solutions, and drive continuous advancements

Chapter 2
Background Study

2.1: Machine Learning

2.1.1: Definition of Machine Learning

• “Machine learning is the science of getting computers to act without being explicitly programmed.”—Stanford University
• It’s a subset of AI which uses statistical methods to enable machines to improve with experience
• It enables a computer to act and take data driven decisions to carry out a certain task
• These programs or algorithms are designed in such a way that they can learn and improve over time when exposed to new data

2.1.2: What the heck is machine learning?

The whole concept of machine learning is figuring out ways in which we can teach a computer to perform a task without needing to provide explicit instructions. Another way to think about it is that we're trying to "program" intuition in a computer. You and I can look at an email and easily discern whether or not it's spam, but how do you get a computer to do such a task? You could construct a huge convoluted logic infrastructure of "if.. then.." statements to sort out the spam emails, but it would be a pain to construct and probably wouldn't work too well. Instead, the machine learning approach is to equip the computer with skills to learn on its own and feed it a bunch of examples. Machine learning is exploding as a field right now as people are realizing a multitude of tasks that we can teach computers to perform by feeding it large datasets.
Consider an example to clarify the concept of ML. Suppose we have a data set of selling ice cream depending on the temperature of a supper mall,

Temperature (degree Celsius)	Selling Ice-Cream (Tk)
20	1000
25	1200
30	1600
32	1800
35	2000
38	2500
40	3300

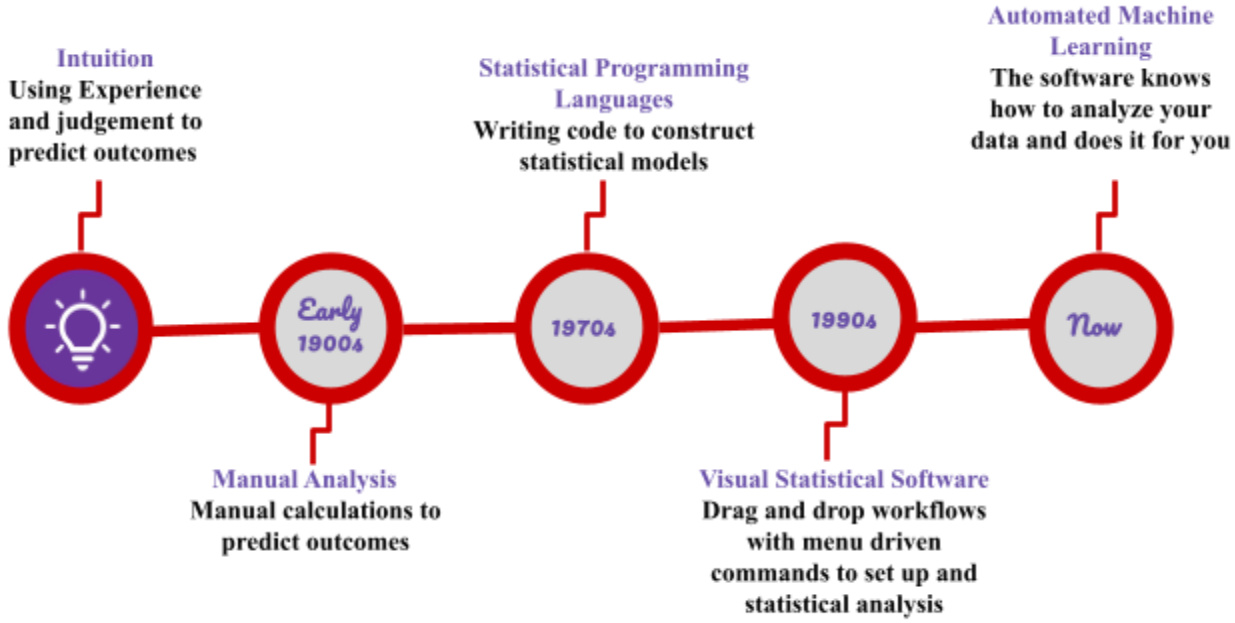
Now if we ask a question to a person, that is, at 37 degrees Celsius what is the selling price of ice cream in that mall? The person can predict the answer. But if we ask a machine the same

question, can it predict the answer? Yes, it can predict if we trained this machine. But how can we train? The process of training is called ML.
Traditional programming uses known algorithms to produce results from the data(Data + Algorithms = Results). On the other hand Machine learning creates new algorithms from data and results (Data + Results = Algorithms)

2.1.3: Normal Computer vs ML

The difference between normal computer software and machine learning is that a human developer hasn't given codes that instructs the system how to react to a situation, instead it is being trained by a large amount of data.

2.1.4: Quick history of ML



2.1.5: The general machine learning framework

Machine learning isn't one singular tool, it's an entire field of tools - each with their own strengths and weaknesses. The way we allow computers to learn is by providing it with a model which serves as the learning framework. The image below is a guide from scikit-learn (a collection of Python implementations of various machine learning algorithms and functions) on selecting the proper model for your machine learning application. Part of being a machine learning expert is knowing which tool to use.

Once you've selected your model, you typically follow the same general procedure.

1. Preprocess your data so that it will feed properly into your model.
2. Construct your model.
3. Train your model on a dataset and tune all relevant parameters for optimal performance.
4. Evaluate your model and determine its usefulness.

2.1.6: Types of Machine Learning

Rather than explicitly programming the computer to complete a task, we're providing a ton of examples (data) for the computer to learn from with the goal of completing some given task. There are times when we tell the computer what we want it to do (by training it on a dataset with predefined outputs) and there are times when we simply hand over a dataset to the computer and ask it to discover something on its own. When we train a computer by providing it with data that has predefined outputs, we call this **supervised learning**. An example of this would be training a spam filter for your email by giving the algorithm a huge set of example emails which are labeled as "spam" or "not spam". When we simply hand over a dataset to the computer and ask it to find something interesting, this is known as **unsupervised learning**, since we're not telling the computer what output we'd like to see. Unsupervised machine learning techniques are ideal because it takes time to label datasets for training supervised models. There's also such a thing as **semi-supervised learning** where we leverage a small dataset of labeled observations combined with a much larger dataset of unlabeled observations to use in training a model to perform some task.

Let's dig deep into it...

Supervised Learning: In supervised machine learning there are some labeled data and we train the machine to predict the output using these labeled data.
The mapping of the input data to the output data is the objective of supervised learning.

Unsupervised Learning: Unsupervised Learning is a type of machine learning in which the algorithms are provided with data that does not contain any labels or explicit instructions on what to do with it. The goal is for the learning algorithm to find structure in the input data on its own.
To put it simply—Unsupervised Learning is a kind of self-learning where the algorithm can find previously hidden patterns in the unlabeled datasets and give the required output without any interference.
Identifying these hidden patterns helps in clustering, association, and detection of anomalies and errors in data.

Semi-Supervised Learning: A Mix of Supervised and Unsupervised Learning. Semi-Supervised Learning works by initially training the model using the labeled dataset, just like Supervised Learning. Once we get the model performing well, we use it to predict the remaining unlabeled data points and label them with the corresponding predictions.
This is followed by training the model on the full dataset, which comprises the truly labeled and "pseudo labeled" datasets.
A common example of an application of semi-supervised learning is a **text document classifier**. This is the type of situation where semi-supervised learning is ideal because it would be nearly impossible to find a large amount of labeled text documents.

Reinforcement Learning: Reinforcement learning is a feedback-based learning method, in which a machine gets a reward for each right action and a penalty for each wrong action. The machine learns automatically with these feedbacks and improves its performance.
The goal of an agent is to get the most reward points, and hence, it improves its performance.

Machine learning algorithms are capable of tasks such as classifying objects found in photos, predicting the optimal price to sell your house with regression, customer and market segmentation using clustering techniques, and much much more.

2.1.7: Steps involved while working with ML

Step	Example
1.Set the research goal.	I want to predict how heavy traffic will be on a given day.
2.Make a hypothesis.	I think the weather forecast is an informative signal.
3.Collect the data.	Collect historical traffic data and weather on each day.
4.Test your hypothesis.	Train a model using this data.
5.Analyze your results.	Is this model better than existing systems?
6.Reach a conclusion.	I should (not) use this model to make predictions, because of X, Y and Z.
7.Refine hypothesis and repeat.	Time of year could be a helpful signal.

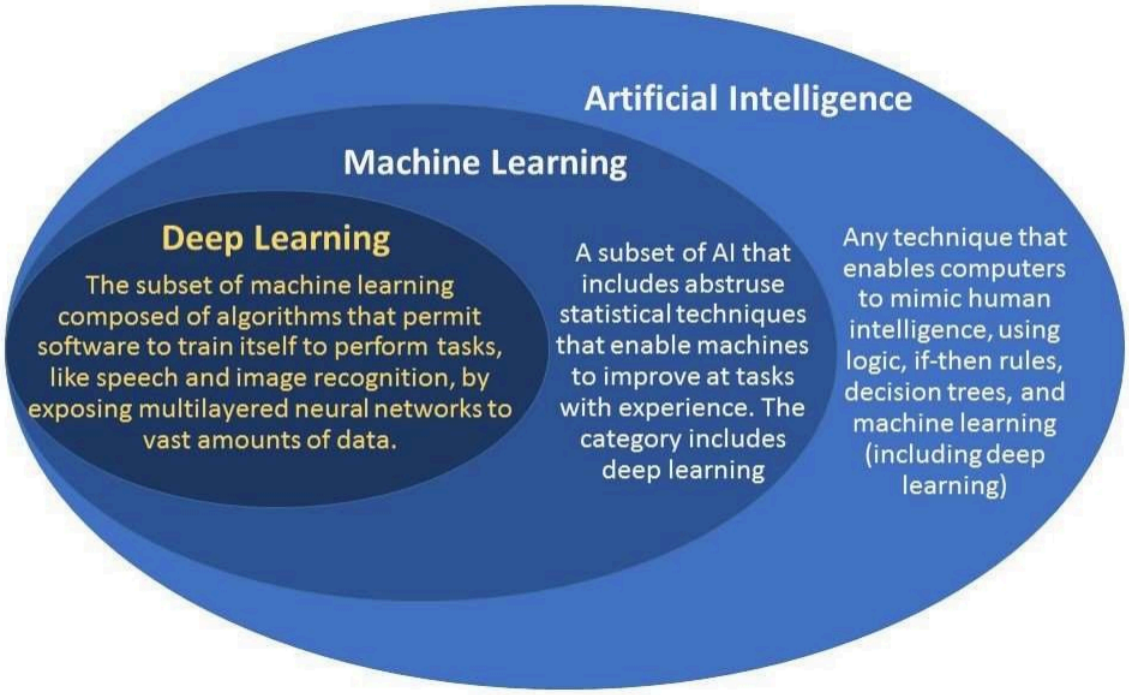
2.1.8: Identifying Good Problems for ML

- **Focus on problems that would be difficult to solve with traditional programming**
  –For example, consider Smart Reply. The Smart Reply team recognized that users spend a lot of time replying to emails and messages; a product that can predict likely responses can save user time
  –Another example is in Google Photos, where the business problem was to find a specific photo by keyword search without manual tagging.

- **Imagine trying to create a system like Smart Reply or Google Photos search with conventional programming**
  –There isn't a clear approach
  –By contrast, machine learning can solve these problems by examining patterns in data and adapting with them. Think of ML as just one of the tools in your toolkit and only bring it out when appropriate.

2.1.9: AI vs ML vs DL

Artificial Intelligence is a concept of creating intelligent machines that stimulates human behavior whereas Machine learning is a subset of Artificial intelligence that allows machines to learn from data without being programmed.



Advantages of Machine Learning

- Fast, Accurate, Efficient.
- Automation of most applications.
- Wide range of real life applications.
- Enhanced cyber security and spam detection.
- No human intervention is needed.
- Handling multi dimensional data.

Disadvantages of Machine Learning

- It is very difficult to identify and rectify the errors.
- Data Acquisition.
- Interpretation of results requires more time and space.

2.2: Deep Learning

2.2.1: What is Deep Learning?

Definition: Deep learning is a subset of machine learning that utilises multi-layered neural networks known as deep neural networks (DNNs) to mimic the intricate decision-making capabilities of the human brain.

Deep learning is a subset of machine learning that uses multi-layered neural networks, called deep neural networks, to simulate the complex decision-making power of the human brain.
—IBM
"Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks"
—Machine Learning Mastery

2.2.2: Structure of DNNs

DNNs consist of three or more layers, although most practical implementations have many more layers. These layers allow for the identification, classification, and analysis of complex phenomena, patterns, and relationships within data.
Training Process: DNNs are trained on extensive datasets to learn to recognize patterns, evaluate possibilities, and make predictions and decisions. The training process involves adjusting the network parameters based on the data to optimise performance.
Enhanced Accuracy: While single-layer neural networks can provide approximate predictions and decisions, the additional layers in DNNs enable refinement and optimization of outcomes, leading to greater accuracy in tasks such as classification and prediction.
Applications: Deep learning powers a wide range of applications and services aimed at automation, including digital assistants, voice-enabled devices, and fraud detection systems. It also underpins emerging technologies like self-driving cars and generative AI, enabling advancements in various fields.
Overall, deep learning plays a crucial role in driving innovation and automation across industries, leveraging the power of neural networks to process vast amounts of data and perform complex tasks with remarkable accuracy.

2.2.3: Deep Learning VS Machine Learning

Here are the key points differentiating deep learning from classical machine learning:

1. **Data Type and Handling**:
   - Machine learning typically works with structured, labelled data where specific features are defined and organised into tables.
   - Deep learning can handle unstructured data such as text and images without extensive preprocessing. It automates feature extraction, reducing the need for human intervention in feature engineering.

2. **Feature Extraction**:
   - In machine learning, the hierarchy of features is established manually by human experts.
   - Deep learning algorithms automatically determine important features from raw data through layers of abstraction, reducing the dependency on manual feature engineering.

3. **Training Process**:
   - In both machine learning and deep learning, training involves adjusting model parameters to fit the data. However, the mechanisms differ.
   - Machine learning algorithms often use techniques like gradient boosting or support vector machines.

- Deep learning algorithms employ gradient descent and backpropagation, adjusting the multiple layers of neural networks to optimize performance.

4. **Learning Types**:
   - Both machine learning and deep learning encompass various learning types, including supervised, unsupervised, and reinforcement learning.
   - Supervised learning in machine learning relies on labelled datasets for prediction or classification tasks.
   - Unsupervised learning involves identifying patterns and structures in unlabeled data.
   - Reinforcement learning is about learning to make decisions through trial and error based on rewards or penalties in an environment.

