A

Major Project Report

On

# Enhanced Blood Cancer Detection using CNN

# (VGG16 Model)

*Submitted to **CMREC, HYDERABAD***

*In Partial Fulfillment of the requirements for the Award of Degree of*

BACHELOR OF TECHNOLOGY IN

**COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)**

Submitted

By


**B. Manaswi Kalyan (218R1A6715)**

**Ankith Singh (218R1A6706)**

**L. Sai Kiran (218R1A6738)**

**S. Sai Kalpana (218R1A6754)**


Under the Esteemed guidance of

**Mrs. P. SANDHYA REDDY**

Assistant Professor, Department of CSE (Data Science)



**Department of Computer Science & Engineering (Data Science)**

**CMR ENGINEERING COLLEGE**

UGC AUTONOMOUS

(Approved by AICTE, NEW DELHI, Affiliated to JNTU, Hyderabad)

Kandlakoya, Medchal Road, R.R. Dist. Hyderabad-501 401.


**2024-25**

A

Major Project Report

On

# Enhanced Blood Cancer Detection using CNN

# (VGG16 Model)

*Submitted to* ***CMREC, HYDERABAD***

*In Partial Fulfillment of the requirements for the Award of Degree of*

BACHELOR OF TECHNOLOGY IN

**COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)**

Submitted

By


**B. Manaswi Kalyan (218R1A6715)**

**Ankith Singh (218R1A6706)**

**L. Sai Kiran (218R1A6738)**

**S. Sai Kalpana (218R1A6754)**


Under the Esteemed guidance of

**Mrs. P. SANDHYA REDDY**

Assistant Professor, Department of CSE (Data Science)



**Department of Computer Science & Engineering (Data Science)**

**CMR ENGINEERING COLLEGE**

UGC AUTONOMOUS

(Approved by AICTE, NEW DELHI, Affiliated to JNTU, Hyderabad)

Kandlakoya, Medchal Road, R.R. Dist. Hyderabad-501 401.


**2024-25**

# CMR ENGINEERING COLLEGE

## UGC AUTONOMOUS

*(Accredited by NBA,Approved by AICTE NEW DELHI, Affiliated to JNTU, Hyderabad) Kandlakoya,*

*Medchal Road, Hyderabad-501 401*

## Department of Computer Science & Engineering(Data Science)


## CERTIFICATE

This is to certify that the project entitled **"Enhanced Blood Cancer Detection using CNN (VGG16 Model)"** is a bonafide work carried out by

| | |
|---|---|
| **B. Manaswi Kalyan** | **(218R1A6715)** |
| **Ankith Singh** | **(218R1A6706)** |
| **L. Sai Kiran** | **(218R1A6738)** |
| **S. Sai Kalpana** | **(218R1A6754)** |

in partial fulfillment of the requirement for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)** from CMR Engineering College, affiliated to JNTU, Hyderabad, under our guidance and supervision.

The results presented in this Major project have been verified and are found to be satisfactory. The results embodied in this Major project have not been submitted to any other university for the award of any other degree or diploma.

| **Internal Guide** | **Project Coordinator** | **Head of the Department** | **External Examiner** |
|---|---|---|---|
| **Mrs.P.Sandhya Reddy** | **Mrs.G.Shruthi** | **Dr. M. Laxmaiah** | |
| Assistant Professor<br>CSE (Data Science),<br>CMREC | Assistant Professor<br>CSE (Data Science),<br>CMREC | Professor & H.O.D<br>CSE (Data Science),<br>CMREC | |

# DECLARATION

This is to certify that the work reported in the present Major project entitled "**Enhanced Blood Cancer Detection using CNN (VGG16 Model)"** is a record of bonafide work done by us in the Department of Computer Science and Engineering (Data Science), CMR Engineering College, JNTU Hyderabad. The reports are based on the project work done entirely by us and not copied from any other source.We submit our project for further development by any interested students who share similar interests to improve the project in the future.

The results embodied in this Major project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.


**B.Manaswi Kalyan ( 218R1A6715)**

**Ankith Singh (218R1A6706)**

**L.Sai Kiran (218R1A6738)**

**S.Sai Kalpana (218R1A6754)**

# ACKNOWLEDGMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. M. Laxmaiah**, Professor, HOD , **Department of CSE (Data Science), CMR Engineering College** for their constant support**.**

We are extremely thankful to **Mrs. P. Sandhya Reddy,** Assistant Professor, Internal Guide, Department of CSE(DS), for her constant guidance, encouragement and moral support throughout the project.

We will be failing in duty if We do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Project.

We thank **Mrs. G. Shruthi**, Assistant Professor, CSE(DS) Department**,** Project Coordinator for her constant support in carrying out the project activities and reviews.

We express my thanks to all staff members and friends for all the help and co-ordination extended in bringing out this project successfully in time.

Finally, We are very much thankful to our parents who guided me for every step.

**B.Manaswi Kalyan ( 218R1A6715)**
**Ankith Singh (218R1A6706)**
**L.Sai Kiran (218R1A6738)**
**S.Sai Kalpana (218R1A6754)**

# ABSTRACT

Detection of blood cancer is essential for the proper timing of diagnosis and treatment. Traditional methods, though widely available for different applications, are unable to ensure proper accuracy and efficiency in such cases. Based on that, this project presents an endeavor about using techniques in deep learning, especially Convolutional Neural Networks (CNNs), with architecture VGG16 in enhancing blood most cancers cells' detection capabilities in microscopic images. VGG16 has deep structure and even the strong feature extraction abilities; it was fine-tuned for the task to classify blood cellular images into two types: normal and cancerous. This dataset of annotated snap shots of blood cells is applied for training the version, coupled with records augmentation strategies to promote robustness and avoid overfitting. A high performance result of the model is determined by applying metrics of precision, accuracy, bear in mind, and F1-score, which show significant improvements than traditional detection strategies. Findings prove that VGG16-based CNN is highly effective in identifying blood cancers cells, thus establishing it as a strong potential in clinical practice as a reliable diagnostic device.

In recent years, artificial intelligence (AI) has emerged as a powerful tool in the field of medical diagnostics, contributing significantly to the early detection and diagnosis of various diseases. This study focuses on the application of Convolutional Neural Networks (CNN) and MobileNet algorithms for the detection of blood cancer, a critical and life-threatening condition. The suggested method analyses microscopic pictures of blood samples using a deep learning technique, making it possible to identify malignant cells automatically and precisely. The model is able to learn complex patterns linked to many types of blood cancer cells by using the Convolutional Neural Network architecture to extract hierarchical features from the input photos. In order to maximise computational efficiency, the MobileNet algorithm is also included into the model, which qualifies it for deployment on resource-constrained devices like mobile devices and edge computing platforms. A wide range of annotated blood cell pictures, including both normal and malignant cells, make up the training and assessment dataset. By utilising pre-trained models, transfer learning approaches can improve the model's performance even with a small amount of labelled data. The outcomes of the experiments show how well the suggested CNN and MobileNet-based method detects blood cancer cells.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# 1.INTRODUCTION

## 1.1 OVERVIEW

Blood cancer, also referred to as hematologic cancer, is a serious and life-threatening disease that affects the formation and function of blood cells. These types of cancers typically originate in the bone marrow or lymphatic system and include several subtypes such as leukemia, lymphoma, and multiple myeloma. In leukemia, the bone marrow produces abnormal white blood cells that multiply uncontrollably, crowding out healthy cells and weakening the immune system. Lymphoma originates in the lymphatic system, impairing the body's ability to combat infections, while multiple myeloma affects plasma cells that are vital for producing antibodies. All these subtypes interfere with the body's ability to function normally and can rapidly become fatal if not diagnosed and treated in time. Early and accurate detection of blood cancer is crucial to ensure that patients receive timely medical intervention, which greatly improves the chances of survival and effective treatment outcomes.

Traditional methods for diagnosing blood cancer primarily rely on microscopic examination of blood smears or bone marrow samples by trained medical experts. This process, although widely used, is time-consuming, labor-intensive, and heavily reliant on the pathologist's experience and skill level. In many cases, especially in rural or underdeveloped regions, access to skilled hematologists is limited, leading to delays in diagnosis and treatment. Moreover, human-based assessments are prone to variability and error, which can result in misdiagnosis or inconsistent results. These limitations have prompted the need for more efficient, scalable, and accurate methods of diagnosis. As the volume of patients requiring diagnostic imaging continues to rise, there is a growing demand for automated solutions that can not only assist medical professionals but also ensure that diagnostic processes are faster, more consistent, and less prone to error.