5.**Complexity and Scale**:
   - Deep learning models, particularly deep neural networks, are capable of handling highly complex tasks and large-scale datasets more efficiently compared to traditional machine learning models.
   - This scalability and ability to handle intricate patterns make deep learning well-suited for tasks like image and speech recognition, natural language processing, and other domains with high-dimensional data.

In summary, while both machine learning and deep learning involve training models to make predictions or identify patterns, deep learning stands out for its ability to handle unstructured data, automate feature extraction, and tackle highly complex tasks with large-scale datasets.

2.2.4: How Deep Learning Works?

Deep learning neural networks, or artificial neural networks, attempts to mimic the human brain through a combination of data inputs, weights, and bias. These elements work together to accurately recognize, classify, and describe objects within the data.

Deep neural networks consist of multiple layers of interconnected nodes, each building upon the previous layer to refine and optimise the prediction or categorization. This progression of computations through the network is called **Forward Propagation**. The input and output layers of a deep neural network are called visible layers. The input layer is where the deep learning model ingests the data for processing, and the output layer is where the final prediction or classification is made.

Another process called **Backpropagation** uses algorithms, like gradient descent, to calculate errors in predictions and then adjusts the weights and biases of the function by moving backwards through the layers in an effort to train the model. Together, forward propagation and backpropagation allow a neural network to make predictions and correct for any errors accordingly. Over time, the algorithm becomes gradually more accurate.

The above describes the simplest type of deep neural network in the simplest terms. However, deep learning algorithms are incredibly complex, and there are different types of neural networks to address specific problems or datasets. For example,

Convolutional neural networks (CNNs), used primarily in computer vision and image classification applications, can detect features and patterns within an image, enabling tasks, like

object detection or recognition. In 2015, a CNN bested a human in an object recognition challenge for the first time.

Recurrent neural networks (RNNs) are typically used in natural language and speech recognition applications as it leverages sequential or times series data.

2.2.5: Deep Learning Applications

Real-world deep learning applications are a part of our daily lives, but in most cases, they are so well-integrated into products and services that users are unaware of the complex data processing that is taking place in the background. Some of these examples include the following:

- **Computer Vision**
- **Natural Language Processing**
- **Time Series Analysis**
- **Robotics**

Advantages of Deep Learning

- High Accuracy
- Automated Feature Engineering
- Scalability
- Flexibility
- Continual Improvement.

Disadvantages of Deep Learning

- High Computational Requirements
- Requires Large Amounts of Labelled Data.
- Interpretability
- Overfitting
- Black-Box Nature

2.2.6: Neural Network - Heart of Deep Learning Models

A neural network is a machine learning program, or model, that makes decisions in a manner similar to the human brain, by using processes that mimic the way biological neurons work together to identify phenomena, weigh options and arrive at conclusions.

Every neural network consists of layers of nodes, or artificial neurons—an input layer, one or more hidden layers, and an output layer. Each node connects to others and has its associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Neural networks rely on training data to learn and improve their accuracy over time. Once they are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to manual identification by human experts. One of the best-known examples of a neural network is Google’s search algorithm.

Neural networks are sometimes called artificial neural networks (ANNs) or simulated neural networks (SNNs). They are a subset of machine learning, and at the heart of deep learning models.

2.3: Computer Vision

2.3.1: What is Computer Vision?

Computer vision is a field of artificial intelligence (AI) that uses machine learning and neural networks to teach computers and systems to derive meaningful information from digital images, videos and other visual inputs—and to make recommendations or take actions when they see defects or issues.

If AI enables computers to think, computer vision enables them to see, observe and understand.

Computer vision works much the same as human vision, except humans have a head start. Human sight has the advantage of lifetimes of context to train how to tell objects apart, how far away they are, whether they are moving or something is wrong with an image.

Computer vision trains machines to perform these functions, but it must do it in much less time with cameras, data and algorithms rather than retinas, optic nerves and a visual cortex. Because a system trained to inspect products or watch a production asset can analyse thousands of products or processes a minute, noticing imperceptible defects or issues, it can quickly surpass human capabilities.

Computer vision is used in industries that range from energy and utilities to manufacturing and automotive—and the market is continuing to grow.

2.3.2: How Does Computer Vision Work?

Computer vision works by enabling machines to interpret and analyze visual data, much like human vision. It requires large amounts of training data to teach models how to recognize patterns, shapes, and objects. Technologies such as image processing, deep learning, and artificial intelligence form its backbone. Machine learning, especially convolutional neural networks (CNNs), plays a crucial role in extracting and learning features from images. The learning process involves feeding labeled data to the model, allowing it to recognize hierarchical features—from simple edges to complex objects. While CNNs dominate static image recognition, recurrent neural networks (RNNs) are often used for sequential data like video analysis. Together, these methods power real-world applications such as facial recognition, self-driving cars, medical imaging, and surveillance systems.

2.3.3: Areas

Computer vision is a rapidly advancing field with numerous areas offering exciting opportunities for research, development, and application. Here are some key areas within computer vision to consider:

- Object Detection and Recognition
- Image Classification and Image Retrieval
- Image Segmentation

2.3.4: Reinforcement Learning in Computer Vision

Reinforcement learning (RL) is a machine learning paradigm where an agent learns to make sequential decisions by interacting with an environment to maximize cumulative rewards. In computer vision, RL is applied to tasks that involve interpreting and processing visual data. It is used in object detection and recognition, where an agent learns to localize and classify objects more efficiently through trial and error. RL also supports image captioning by enabling models to generate more contextually accurate and descriptive captions. In visual navigation, agents

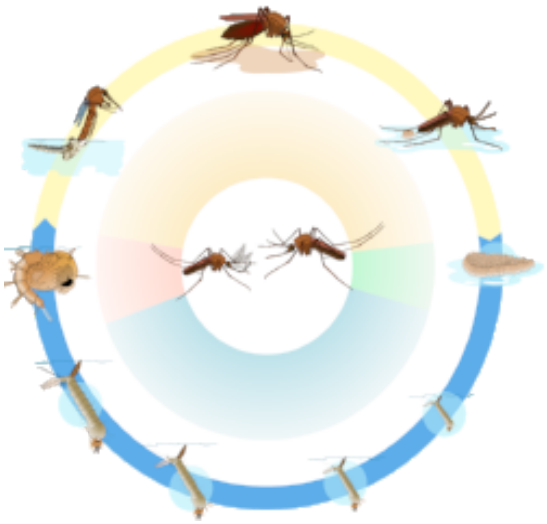
learn to move through complex environments by interpreting visual cues, which is vital for autonomous robots and self-driving cars. Furthermore, RL contributes to image enhancement and restoration by learning strategies to reduce noise, sharpen details, and improve overall image quality. It also plays a role in active learning and adaptive sampling, where models select the most informative visual data for training, thus improving performance while reducing data requirements.

2.3.5: Challenges and Considerations

- Sample Efficiency
- Reward Design
- Generalisation

2.4: Domain Knowledge (Mosquito Larvae)

2.4.1: Understanding the Life Cycle of the Mosquito



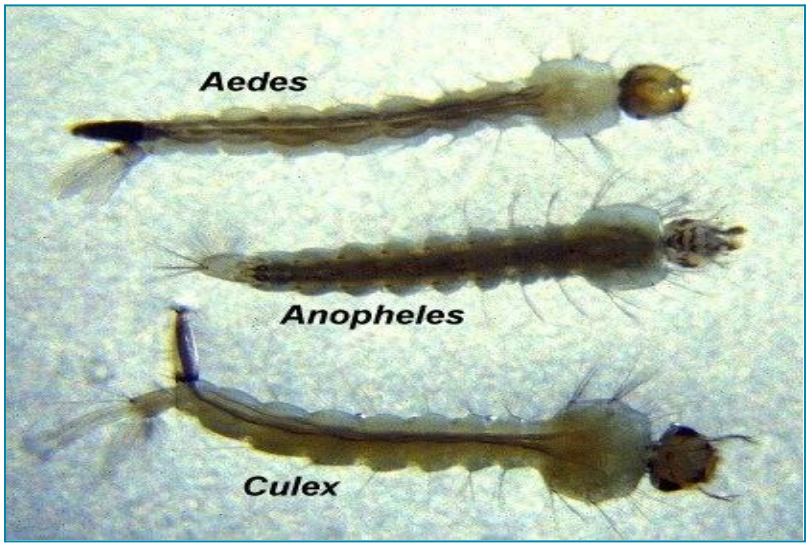
Mosquitoes (Order Diptera, Family Culicidae) are some of the most adaptable and successful insects on Earth and are found in some extraordinary places. Virtually any natural or man-made collection of water can support mosquito production. They've been discovered in mines nearly a mile below the surface, and on mountain peaks at 14,000 feet, and if you know where to look, there is a good possibility that there are mosquitoes breeding in your own backyard. Not every species of mosquito causes problems for people, but many have profoundly negative effects. Mosquitoes can be distinguished easily from other flies by the fact that they have both a long, piercing proboscis and scales on the veins of their wings. Approximately 176 species of mosquitoes are found in the United States, with more than 3,000 species known throughout the world. In the United States, only a few of these species are important as carriers of disease, but many more are important nuisance species that dramatically affect peoples' quality of life.

2.4.2: Mosquito Species and Larvae Classification

Aedes: Aedes aegypti and Aedes albopictus are the primary vectors of dengue, Zika, and chikungunya. These mosquitoes thrive in urban environments and prefer breeding in artificial containers like water storage tanks, tires, and pots. They are characterized by white markings on their legs and a lyre-shaped pattern on the thorax.

Anopheles: Anopheles mosquitoes are the primary vectors of malaria. Unlike Aedes, they typically breed in larger bodies of water such as ponds, lakes, and marshes. Anopheles larvae are distinguishable from other species by their vertical position in the water and distinctive body shape.

Culex: Culex mosquitoes are responsible for transmitting diseases like West Nile virus, Japanese encephalitis, and filariasis. They tend to breed in stagnant water, especially in polluted environments, and have a more widespread distribution compared to Aedes and Anopheles.

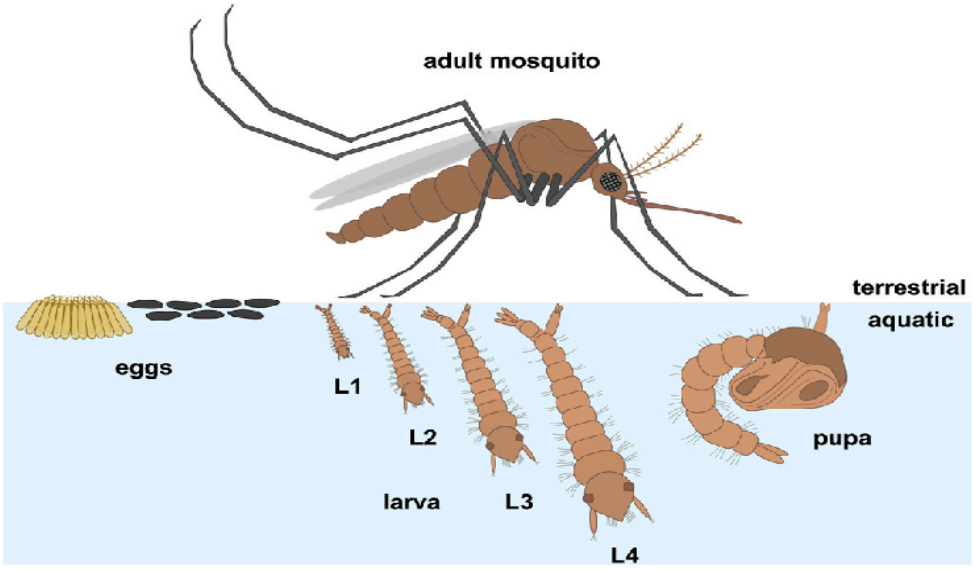


Mosquito Larvae

Reproduction and Larval Stages:

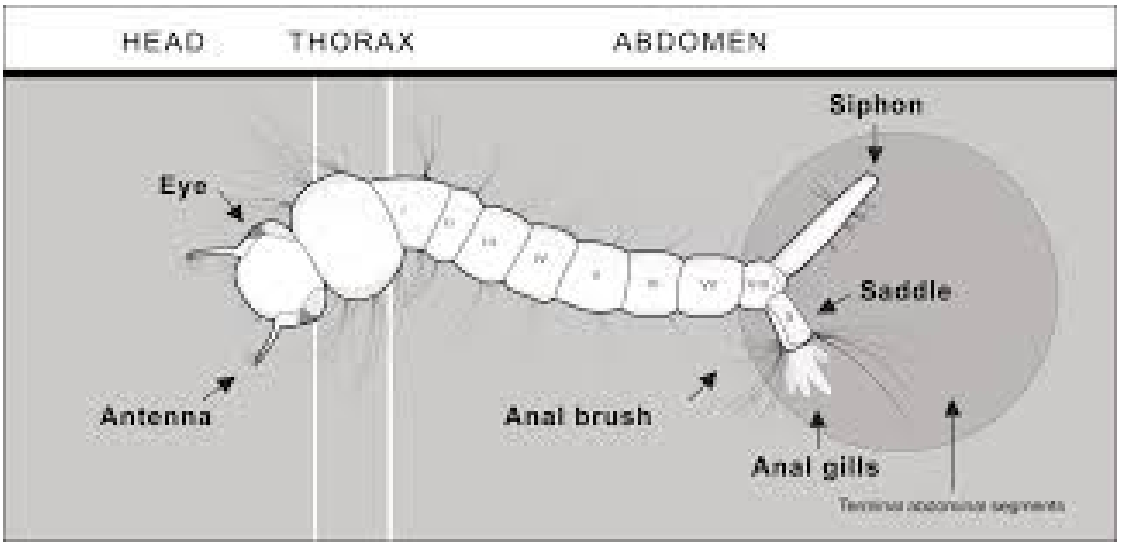
- Reproduction: Female mosquitoes lay their eggs on the water's surface or on objects that come into contact with water. Once the eggs hatch, the larvae go through four stages (L1 to L4) before becoming pupae and eventually emerging as adult mosquitoes.

- Larval Stages:
    - **L1 (First Instar):** Tiny larvae that swim near the water surface and feed on microorganisms.
    - **L2 (Second Instar):** Larger larvae with distinct body parts.
    - **L3 (Third Instar):** More developed larvae with prominent head and body segmentation.
    - **L4 (Fourth Instar):** Final larval stage before pupation.