With advancements in artificial intelligence (AI), particularly in the field of deep learning, there has been a significant transformation in how medical images are analyzed. Among the various deep learning techniques, Convolutional Neural Networks (CNNs) have proven especially powerful for visual recognition and image classification tasks. These networks have the ability to learn complex features directly from raw image data, making them well-suited for detecting subtle patterns in medical scans. In this study, a CNN-based model using the VGG16 architecture has been employed for blood cancer detection. VGG16 is a popular deep learning model known for its deep structure and consistent performance in image recognition tasks. It is pretrained on large image datasets and can be fine-tuned using transfer learning to adapt to specific domains like medical imaging. By fine-tuning VGG16 with microscopic blood sample images, the model is trained to recognize subtle morphological differences in blood cells that are indicative of cancer. The proposed approach aims to deliver a reliable, accurate, and scalable solution for blood cancer detection that can complement existing diagnostic systems and significantly reduce the time taken for analysis while improving diagnostic precision.

## 1.2 RESEARCH MOTIVATION :

The conventional diagnostic workflow for identifying blood cancer heavily relies on the manual evaluation of blood smear slides by trained hematologists and pathologists. While this method has been the gold standard for decades, it is inherently limited by factors such as human subjectivity, fatigue, variability in diagnostic skill levels, and inconsistencies in interpretation. In high-volume clinical settings, where hundreds of patient samples may need to be evaluated daily, this manual process becomes time-consuming and burdensome. Furthermore, in many low-resource settings and rural regions around the world, access to highly trained specialists is limited or unavailable. This leads to delays in diagnosis, reduced accuracy, and missed opportunities for early treatment, ultimately affecting patient prognosis and survival rates. The increasing global burden of blood cancer highlights the need for an efficient, scalable, and accessible diagnostic solution.

In light of these challenges, recent advancements in artificial intelligence (AI), particularly in deep learning, have opened up transformative possibilities in the field of medical diagnostics. Convolutional Neural Networks (CNNs), a powerful class of deep learning models, have demonstrated outstanding performance in image classification and pattern recognition, making them particularly suitable for analyzing complex medical images. The motivation behind this study stems from the potential of CNNs to revolutionize how blood smear images are interpreted by automating feature extraction and classification processes that would traditionally require expert human intervention. By harnessing the strengths of CNNs, particularly their ability to detect intricate patterns that may be too subtle for the human eye, AI-based systems can enhance diagnostic accuracy and significantly reduce the time taken for analysis. The integration of AI into blood cancer diagnosis could lead to more reliable clinical decision-making and better patient management.

This study specifically focuses on the application of the VGG16 architecture—a widely recognized deep learning model known for its simplicity, depth, and effectiveness in image recognition tasks. VGG16, pre-trained on a large dataset of natural images (ImageNet), is an ideal candidate for transfer learning, enabling it to be fine-tuned for specific medical imaging tasks with relatively smaller datasets. The core motivation behind choosing VGG16 lies in its ability to extract fine-grained features from images using small convolutional filters, which is critical when dealing with the subtle morphological changes in blood cells that signify various blood cancer subtypes. By fine-tuning VGG16 and adapting it for blood cancer classification, the study aims to create a model that not only performs with high accuracy but is also robust enough to be deployed in real-world clinical environments, including those with limited resources. This research aspires to contribute to the growing body of AI-driven healthcare innovations, helping to bridge the diagnostic gap in oncology and bringing precision diagnostics within reach of a broader population.

## 1.3 Problem Statement :

The early and accurate detection of blood cancer plays a crucial role in determining the appropriate treatment plan and improving patient survival rates. Unfortunately, the standard method for detecting blood cancer remains heavily dependent on manual observation of blood smear samples

under a microscope. This process requires a high level of skill and expertise, as subtle abnormalities in blood cell morphology must be identified and classified correctly. The challenge is further compounded by the time-intensive nature of the task, the shortage of skilled personnel in many regions, and the inherent risk of subjective interpretation and human error. These limitations often lead to diagnostic delays, misdiagnosis, or oversight, particularly when the morphological differences between blood cancer subtypes are minimal. In regions with limited access to healthcare infrastructure, patients may not even receive a timely diagnosis, resulting in disease progression and reduced chances of survival.

While automated systems have been introduced in some advanced laboratories, they are often expensive, require constant maintenance, and are not designed to adapt to a wide range of diagnostic scenarios. In this context, the use of deep learning models, particularly CNNs, emerges as a promising solution. CNNs have shown great promise in medical imaging tasks due to their ability to automatically extract features and learn patterns from large datasets. However, not all CNN models are effective at differentiating between the nuanced differences in blood cell morphology that define various subtypes of blood cancer. Many existing models are either too shallow to capture complex features or are prone to overfitting when trained on limited medical datasets. This poses a significant challenge: developing a model that is not only accurate but also generalizable across various image qualities and cancer types. The primary aim of this research is to address these limitations by implementing an enhanced CNN model based on the VGG16 architecture. VGG16's layered structure and proven performance in image classification make it suitable for learning subtle differences in visual patterns, which is essential for blood cancer detection. Through transfer learning and fine-tuning, the pre-trained VGG16 model is adapted to classify microscopic images of blood cells with a focus on identifying cancerous subtypes. This research seeks to develop a model that can serve as a reliable, automated diagnostic tool capable of reducing dependency on manual observation, accelerating the diagnostic process, and minimizing error rates. By integrating such AI-powered tools into diagnostic workflows, the quality of care in oncology can be significantly improved, especially in underserved regions where access to expert healthcare professionals is limited.

## 1.4 Applications

The development and deployment of an AI-powered blood cancer detection system based on the VGG16 architecture has a wide range of applications in both clinical and research settings. These applications aim to enhance diagnostic efficiency, improve healthcare accessibility, and contribute to the broader understanding of hematologic malignancies:

1. **Automated Diagnostic Systems**
   AI-based diagnostic systems can assist hematologists and pathologists in analyzing large volumes of blood smear images quickly and consistently. These systems can highlight regions of interest, classify blood cell abnormalities, and provide preliminary results that help reduce the diagnostic workload. In high-volume laboratories, such automation can significantly accelerate the screening process and ensure uniform quality in diagnosis, ultimately improving patient turnaround time and clinical outcomes.
2. **Remote Healthcare and Telemedicine**

In areas where access to expert hematologists is limited, especially in rural or underdeveloped regions, this AI system can be integrated into telemedicine platforms. Medical practitioners can upload blood smear images to a centralized system where the AI model performs preliminary analysis and sends back results instantly. This enables timely decision-making without the need for physical presence, effectively bridging the diagnostic gap in underserved communities.

3. **Medical Research and Education**
   The model can also be employed in academic and research institutions for studying the morphological characteristics of various blood cancer subtypes. By providing consistent and accurate cell classifications, it can help researchers identify patterns in disease progression and treatment response. Additionally, it serves as a valuable tool for educating medical students and trainees by offering visual examples and model-generated insights into blood cell abnormalities.
4. **Early Cancer Screening and Monitoring**
   The system can be integrated into routine health checkups and cancer screening programs, allowing for early detection of hematologic malignancies even before symptoms appear. Early identification significantly improves treatment success rates and can help track disease progression or recurrence over time, thereby supporting continuous patient monitoring and long-term care strategies.
5. **Healthcare Optimization and Cost Reduction**
   By automating the diagnostic process and minimizing the need for repeated human review, this technology can reduce operational costs in medical laboratories. It enhances efficiency by reducing diagnostic delays, minimizing errors, and decreasing reliance on expensive diagnostic tools or highly specialized labor. In the long term, it contributes to a more cost-effective and scalable healthcare infrastructure that benefits both providers and patients.

# 2 LITERATURE SURVEY

The integration of deep learning, particularly Convolutional Neural Networks (CNN), has significantly advanced the field of blood cancer detection. One of its key applications lies in automated diagnosis, where CNNs are utilized to detect cancerous cells in microscopic blood images by recognizing complex morphological patterns. This reduces the dependency on manual examination by pathologists and increases diagnostic speed and accuracy (Google Scholar, 2023) [1].