- Morphological Differences Between Aedes, Anopheles, and Culex Larvae:
    1. **Head**:
        - Aedes Larvae: The head is **rounded** and slightly wider, with visible mouth brushes used for feeding.
        - Anopheles Larvae: The head is more **pointed** and narrower compared to Aedes.
        - Culex Larvae: The head is **somewhat rounded**, similar to Aedes but slightly smaller.

    2. **Thorax**:
        - Aedes Larvae: The thorax is **larger and more robust**, without bristles or tufts.
        - Anopheles Larvae: The thorax is **more flattened**, making them appear broader.
        - Culex Larvae: The thorax is **intermediate in shape**, not as flat as Anopheles but not as robust as Aedes.



    3. **Abdomen**:
        - Aedes Larvae: The abdomen is **cylindrical**, with **prominent comb scales** on certain segments.
        - Anopheles Larvae: The abdomen is **flattened**, often with **small palmate hairs** used for floating.
        - Culex Larvae: The abdomen is **elongated** and relatively smooth, lacking large bristles.

    4. **Siphon (Breathing Tube)**:
        - Aedes Larvae: Have a **longer and thinner siphon**, allowing them to breathe at the water surface.
        - Anopheles Larvae: Lack a siphon and breathe **horizontally**, keeping their bodies parallel to the water surface.

- Culex Larvae: Have a **moderately long and uniform siphon**, shorter than Aedes but longer than Anopheles.

5. **Saddle (9th Abdominal Segment Plate):**
   - Aedes Larvae: Have an incomplete saddle, meaning it does not fully encircle the 9th abdominal segment. It often features comb-like scales or spines and appears darker and more hardened than the rest of the abdomen.
   - Anopheles Larvae: Do not have a saddle; instead, the 9th segment is soft and covered with palmate hairs, making them unique among mosquito larvae.
   - Culex Larvae: Have a complete saddle, which fully encircles the 9th abdominal segment. It is smooth and uniform, typically lighter in color compared to Aedes.

Chapter 3
Literature Review

3.1 Previous Work

During our research on mosquito larvae classification using deep learning and mobile technologies, we reviewed several significant studies that have contributed to this domain.

One early study **[1]** proposed the classification of *Aedes aegypti* and *Aedes albopictus* larvae using Deep Neural Networks (DNN) along with a smartphone-based imaging system enhanced with a 60x zoom microscope. The model utilized a Single Shot Detector (SSD) and achieved an accuracy of 94.19% in classification and 92.85% in region of interest (ROI) segmentation. The primary feature used was the comb-like silks of the larvae. However, the system could only detect Aedes species and lacked clear hardware documentation.

Another study **[2]** introduced a CNN-based model developed using Keras and TensorFlow for classifying mosquito vs. non-mosquito larvae. Implemented on both PCs and Raspberry Pi devices, the model reported 90.18% test accuracy and 86% real-world performance. While it demonstrated good general classification, it did not offer species-level predictions.

In a comparative study **[3]** involving multiple CNN architectures, including VGG16, VGG19, ResNet50, and InceptionV3, ResNet50 was found to be most suitable for real-world deployment, especially for non-Aedes classification. The dataset was gathered from online image repositories, and automatic feature extraction was utilized. However, the focus remained on Aedes species, with the issue of small dataset size and possible overfitting.

Another interesting direction was taken by researchers **[4]** who used Artificial Neural Networks (ANN) and Multiple Linear Regression (MLR) to predict mosquito population trends based on time-lagged meteorological data like humidity and temperature. Humidity was found to be the most influential factor. While this approach helped forecast larval abundance, it did not incorporate image-based classification.

An image-based study **[5]** employed ResNet50 to classify Aedes vs. Non-Aedes larvae using smartphone images, comparing raw images with those taken through a 60x microscope. The best performance (98.76%) was achieved with magnified images after enhanced preprocessing. This work highlighted the value of preprocessing but still did not extend to species-level classification beyond Aedes.

A combined approach **[6]** using CNN and Kalman Filters was proposed for classifying and tracking larvae in video sequences. The dataset included 50 short videos of larvae in diverse environmental conditions. While promising, the full paper was not accessible, and the system mainly focused on classification, not species-level identification.

Ensemble learning was used in another study **[7]**, combining models like VGG16, VGG19, ResNet50, and ResNet152. U-Net was employed for segmentation before classification,

eliminating the need for expert supervision or microscope input. Despite a limited dataset of 67 manually annotated images, the model achieved over 99% accuracy for Aedes classification.

A separate CNN-based study **[8]** used AlexNet to analyze the 8th segment of larvae (a comb-like feature). It showed 96.8% accuracy using a small dataset of 300 images with microscope-based imaging, raising concerns about its practical usability in real-world field settings.

In a more advanced evaluation **[9]**, researchers compared Vision Transformers (ViT), ConvNeXT, CvT-13, and CNN models for three-class classification: Aedes, Culex, and Neither. ConvNeXT showed the best performance (65.63%), while ViT underperformed due to dataset limitations and short training durations. The models were tested across datasets from Africa and the Americas, revealing generalization challenges.

Finally, a mobile-based application called iMOLAP **[10]** was developed using Inception V3 for on-field mosquito larvae detection and classification via Android smartphones. The app offered real-time classification, location tracking, and community-based reporting of breeding spots. It achieved 92.8% accuracy for Aedes larvae images. Despite its innovation, it was tested on a relatively small dataset of 420 images, potentially affecting its generalizability.

3.2 Limitations of Existing Research

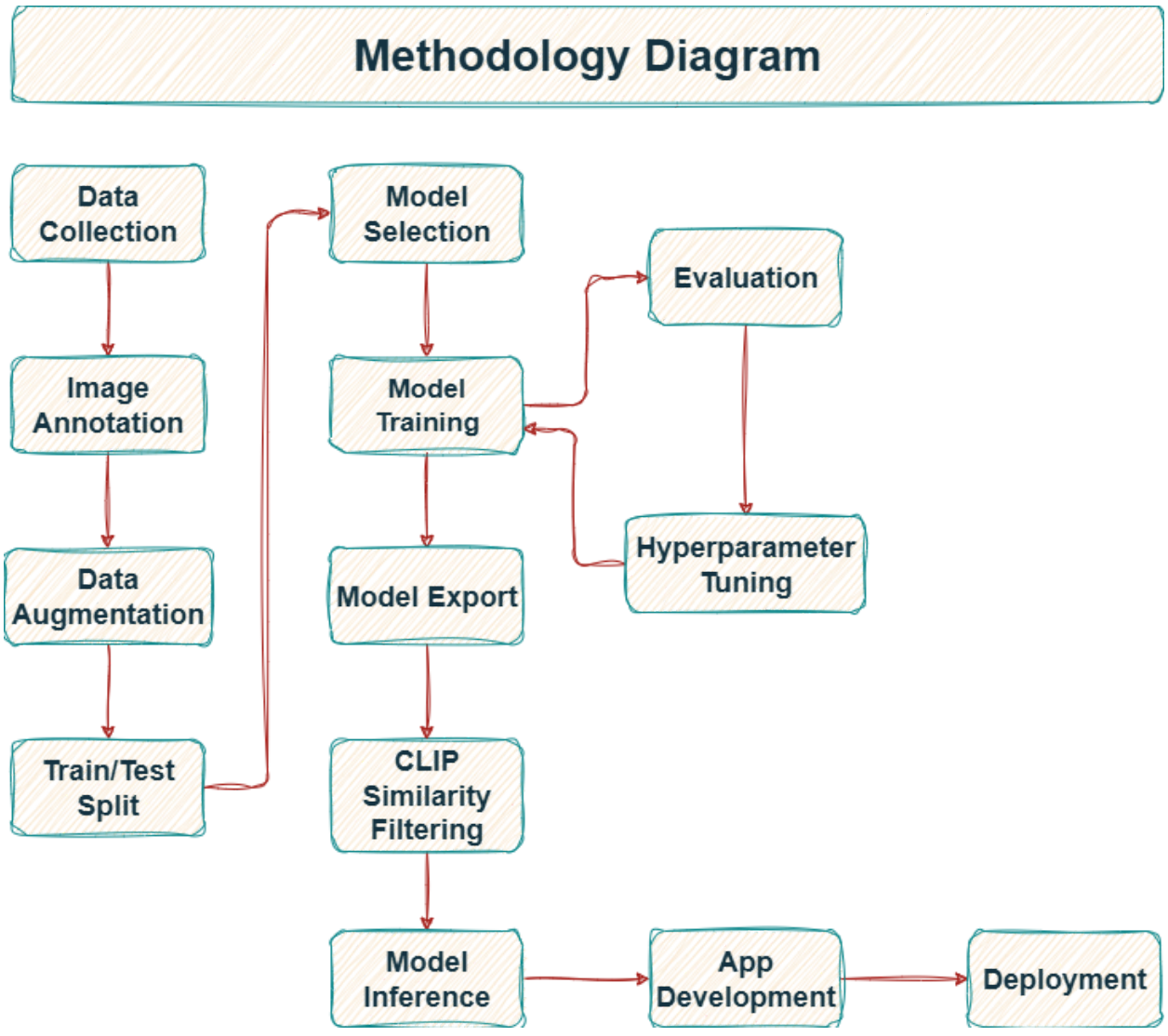
While these studies demonstrate promising developments in mosquito larvae classification and detection using deep learning, several limitations remain across existing works:

- Most models primarily focus on *Aedes* larvae classification and do not generalize to other species such as *Anopheles* or *Culex*, thereby reducing the scope of practical deployment [1], [3], [5], [6], [7].
- Multiple studies rely on microscope-enhanced or high-resolution imaging, which may not be feasible for large-scale or community-level use due to hardware limitations [1], [5], [8].
- Dataset sizes in many studies were relatively small (ranging from 67 to 420 images), increasing the risk of overfitting and limiting the model's ability to generalize across regions and environments [3], [7], [8], [10].
- Some systems (e.g., Kalman filter-based tracking or ANN with meteorological inputs) do not utilize image-based species classification, making them less relevant for larval identification in real-time [4], [6].
- Several works lack publicly accessible codebases or detailed descriptions of the model architecture and hardware integration, making replication and comparison difficult.
- Real-time mobile applications like iMOLAP are limited by smartphone processing capabilities and environmental factors like lighting and camera distance, which affect accuracy [10].
- Vision Transformer-based models, although cutting-edge, underperformed due to insufficient training epochs and high parameter counts, making them resource-heavy for deployment on low-power devices [9].

These limitations suggest a need for robust, scalable, and low-cost mosquito larvae classification systems capable of species-specific detection and mobile deployment. Our work aims to bridge these gaps by integrating deep learning models with open-set filtering using CLIP, multi-species classification, and a mobile-friendly, real-time deployment using Streamlit.

Chapter 4
Methodology

The proposed system is designed to build a lightweight, accurate, and mobile-accessible mosquito larvae classification system capable of distinguishing among **Aedes**, **Anopheles**, **Culex**, and optionally rejecting non-larval images using **OpenAI CLIP**. The complete methodology follows the steps illustrated in the pipeline diagram below.



4.1: Data Collection

4.1.1: Process of Data Collection

- We collected data in collaboration with **Sylhet City Corporation's dengue surveillance team**.
- Data is being gathered **systematically from all 35 wards**, covering multiple breeding sites.
- Captured **smart phone images (up to 10x zoom) of mosquito larvae in water bodies**.
- Additional images were sourced from online repositories for diversity.
- Images include **Aedes, Anopheles and Culex.**


4.1.2: At a Glance of Dataset

- **Our Own Data:**
  - **Raw Data = 1200+** (Aedes = 60%, Anopheles = 20 %, Culex = 20%)
  - **After Preprocessing = 1000+** (Aedes = 60%, Anopheles = 20 %, Culex = 20%)
- **Collected from Online Source:**
  - **Raw Data = 1000+** (Aedes = 40%, Anopheles = 30 %, Culex = 30%)
  - **After Preprocessing 800+ =** (Aedes = 40%, Anopheles = 30 %, Culex = 30%)
- **Total Data:**
  - **Raw Data = 2250+** (Aedes = 45%, Anopheles = 27.5%, Culex = 27.5%)
  - **After Preprocessing 1850+** (Aedes = 51.1%, Anopheles = 24.4%, Culex = 24.4%)

4.1.3: Data Pre-processing Steps

- Resizing to 640×640 pixels.
- For YOLO model
  - Labelling data with its type and bounding box information
- For Other's model
  - Separate data into their respective folder (Aedes, Anopheles, Culex)

4.1.4: Uploading Data

- YOLO model: Kaggle
- Rest of the model: Kaggle

4.2: Image Annotation

For the YOLOv8 detection model, images were annotated with bounding boxes using **Roboflow**, and annotations were saved in **YOLO format (class x_center y_center width height)**.

4.3: Data Augmentation

To prevent overfitting and enrich model generalization, data augmentation techniques were applied:

- Random horizontal and vertical flips
- Brightness and contrast adjustments
- RandAugment for MobileNetV3
- MixUp augmentation for EfficientNetB0

4.4: Train/Test Split

- YOLO model (train = 70%, val = 20%, test = 10%)
- Rest of the models (train = 70%, val = 15%, test = 15%)

For classification tasks, the dataset used **ImageFolder** format with class-labeled subfolders.

4.5: Model Selection

The system experimented with both **classification** and **object detection** models:

- **Classification Models:**
  - EfficientNetB0
  - MobileNetV3-Large
  - ResNet50
  - Vision Transformer (ViT)
  - MobileViT (for mobile optimization)
  - Ensemble (ResNet50 + MobileViT)
- **Detection Model:**
  - YOLOv8m (optimized for speed and accuracy)

4.6: Model Training

4.6.1: Classification

- **Framework:** PyTorch and timm
- **Loss Function:** CrossEntropyLoss
- **Optimizer:** Adam / AdamW
- **Scheduler:** CosineAnnealingLR
- **Metrics Tracked:** Accuracy, Precision, Recall, F1-Score
- **Advanced Techniques:**
  - MixUp (EfficientNetB0)
  - Stochastic Weight Averaging (SWA)
  - Early stopping
  - Class imbalance handling (WeightedRandomSampler)

4.6.2: Object Detection

- **Framework:** Ultralytics YOLOv8
- **Model:** YOLOv8m
- **Training:** 120 epochs, batch size 16
- **Metrics:** mAP@0.5, mAP@0.5:0.95, Precision, Recall

4.7: Evaluation

Each model was evaluated using the test dataset:

- **Metrics:**
  - Accuracy
  - Precision
  - Recall
  - F1-Score
  - Confusion Matrix
  - Inference Time (sec)
- **YOLOv8 additionally evaluated on:**
  - mAP@0.5
  - mAP@0.5:0.95
  - Precision / Recall per class

4.8: Hyperparameter Tuning

Multiple training runs were conducted by adjusting:

- Learning rate
- Batch size
- Early stopping patience
- Data augmentations
- Optimizers (**Adam vs AdamW**)

Best-performing hyperparameters were selected based on validation performance and inference time.

4.9: Model Export

- All models were exported in **.pt** format:
  - Classification models: **torch.save(model)**
  - YOLOv8: **model.save("best.pt")**
- Vision Transformer hosted on **Hugging Face**.
- Other models stored locally/**GitHub**.

4.10: CLIP Similarity Filtering (Open-Set Rejection)

To prevent irrelevant images (e.g., humans, animals) from being misclassified, we integrated **OpenAI CLIP ViT-B/32** for filtering:

- Image features are compared with larva-related textual prompts:
  - "a high-quality photo of Aedes mosquito larva in water"
  - "a high-quality photo of Anopheles mosquito larva"
  - "a high-quality photo of Culex mosquito larva close-up"
  - "a random object that is not a mosquito larva"

- Cosine similarity threshold of **0.28**
- If similarity < threshold → inference is skipped
- Ensures robustness and safety in real-world use

4.11: Model Inference

If the input passes CLIP filtering:

- Image is resized and normalized
- Model inference is performed
- Performed classification or detection if CLIP check passed.
- Highest probability class is selected using **softmax**
- For YOLOv8, bounding boxes and labels are returned

4.12: App Development

- Developed an interactive web app using Streamlit.
- Features:
    - Upload / Webcam support
    - Model selector
    - CLIP filtering before inference
    - Real-time predictions
    - Mobile-optimized UI
    - Downloadable prediction result

4.13: Deployment

- Uploaded complete project folder to GitHub.
- Deployed on Streamlit Cloud.
- Large models (e.g., ViT) loaded from Hugging Face.

Chapter 5
Implementation

5.1: Environment Development

The development and deployment of the mosquito larvae classification system involved a combination of powerful cloud computing, local development tools, and user-friendly deployment platforms.

5.1.1: Platforms and Tools

Model training and evaluation were conducted using Kaggle Notebooks, leveraging free access to dual **NVIDIA Tesla T4 GPUs**. This platform was chosen for its ease of use, GPU support, and integration with cloud storage, allowing for efficient large-scale model training and experimentation.

Local model inference, debugging, and Streamlit application development were carried out on Visual Studio Code (VS Code) with Python environment management using venv.

The complete Streamlit application was deployed publicly using Streamlit Cloud, making it accessible from both mobile and desktop browsers. To ensure efficient mobile compatibility and avoid the memory limitations of browser-based inference, large models such as Vision Transformer (ViT)—which is approximately 343 MB—were hosted on Hugging Face Hub and dynamically downloaded by the app at runtime.

5.1.2: Technologies and Libraries

The project utilized the following key frameworks and libraries:

Category	Tools / Libraries
Deep Learning Framework	PyTorch, timm (for EfficientNet, ViT)
Model Zoo / Training	timm, torchvision.models, ultralytics (YOLOv8)
Deployment	Streamlit (web interface), Hugging Face (model hosting)
Data Augmentation	torchvision.transforms, RandAugment
Inference Optimization	TorchScript/PT format, Streamlit cache, softmax
CLIP-based Filtering	OpenAI CLIP (ViT-B/32) for anomaly detection

5.1.3: Model Integration in Streamlit

A modular design was adopted for deploying multiple models in a user-selectable interface, including:

- EfficientNetB0
- MobileNetV3
- ResNet50
- Vision Transformer (ViT)

- YOLOv8m

The app allows the user to either upload an image or capture an image using a webcam. All inputs are first checked using OpenAI CLIP, which compares the uploaded image against textual prompts (e.g., "Aedes mosquito larva") to filter out irrelevant or non-larvae images. If the similarity threshold is met, the selected model is triggered to perform the classification task. Otherwise, the inference is aborted with a warning message—effectively acting as an open-set image filter.
To minimize redundancy, models were loaded on demand using Streamlit's @st.cache_resource decorator, significantly improving performance and responsiveness.

5.2: Model Pipeline

This section outlines the end-to-end model pipeline implemented for the mosquito larvae classification system. The pipeline was designed to be modular, scalable, and compatible with real-time applications, supporting six individual deep learning models and one ensemble model. The workflow spans from dataset preparation to inference and result display.

5.2.1: Dataset Preprocessing

To facilitate both object detection and classification tasks, the dataset was first annotated using Roboflow, where bounding boxes were manually drawn for each larva, and the images were resized to 640×640 pixels for compatibility with YOLOv8.
For classification models (EfficientNet, ResNet, ViT, etc.), the dataset was later restructured into three categories: Aedes, Anopheles, and Culex, following a directory-based split:
This structure was essential for use with torchvision.datasets.ImageFolder. Images were resized to 224×224 pixels and normalized using ImageNet statistics:

- Mean: [0.485, 0.456, 0.406]
- Std: [0.229, 0.224, 0.225]

To improve generalization, RandAugment was used during training along with WeightedRandomSampler to address class imbalance in the training data.

5.2.2: Classification Models

Six deep learning models were trained and evaluated:

Model	Description
EfficientNetB0	A lightweight yet powerful convolutional model from the EfficientNet family, known for its accuracy and mobile-friendliness.
MobileNetV3	An ultra-lightweight model optimized for mobile inference, using depthwise separable convolutions.
ResNet50	A deep CNN with residual connections, suitable for high-accuracy classification in moderately complex images.
Vision Transformer (ViT)	A transformer-based image classifier that divides the image into patches and applies self-attention for global feature learning.

YOLOv8m	A real-time object detection model used for identifying and localizing mosquito larvae in raw images.
MobileViT	A hybrid architecture combining CNN and transformer modules, optimized for mobile devices.

Each model was trained using PyTorch on Kaggle with dual Tesla T4 GPUs, and later exported in .pt format (both state_dict and full model). The best-performing models were integrated into a Streamlit app.

5.2.3: Ensemble Model (ResNet50 + MobileViT)

To further enhance accuracy and robustness, an ensemble model was built by averaging the logits from both ResNet50 and MobileViT before applying softmax. This approach combined the spatial sensitivity of CNNs with the global contextual awareness of transformers.
The ensemble significantly improved prediction confidence and class separation, particularly for Anopheles, which had previously shown lower recall.

5.2.4: Training Environment

- Platform: Kaggle Notebook
- GPU: T4 × 2

Individual Model pipeline :

5.2.5: EfficientNetB0 Model

Model Overview:
EfficientNetB0 is a lightweight convolutional neural network optimized for efficiency and accuracy. It was selected for its performance in resource-constrained environments like mobile deployment.
Training Environment:

- Epochs: 25 (Early stopping used)
- Batch size: 32
- Image size: 224 × 224

Data Augmentation & Preprocessing:

- Strong augmentation using **RandAugment**
- Resize to 224×224
- Normalization using ImageNet mean and std
- Class imbalance handled via **WeightedRandomSampler**

Loss Function:

- **SoftTargetCrossEntropy** (timm) with **MixUp** ($\alpha = 0.2$), label smoothing = 0.1

Optimizer & Scheduler:

- **AdamW** optimizer (LR = 5e-4, weight_decay = 1e-4)
- CosineAnnealingLR scheduler with **T_max = 20**

Regularization Enhancements:

- **MixUp augmentation:** Soft label interpolation for better generalization.

- **Stochastic Weight Averaging (SWA):** Activated from epoch 15 to stabilize the model toward the end of training.

Early Stopping:

- Activated when validation loss failed to improve for 4 consecutive epochs.

Evaluation Metrics:

- **Loss curve**, **validation accuracy**, **confusion matrix**, and **ROC curve** were plotted

Model Saving:

- Best model checkpoint (**best_efficientnetb0.pth**) saved.

5.2.6: MobileNetV3 Model

Model Overview:

MobileNetV3 is a highly efficient convolutional neural network optimized for edge devices. It achieves competitive accuracy with significantly fewer parameters and lower latency, making it an ideal choice for mobile deployment.

Training Environment:

- Epochs: 25 (with early stopping)
- Batch size: 32
- Image size: 256×256 → Center Cropped to 224×224

Data Preprocessing and Augmentation:

- **Resize** to 256×256 and **CenterCrop** to 224×224
- **RandAugment** with 2 operations and magnitude 9
- **Normalization** with ImageNet mean and std
- **WeightedRandomSampler** used to handle class imbalance

Loss Function and Optimization:

- **MixUp** ($\alpha = 0.1$) with **label smoothing** (0.1)
- Loss: **SoftTargetCrossEntropy** from **timm**
- Optimizer: **AdamW** with scaled learning rate **1.25e-4**
- Learning Rate Scheduler: **CosineAnnealingLR** with **T_max=20**

Advanced Training Enhancements:

- **Automatic Mixed Precision (AMP):** Enabled using **torch.cuda.amp** for faster training and reduced memory usage.
- **Early Stopping:** Triggered after 4 epochs without validation loss improvement
- **Training History:** Tracked and visualized (loss curves, accuracy curves)

Model Export:

- Best model saved as **best_mobilenetv3.pth**
- Final model exported with full weights:
  torch.save(model, "/kaggle/working/mobilenetv3_large_final.pt")

5.2.7: ResNet50 Model

Model Overview:

ResNet50 is a deep convolutional neural network based on residual learning. It consists of 50 layers and is known for its strong feature extraction capability. The architecture is particularly well-suited for complex image classification tasks with moderate-sized datasets.

Training Environment:

- Epochs: 15
- Batch size: As defined by external loader

- Input size: 224×224 (standard for ResNet)
- Optimizer: **Adam** with learning rate **1e-4**
- Loss function: **CrossEntropyLoss**

Transfer Learning:

- The base ResNet50 model was loaded with **ImageNet pretrained weights**

Evaluation Metrics:

- **Loss curve**, **validation accuracy**, **confusion matrix**, and **ROC curve** were plotted

Model Export:

- State dict saved using:
  torch.save(model.state_dict(), "resnet50_larvae_model.pth")
- Full model exported as:
  torch.save(model, "/kaggle/working/resnet50_final.pt"

5.2.8: MobileViT Model

Model Overview:

MobileViT is a hybrid architecture that combines convolutional layers for local feature extraction with transformer blocks for capturing long-range dependencies. It is lightweight, making it suitable for mobile applications while maintaining high accuracy.

Training Environment:

- Epochs: 25
- Batch size: 32
- Input size: 224×224
- Framework: PyTorch + **timm**
- Dataset: 3 classes (Aedes, Anopheles, Culex)

Preprocessing:

- Resized all images to **224×224**
- Normalized with mean = std = [0.5, 0.5, 0.5]
- Applied basic **ToTensor()** transform

Model Architecture & Training:

- Pretrained model loaded via:
  model = timm.create_model("mobilevit_s", pretrained=True, num_classes=3)
- Loss function: **CrossEntropyLoss**
- Optimizer: **AdamW** with learning rate **1e-4**
- Scheduler: CosineAnnealingLR (T_max = 10)

Training Highlights:

- Trained for 25 epochs with:
  - Training and validation loss monitoring
  - Validation accuracy tracking
  - Early stopping based on best accuracy

Evaluation Metrics:

- **Loss curve**, **validation accuracy**, **confusion matrix**, and **ROC curve** were plotted

Model Export:

Final trained model (state_dict) saved for deployment:
torch.save(model.state_dict(), "/kaggle/working/best_mobilevit.pth")

5.2.9: Vision Transformer (ViT) Model

Model Overview:

The Vision Transformer (ViT) applies transformer architecture to image data by splitting images into patches and processing them like token sequences. It captures global attention and long-range dependencies more effectively than traditional CNNs. Due to its transformer-based architecture, ViT often requires robust regularization and data augmentation for optimal performance.

Training Environment:

- Epochs: 25
- Batch size: 32
- Input resolution: 224×224
- Framework: PyTorch + timm

Preprocessing:

- Resized input images to **256×256** and center cropped to **224×224**
- Applied **RandAugment** for data augmentation

Model Architecture & Training:

- Pretrained model loaded and customized:
  vit = timm.create_model('vit_base_patch16_224', pretrained=True, num_classes=3)
  vit.head = nn.Sequential(
  nn.Dropout(0.3),
  nn.Linear(vit.head.in_features, 3))
- Loss: **SoftTargetCrossEntropy**
- Optimizer: AdamW with scaled LR **1e-3 * (batch_size / 256)**
- Scheduler: **Cosine Annealing** (T_max=20)
- Augmentation: MixUp with mixup_alpha=0.1, cutmix_alpha=0.0, and label smoothing 0.1

Training Strategy:

- Used **WeightedRandomSampler** to handle class imbalance
- Used **AMP (Automatic Mixed Precision)** with torch.amp.autocast and **GradScaler** for faster training
- Tracked training loss, validation loss, and accuracy each epoch
- Early stopping based on validation loss (patience = 4)

Best model saved using:

torch.save(vit.state_dict(), "best_vit.pth")

Evaluation Metrics:

- Accuracy, Precision, Recall, and F1-score (macro average)
- Multi-class ROC-AUC computed and plotted
- Confusion matrix plotted with **seaborn.heatmap**
- Visualization of 5 random predictions using test images

Model Export:

- Final model saved in full format:
  torch.save(vit, "/kaggle/working/vit_final.pt")

5.2.10: YOLOv8m Model

Model Overview:

YOLOv8 is a state-of-the-art object detection model developed by Ultralytics. It performs both localization (bounding box detection) and classification simultaneously. The **YOLOv8m (medium version)** balances detection accuracy and inference speed, making it ideal for mobile-based mosquito larvae identification.

Training Environment:

- Epochs: 120 (early stopping: 20 patience)
- Batch size: 16
- Input size: 640×640
- Framework: Ultralytics + PyTorch backend
- Dataset Format: YOLO format (with **.txt** annotations for bounding boxes)

Dataset Preparation:

- YAML configuration file defined:
  train: /data/train/images
  val: /data/valid/images
  test: /data/test/images
  nc: 3
  names: ['Aedes', 'Anopheles', 'Culex']
- Label files in YOLO format:
  Each .txt file contains:
  class_id x_center y_center width height

Model Initialization & Training:

- Pretrained model used:
  model = YOLO("yolov8m.pt")
- Trained using:

```
model.train(
   data="data.yaml",
   epochs=120,
   imgsz=640,
   batch=16,
   patience=20,
   device=0
)
```

- Trained weights saved as:
  /kaggle/working/runs/detect/train/weights/best.pt

Evaluation Metrics:

Evaluation performed using:

metrics = model.val()

- Reported metrics:
  - **Precision** (mean): metrics.box.p.mean()
  - **Recall** (mean): metrics.box.r.mean()
  - **mAP@0.5**: metrics.box.map.mean()
  - **mAP@0.5:0.95**: metrics.box.map50.mean()

Model Export:

- Best trained model stored at:
  /kaggle/working/runs/detect/train/weights/best.pt

5.2.11: Ensemble Model (ResNet50 + MobileViT)

Model Overview:

To improve classification robustness, an ensemble model was created by combining predictions from two diverse architectures—ResNet50 (a deep CNN) and MobileViT (a lightweight Vision Transformer). This approach uses soft voting (averaging class probabilities) to enhance generalization and reduce variance across individual model predictions.

Models Used:

- **ResNet50**: A deep residual network pretrained on ImageNet.
- **MobileViT**: A mobile-friendly transformer-CNN hybrid from **timm**.

Training (Individual Models):

- ResNet50 and MobileViT were trained separately on the same mosquito larvae dataset using:
  - Dataset: 224×224 normalized images
  - Frameworks: **torchvision.models** and **timm.create_model**
  - Loss Function: CrossEntropyLoss
  - Optimizer: Adam / AdamW
  - Validation during training was used to save the best weights for each model.

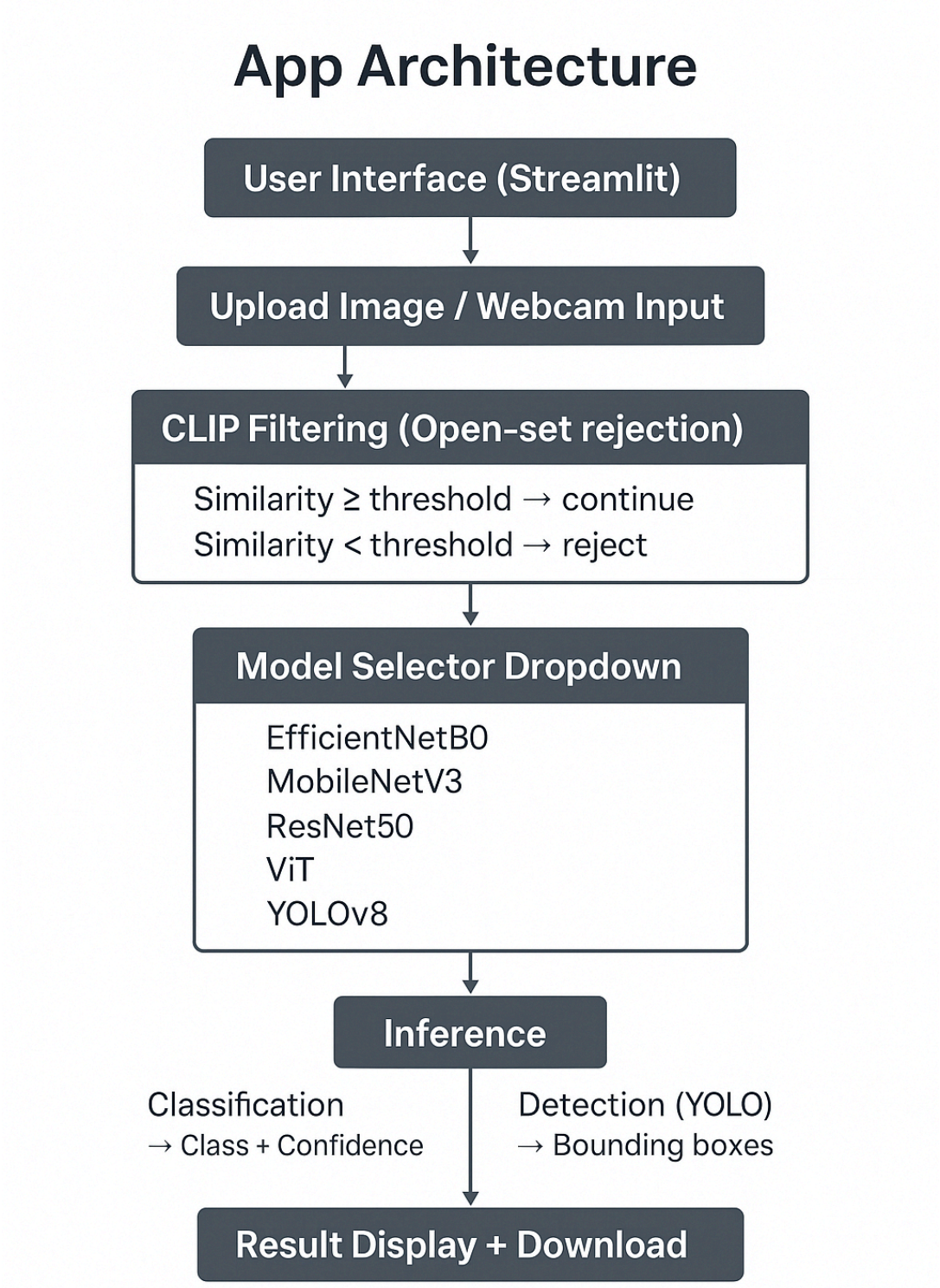
Test-Time Ensemble Strategy:

- Softmax outputs from each model were averaged:
  res_out = torch.softmax(resnet(images), dim=1)
  mob_out = torch.softmax(mobilevit(images), dim=1)
  avg_probs = (res_out + mob_out) / 2
- Final prediction taken from highest combined probability:
  _, predicted = torch.max(avg_probs, 1)

5.3: App Architecture & Streamlit Integration

The developed mosquito larvae classification system is deployed as a user-friendly web application using Streamlit. The app was designed with mobile responsiveness, ease of use, and real-time interaction in mind. It supports both image upload and live webcam input, allowing flexible usage in field or lab settings.

Diagram:


App Architecture
User Interface (Streamlit)
Upload Image / Webcam Input
CLIP Filtering (Open-set rejection)
Similarity ≥ threshold → continue
Similarity < threshold → reject
Model Selector Dropdown
EfficientNetB0
MobileNetV3
ResNet50
ViT
YOLOv8
Inference
Classification
→ Class + Confidence
Detection (YOLO)
→ Bounding boxes
Result Display + Download

Key Functionalities:

- **Input Modes:**
  - Upload from device (JPEG, PNG)
  - Capture using webcam or mobile camera
- **Model Selection:**
  - EfficientNetB0, MobileNetV3, ResNet50, Vision Transformer (ViT), YOLOv8
- **Output:**
  - Predicted class label (Aedes, Anopheles, Culex)
  - Confidence score
  - Inference time
  - Visual explanation (GradCAM can be optionally added)
  - Downloadable prediction summary

Backend Flow:

- The selected model is loaded on demand.
- Input image undergoes CLIP-based filtering.
- If passed, the image is preprocessed and fed into the selected model.
- The result is displayed and can be downloaded.

Technologies:

- **Frontend:** Streamlit
- **Backend:** PyTorch, timm, torchvision
- **Filtering:** OpenAI CLIP (ViT-B/32)
- **Deployment-ready for**: Streamlit Cloud, Hugging Face Spaces

5.4: CLIP-Based Filtering Module

To enhance reliability and handle unexpected or irrelevant inputs (e.g., photos of objects, people, other living things), we integrated **OpenAI's CLIP model** for pre-inference filtering.

```
38  # Class labels
39  class_names = ['Aedes', 'Anopheles', 'Culex']
40  clip_class_names = [
41      "a high-quality photo of Aedes mosquito larva in water",
42      "a high-quality photo of Anopheles mosquito larva",
43      "a high-quality photo of Culex mosquito larva close-up",
44      "a random object that is not a mosquito larva"
45  ]
46
47  # Load CLIP
48  @st.cache_resource
49  def load_clip():
50      model, preprocess = clip.load("ViT-B/32", device="cpu")
51      return model, preprocess
52
53  clip_model, clip_preprocess = load_clip()
54
```

```
        # --- CLIP Similarity Filtering ---
        st.write("📊 Running CLIP similarity check...")
        threshold = 0.28
        is_larva, sim_score = is_larva_image(image, threshold=threshold)


        if not is_larva:
            st.error("🛑 Image does not resemble a mosquito larva. Prediction aborted.")
            st.stop()
        else:
            st.success("✅ CLIP similarity check passed — looks like a Mosquito larva.")

```

CLIP Code Snippet

Filtering Logic:

- **Similarity Threshold:** 0.28 (empirically chosen)
- If similarity ≥ threshold, proceed to model inference.
- Else, skip inference and show "🛑 Image does not resemble a mosquito larva. Prediction aborted." message.

Advantages:

- Prevents false predictions from unrelated images
- Enables **open-set rejection**, making the system robust in real-world usage

5.5: Model Export & Deployment

To enable practical use and portability, trained models were exported and deployed in an optimized format.

Model Formats:

- Classification models (EfficientNetB0, ViT, etc.) exported as:
  - .pt full models for easy torch.load(...) usage in apps
- YOLOv8 detection model saved using **Ultralytics framework's .pt** format

Deployment:

- App runs locally and online on **Streamlit Cloud** with minimal setup
- Fully supports mobile devices for field testing
- Model loading is cached for performance

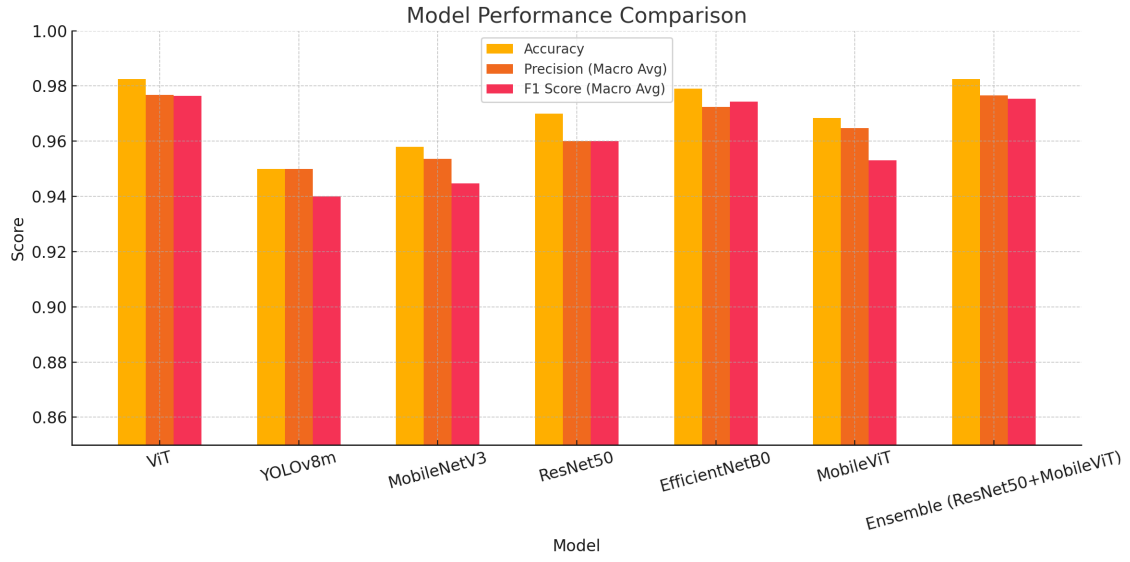
Mobile Optimization:

- Lightweight models (MobileNetV3, MobileViT, EfficientNetB0) prioritized
- UI elements are mobile-friendly (e.g., buttons, toggles)
- Webcam input works seamlessly on mobile browsers

Chapter 6
Result Discussion And Performance Analysis

6.1: Mosquito Larvae Classification Model Comparison

Model	Model Size (MB)	Accuracy	Precision (Macro Avg)	Recall (Macro Avg)	F1 Score (Macro Avg)
ViT	343.26	0.9825	0.9768	0.9769	0.9765
YOLOv8m	52.14	0.9500	0.9500	0.9400	0.9400
MobileNetV3	17.08	0.9580	0.9537	0.9386	0.9447
ResNet50	94.4	0.9700	0.9600	0.9600	0.9600
EfficientNetB0	16.4	0.9790	0.9724	0.9763	0.9743
MobileViT	19.93	0.9685	0.9647	0.9476	0.9531
Ensemble (ResNet50+MobileViT)	113.9300	0.9825	0.9766	0.9746	0.9754



6.2: Class-wise Metrics

Model	Class	Precision	Recall	F1-Score	Support
ViT	Aedes	1.0000	0.9873	0.9936	157
	Anopheles	0.9804	0.9434	0.9615	53
	Culex	0.9500	1.0000	0.9744	76
YOLOv8m	Aedes	0.9600	0.9700	0.9700	135
	Anopheles	0.9700	0.8600	0.9100	42
	Culex	0.9100	0.9800	0.9500	54
MobileNetV3	Aedes	0.9810	0.9873	0.9841	157
	Anopheles	0.9787	0.8679	0.9200	53
	Culex	0.9012	0.9605	0.9299	76
ResNet50	Aedes	0.9900	0.9900	0.9900	157
	Anopheles	0.9300	0.9400	0.9300	53
	Culex	0.9500	0.9500	0.9500	76
EfficientNetB0	Aedes	0.9936	0.9873	0.9904	157
	Anopheles	0.9630	0.9811	0.9720	53
	Culex	0.9605	0.9605	0.9605	76
MobileViT	Aedes	1.0000	0.9900	1.0000	157
	Anopheles	1.0000	0.8500	0.9200	53
	Culex	0.8900	1.0000	0.9400	76
Ensemble(ResNet50+MobileViT)	Aedes	1.0000	0.9936	0.9968	157
	Anopheles	0.9804	0.9434	0.9615	53
	Culex	0.9494	0.9868	0.9677	76

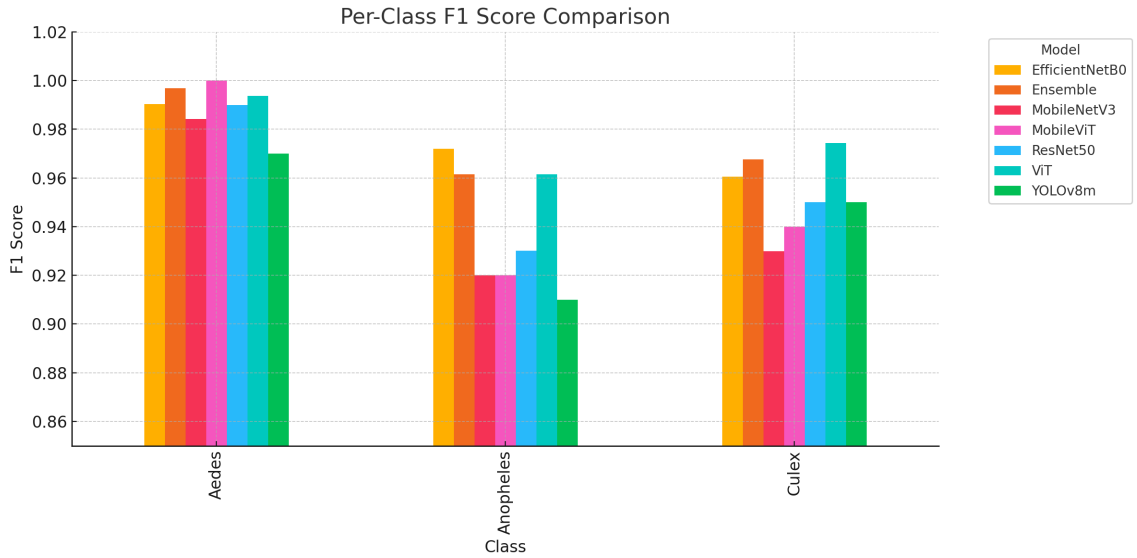
6.3: Result Discussion

This section discusses the performance of multiple deep learning models for mosquito larvae classification, including Vision Transformer (ViT), YOLOv8m, MobileNetV3, ResNet50, EfficientNetB0, MobileViT, and an ensemble approach combining ResNet50 and MobileViT.

Among all models, the Vision Transformer and the Ensemble model demonstrated the highest overall accuracy (98.25%), macro F1-score (97.54%+), and strong per-class performance, particularly for the Culex and Anopheles classes. MobileNetV3 and MobileViT also provided competitive results while being highly lightweight (under 20MB), making them suitable for mobile deployment.

The YOLOv8m model, although slightly lower in F1-score for Anopheles, provides the benefit of object detection, which can be explored further in future work. The class-wise analysis shows that all models performed exceptionally well for Aedes classification, with most achieving over 98% precision and recall. However, Anopheles larvae were more challenging, often resulting in lower recall, possibly due to intra-class similarity or data imbalance.

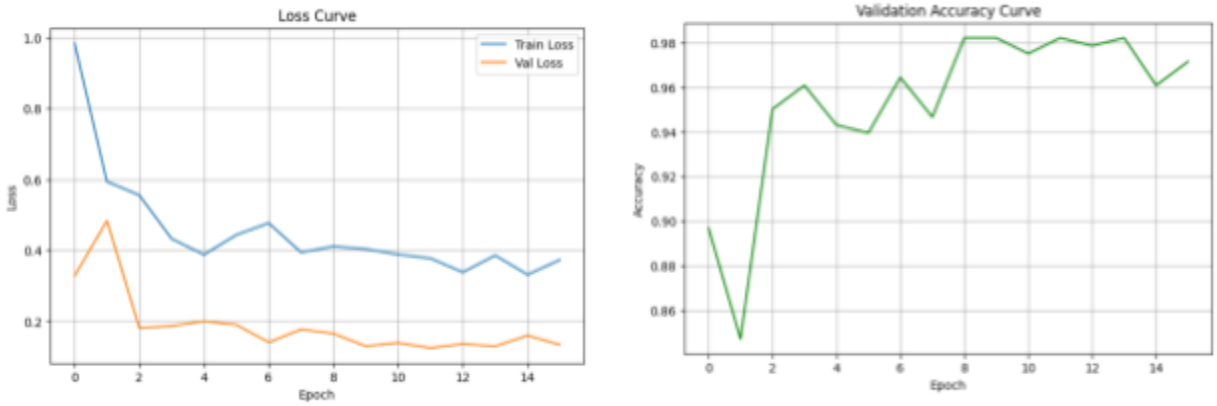
Overall, the ensemble approach offered a balanced trade-off between accuracy and robustness, benefiting from complementary strengths of ResNet50 and MobileViT. For practical deployments, MobileViT and EfficientNetB0 stand out due to their small size and strong accuracy, making them ideal for real-time mobile applications.



Per-class F1 score comparison across models

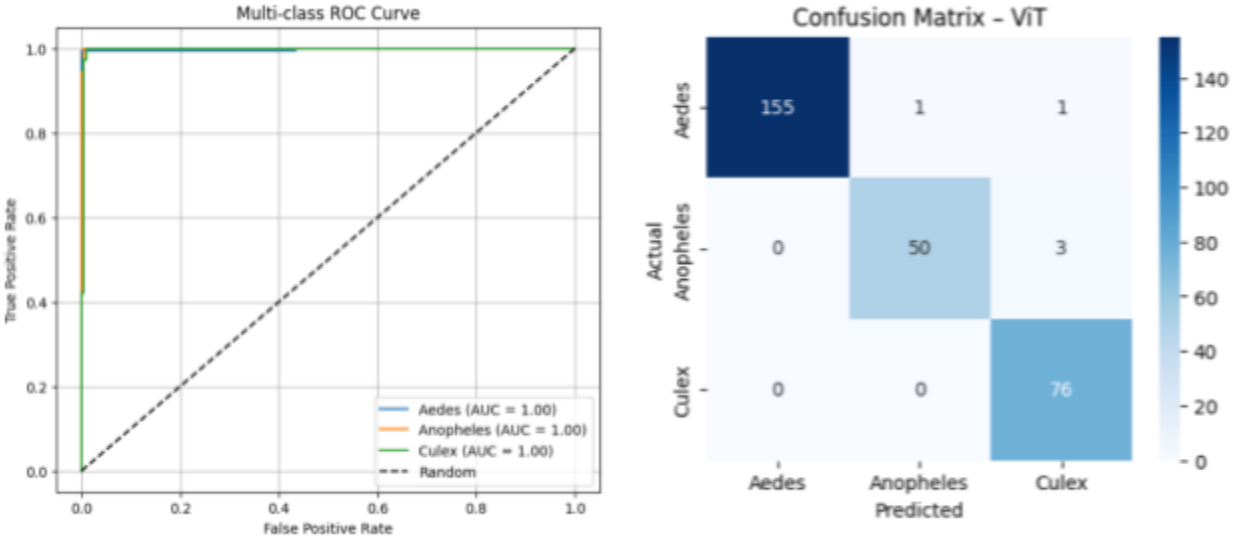
6.4: Model Wise Training Curves and Confusion Matrix

6.4.1: Vision Transformer (ViT)



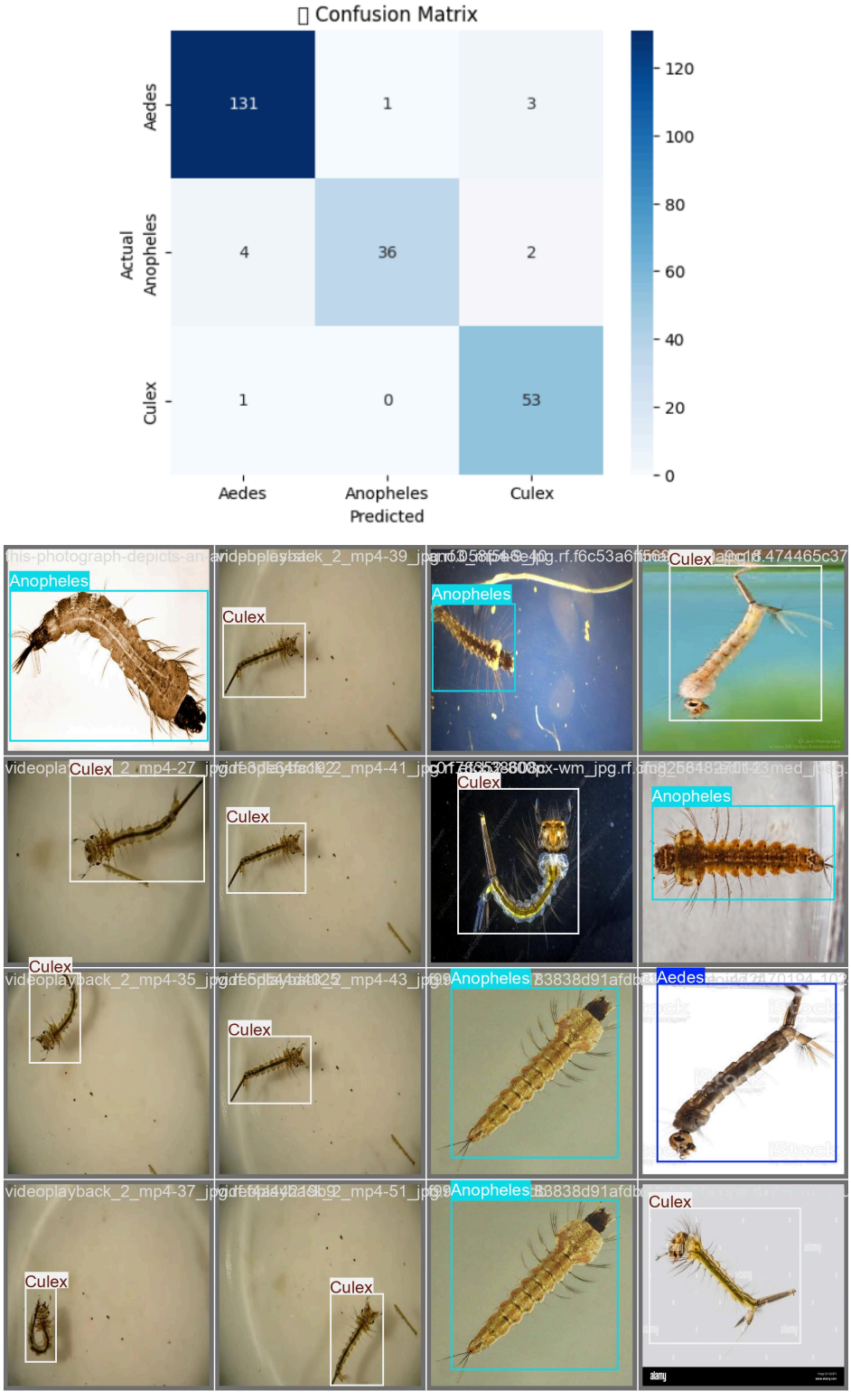
ViT Training Curves

- The loss curve shows smooth convergence with no overfitting. The use of regularization (like label smoothing, augmentation, or dropout) is likely effective.
- The model achieves high accuracy early and maintains it with minimal variance, suggesting that it has learned robust and generalizable features.



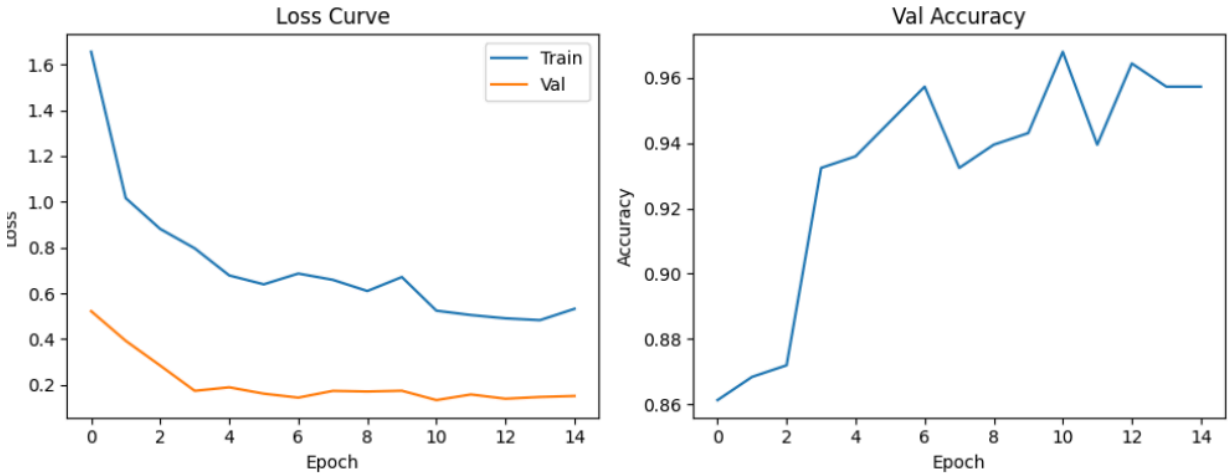
ViT ROC Curve and Confusion Matrix

6.4.2: YOLOV8m

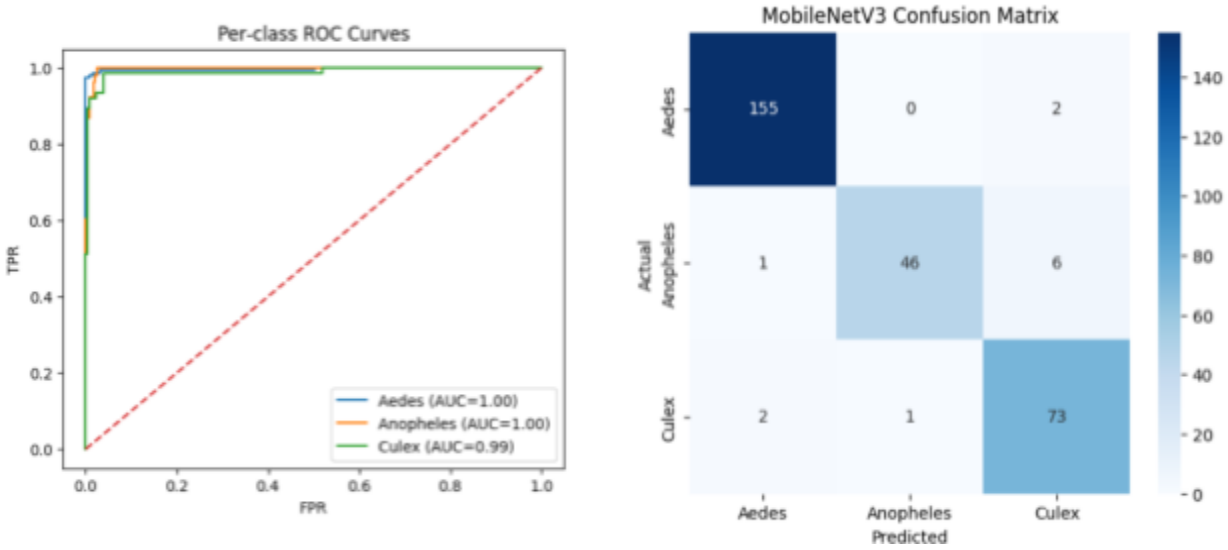


val_batch0_labels.jpg

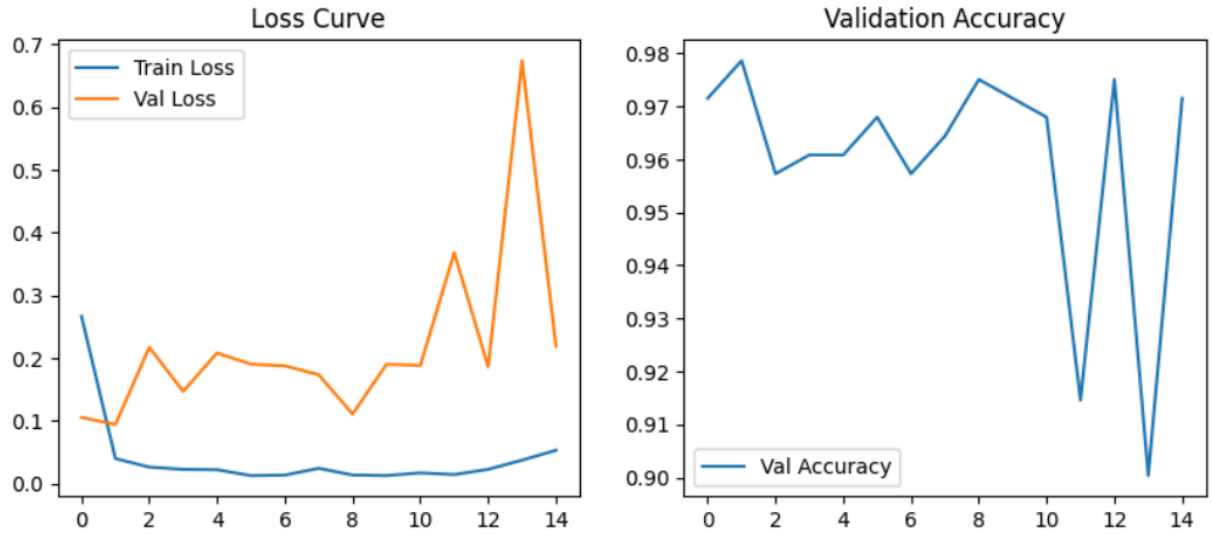
6.4.3: MobileNetV3



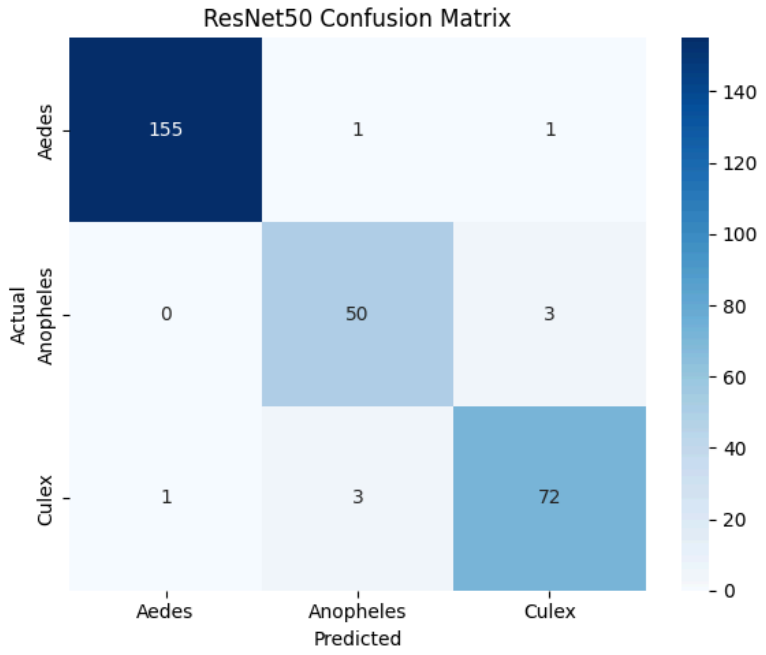
- The loss curves indicate **efficient convergence and minimal overfitting**. The gap between training and validation loss remains reasonable, which is a sign of good generalization.
- This is a **well-trained and stable model**, ideal for mobile deployment scenarios. The accuracy progression shows strong learning ability and sustained performance after convergence.
- These results validate MobileNetV3 as a highly efficient and lightweight architecture for mosquito larvae classification, balancing performance and deployment efficiency.



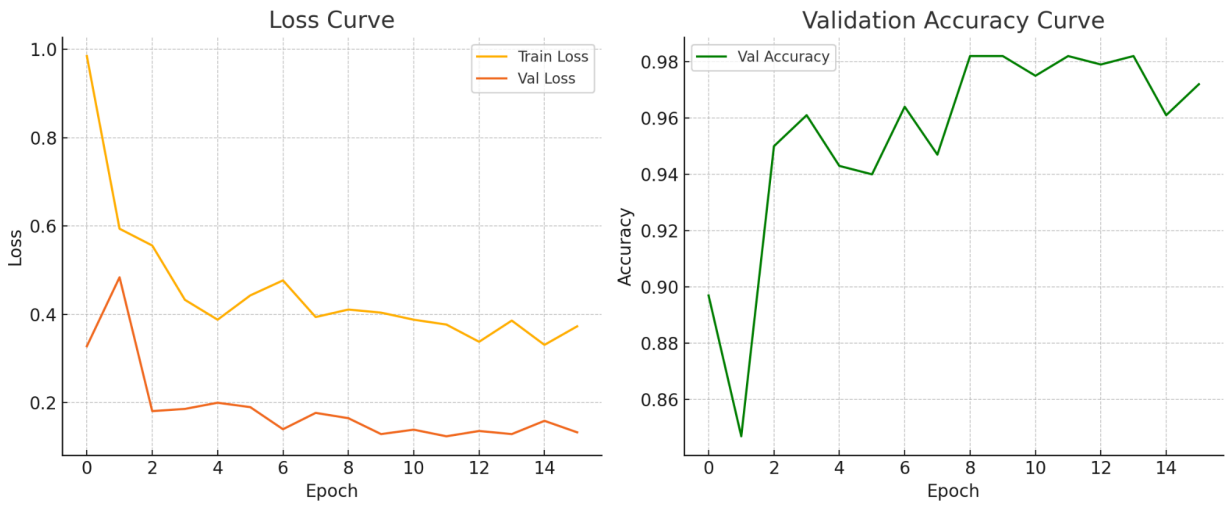
6.4.4: ResNet50



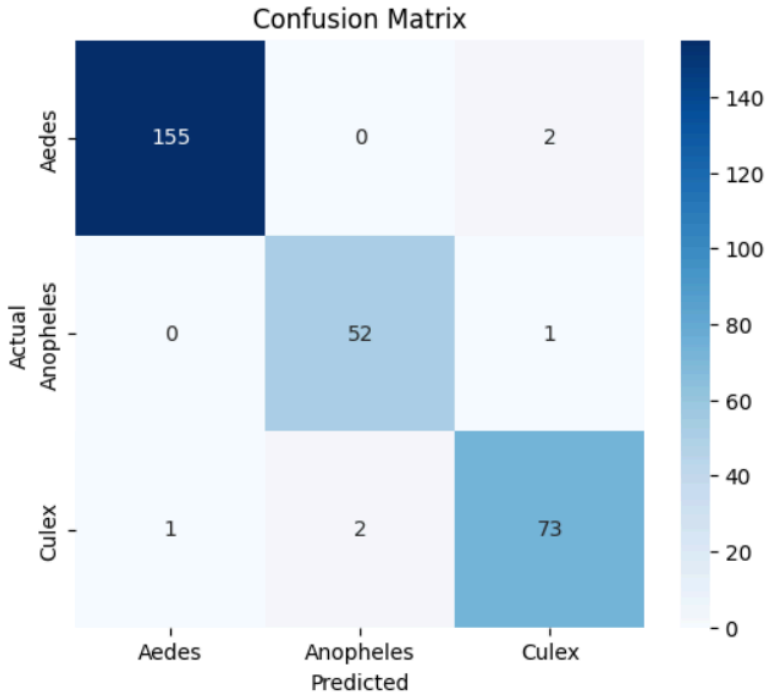
- The ResNet50 model initially shows strong performance, achieving high accuracy and low loss within the first 10 epochs. However, as training continues, validation loss increases sharply, and accuracy fluctuates, indicating overfitting. These results highlight the need for early stopping and improved regularization when using deeper convolutional networks like ResNet50 on limited datasets.



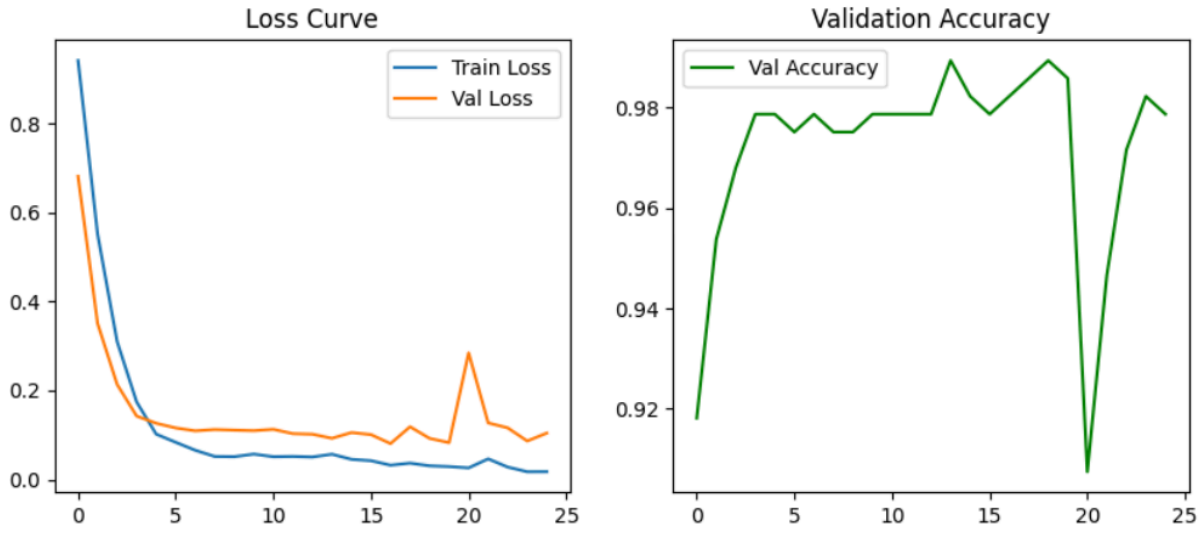
6.4.5: EfficientNetB0



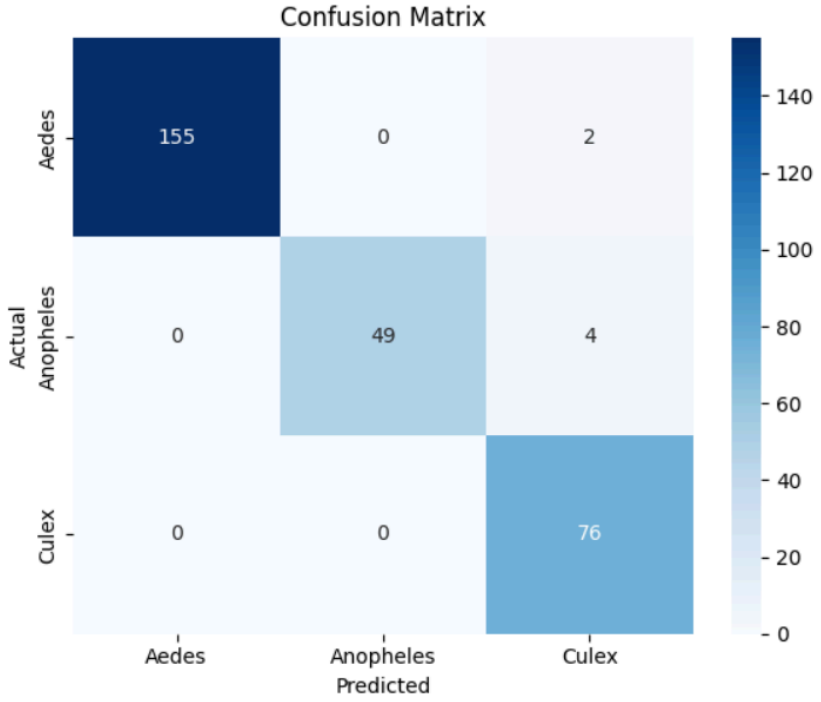
- EfficientNetB0 shows **stable convergence** with a well-regularized training process. Slight training loss jitter is tolerable, as validation loss remains unaffected.
- The model achieves **high and stable validation accuracy**, indicating strong generalization and reliability—ideal for real-world or mobile applications.
- These results highlight EfficientNetB0's ability to balance lightweight architecture with high accuracy, making it highly suitable for real-time mosquito larvae classification tasks on resource-constrained devices.



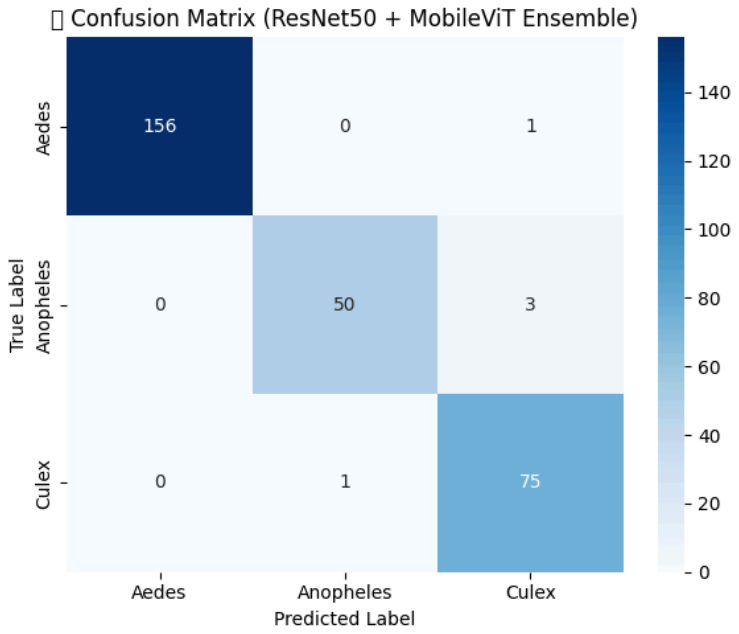
6.4.6: MobileViT



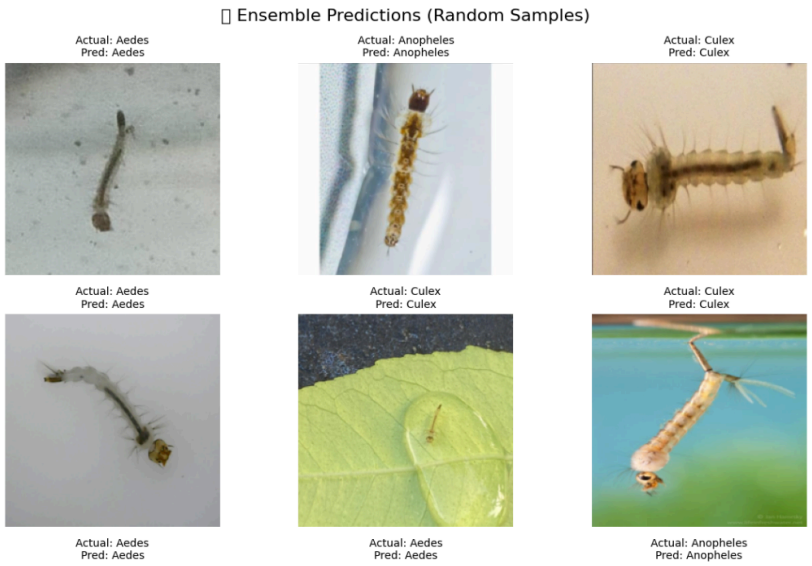
- This is a **well-behaved training curve**. The model learns effectively, generalizes well, and shows **no major signs of overfitting**. The spike near epoch 20 could be handled by early stopping, which seems to be applied.
- The model demonstrates **strong generalization** with minor instability. Likely influenced by data augmentation or optimizer momentum shifts, but overall robustness is excellent.



6.4.7: Ensemble (ResNet50+MobileViT)

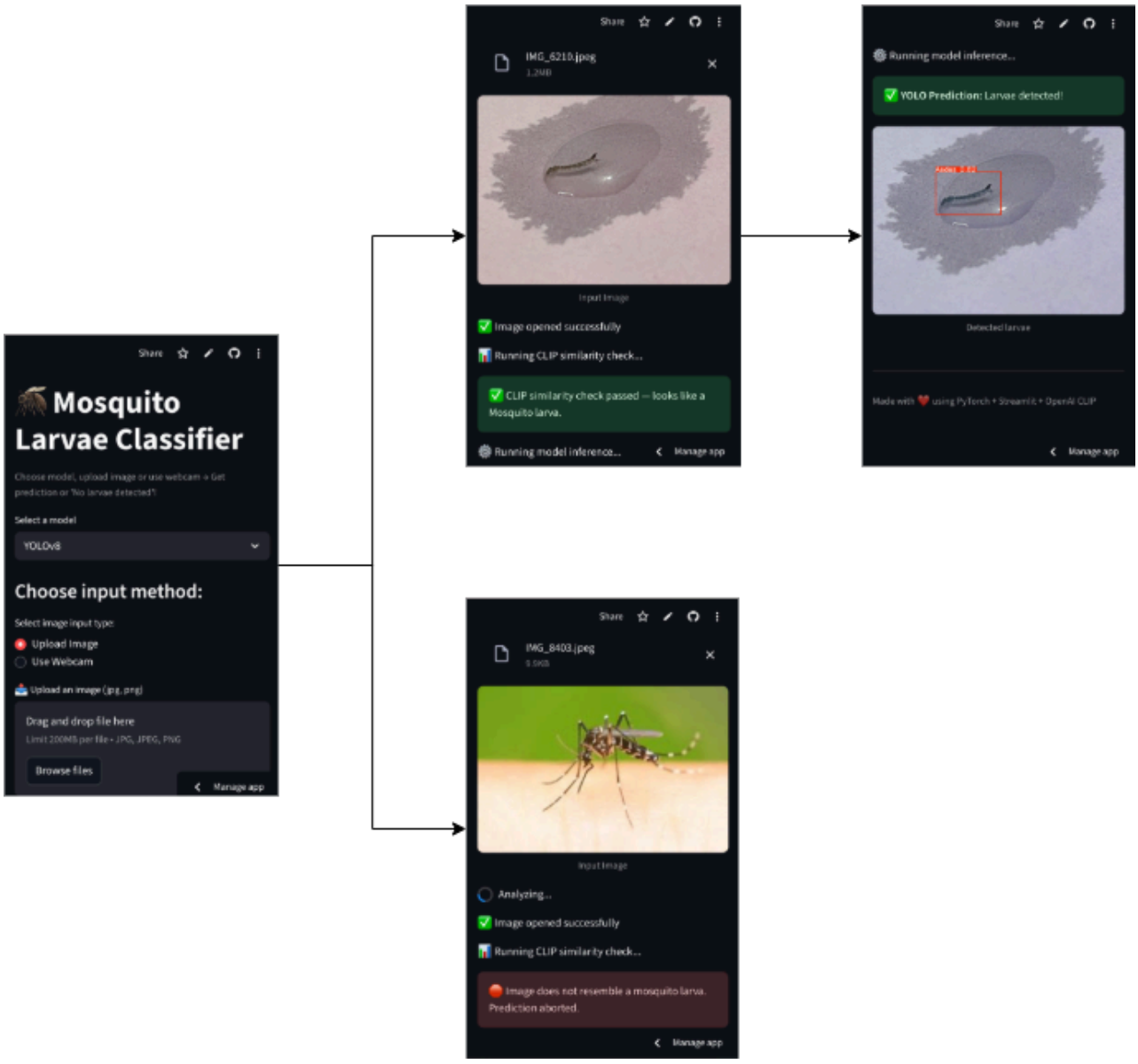


- The confusion matrix reveals that the ensemble model of ResNet50 and MobileViT achieves high classification accuracy across all classes. Aedes larvae are almost perfectly identified, while Anopheles and Culex have minimal confusion, likely due to visual similarities in morphology. Notably, no samples were misclassified as Aedes from other classes, indicating strong model confidence in distinguishing Aedes larvae. This supports the ensemble model as a highly reliable classifier, especially valuable in real-world scenarios where misclassification may affect mosquito control measures.



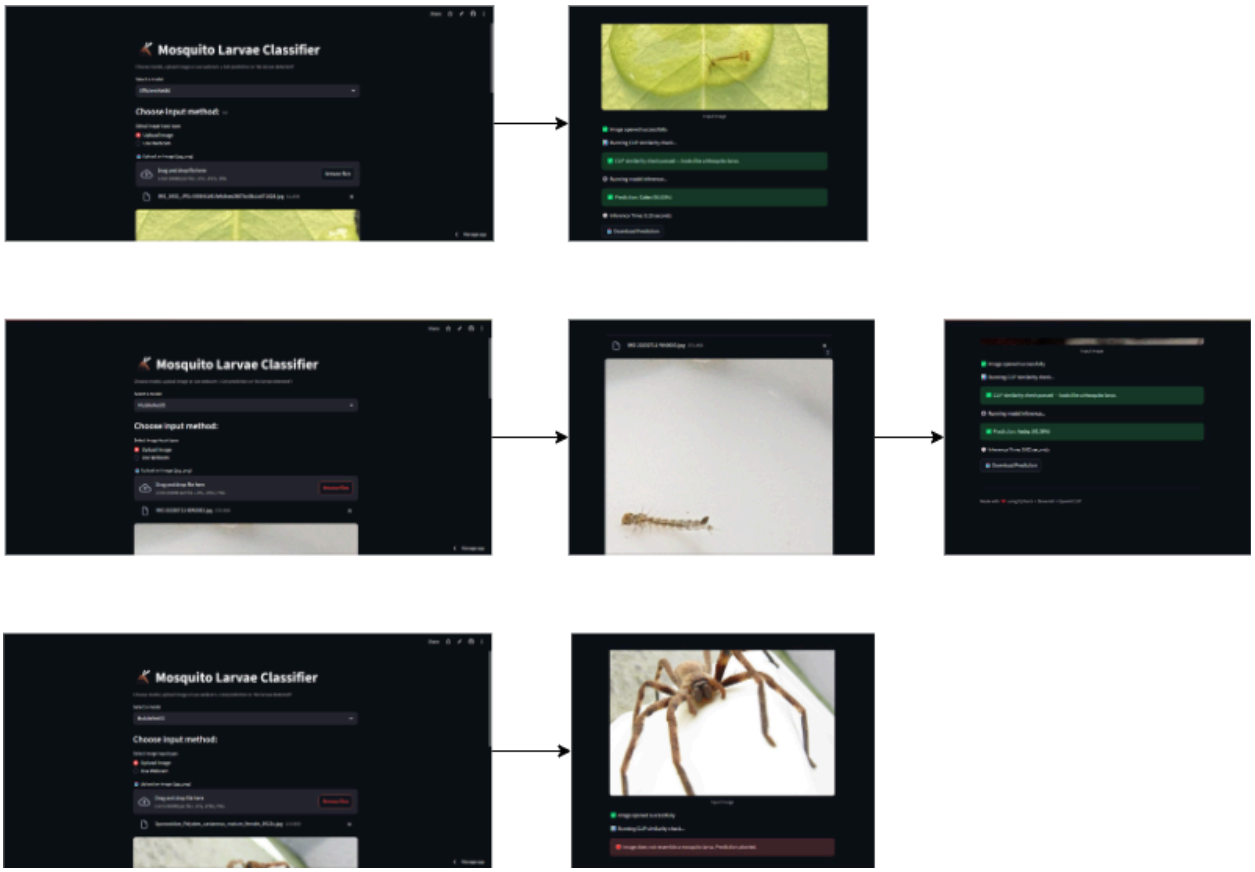
6.5: Webb Application

6.5.1: In Mobile (web app)



6.5.2: Use of mobile camera

6.5.3: In Computer



Chapter 7
Conclusion

This thesis presents a lightweight, accurate, and mobile-accessible mosquito larvae classification system using deep learning and modern open-set filtering techniques. The system is capable of identifying three major mosquito species—**Aedes**, **Anopheles**, and **Culex**—based on larval images captured in real-world conditions. A diverse set of models were trained and evaluated, including EfficientNetB0, MobileNetV3, ResNet50, Vision Transformer (ViT), and MobileViT. Additionally, a ResNet50 + MobileViT **ensemble** was introduced for improved performance.

To address the issue of irrelevant or out-of-distribution images, **OpenAI CLIP** was integrated as a **pre-filtering module**, significantly enhancing the robustness of the classification pipeline. YOLOv8 was also incorporated for object detection, offering bounding box localization of larvae. The final system was deployed as a responsive **web and mobile app** using **Streamlit**, supporting webcam input, dark mode, model selection, and downloadable results—making it suitable for use in field environments and low-resource settings.

Evaluation metrics such as **accuracy, precision, recall, F1-score**, and **inference time** confirmed the system's reliability and real-time capabilities. The combination of lightweight models, open-set filtering, and mobile deployment makes this solution practical for early dengue prevention and vector surveillance.

Chapter 8
Future Work

While the current system performs reliably, there are several areas for potential enhancement in future research:

- **Expand the Dataset**: Collecting more high-quality images under varying lighting and environmental conditions—including additional mosquito species and larval stages—would improve model generalization.

- **Multi-Larva Detection and Classification**: Extend the system to handle **multiple mosquito larvae in a single image**, accurately detecting and classifying each larva separately using advanced object detection and instance segmentation models (e.g., **YOLOv8**, **Mask R-CNN**, or **SegFormer**).

- **Anomaly Detection**: Integrating unsupervised or semi-supervised anomaly detection to complement or replace CLIP filtering for better rejection of unknown inputs.

- **Model Optimization**: Converting models to **ONNX**, **TorchScript**, or **TFLite** for faster inference and offline mobile deployment.

- **Full Android/iOS App**: Rebuilding the system as a native mobile app using frameworks like Flutter or React Native, possibly integrated with GPS to map larval hotspots.

- **Offline Mode**: Enable offline functionality for use in remote areas without internet access by bundling lightweight models with the app.

- **Live Data Visualization**: Develop an integrated dashboard to track detection history, species-wise distribution, and location-wise infestation patterns.

- **Backend Integration**: Linking the app with a **Firebase** or **Django REST API** to store classification logs, user images, or alert local health authorities.

- **Visualization Dashboard**: Build a dashboard to monitor detection history, locations, and potential larval outbreaks in real time.

References

[1] Antonio Arista-Jalife , Mariko Nakano , Zaira Garcia-Nonoal , “Aedes mosquito classification in its larval stage using deep neural networks” (2019)

[2] Meer Shadman Saeed, Syeda Fahima Nazreen, Syed Shah Sufi Azmat Ullah , "Classification of Mosquito Larvae Using Convolutional Neural Network” (2021)

[3] Antonio Arista-Jalife , Mariko Nakano , Zaira Garcia-Nonoal , “Mosquito Larvae classification using Deep Learning” (2019)

[4] Keun Young Lee1 , Namil Chung , Suntae Hwang , “Application of an artificial neural network (ANN) model for predicting mosquito abundances in urban areas” (2015)

[5] W. D. M. De Silva , S. Jayalal, “Dengue mosquito larvae classification using digital images” (2020)

[6] Sandhu, Muhammad Abdullah , “Dengue larvae classification and tracking using CNN and kalman filtering” (2023)

[7] Md Shakhawat Hossain , Md Ezaz Raihan , Md Sakir Hossain , “Aedes Larva classification Using Ensemble Learning to Prevent Dengue Endemic” (2022)

[8] A. Sanchez-Ortiz, A. Fierro-Radilla, A. Arista-Jalife , “Mosquito Larva Classification Method Based on Convolutional Neural Networks ” (2017)

[9] Aswin Surya, David B. Peral, Austin VanLoon, Akhila Rajesh , “A Mosquito is Worth 16x16 Larvae: Evaluation of Deep Learning Architectures for Mosquito Larvae Classification” (2022)

[10]: Siti Azirah Asmai et al., International Journal of Advanced Trends in Computer Science and Engineering, “ Aedes Mosquito Larvae Recognition with A Mobile App” (2020)

[11] Aqsha Biyano Tresan Andanaputra,Hanry Ham, “Mobile Based Application of Mosquito Larvae Checking Reports : Malaka Sari Village Case” (2020)

[12] Pratana Kukieattikool, Ladawan Klinkusoom, Watcharakon Noothong, “Improvements to the Aedes larvae mobile detection system” (2020)