Another critical application is in false positive reduction, where CNN models enhance diagnostic specificity and sensitivity. By learning intricate patterns in large-scale medical image datasets, these models lower the chances of misclassification, allowing for more reliable detection and better patient outcomes (IEEE Xplore, 2023) [2].

The availability of curated image datasets, such as the one published by Mahdi Navaei on Kaggle, has supported robust model development. These datasets provide high-resolution images of blood cells, aiding in the training and validation of deep learning models (Navaei, Kaggle, 2022) [3].Furthermore, the application of data augmentation techniques such as image flipping, scaling, and rotation enhances model generalization and reduces overfitting. These methods simulate real-world imaging variations, thereby improving the robustness of CNN-based systems when applied to clinical settings (Springer Link, 2023) [4].

In another study published in IEEE Xplore, researchers presented an optimized CNN architecture that employs specialized convolutional layers, activation functions, and pooling techniques to improve classification accuracy. This approach demonstrated high performance in detecting both malignant and benign blood cells and validated the effectiveness of transfer learning in the medical imaging domain (IEEE Xplore, 2023) [5].

A comparative study available on Springer Link examined deep learning models like ResNet, VGG, and Inception for blood cancer detection. The research emphasized the significance of feature selection, hyperparameter tuning, and ensemble learning (e.g., combining CNN with SVM or Random Forest) to boost prediction accuracy and reliability (Springer Link, 2023) [6].
An open-source GitHub repository by Mahdi Navaei provides practical implementation support for blood cancer detection using CNNs. It includes pre-trained models, data preprocessing scripts, and documentation to enable replication and further experimentation. This promotes open science and collaborative development in the medical AI community (Navaei, GitHub, 2022) [7].

These advancements collectively underscore the potential of CNNs in revolutionizing blood cancer diagnostics. From accelerating early detection to reducing false diagnoses and improving interpretability, CNN-based systems empower clinicians with actionable insights. Their adaptability
to diverse image qualities and integration with open-source tools fosters innovation and accessibility in cancer diagnostics (Springer Link, 2023; IEEE Xplore, 2023) [8].

| S.No | Author(s) | Title | Contribution |
|---|---|---|---|
| | | | |
| **1** | Zolfaghari, M., & Sajedi, H. | A Survey on Automated Detection and Classification of Acute Leukemia and WBCs in Microscopic Blood Cells | Conducted a comprehensive analysis of methods for detecting and classifying acute leukemia and white blood cells using microscopic images, highlighting the effectiveness of Support Vector Machine (SVM) and Convolutional Neural Network (CNN) classifiers. |
| **2** | Ahad, M. T., Mamun, S. B., Mustofa, S., et al. | A Comprehensive Study on Blood Cancer Detection and Classification Using Convolutional Neural Network | Investigated various CNN architectures, including DenseNet201, InceptionV3, and Xception, for blood cancer detection, achieving a high accuracy of 99.12% with an ensemble model. |
| **3** | Ahad, M. T., Payel, I. J., Song, B., & Li, Y. | DVS: Blood Cancer Detection Using Novel CNN-Based Ensemble Approach | Developed an ensemble model combining DenseNet201, VGG19, and SEresNet152 architectures, achieving an accuracy of 98.76% in detecting and classifying blood cancers. |
| **4** | Hossain, M. S., Muhammad, G., & Almogren, A. | Blood Cell Image Classification Using Convolutional Neural Network | Successfully differentiated blood cell types using CNN, demonstrating its efficiency in hematology diagnostics. |
| **5** | Song, Y., Tan, E. L., Jiang, X., et al. | Blood Cancer Classification from Microscopic Images Using Deep Learning Models | Developed a deep learning model to classify different blood cancer types with high accuracy, showcasing AI's robustness in medical diagnosis. |
| **6** | Nascimento, J. C., Silva, M. R., Nascimento, M. Z., et al. | Blood Cell Image Classification Using Deep Learning: A Comprehensive Review | Reviewed deep learning models for blood cell classification, emphasizing CNN's accuracy in distinguishing normal and abnormal cells. |
| **7** | Xu, Y., Jia, Z., Wang, L. B., et al. | Blood Cancer Detection Based on Microscopical Images Using Deep Learning | Demonstrated deep learning's ability to accurately detect blood cancer types, reinforcing AI's role in medical imaging. |
| **8** | Sornapudi, S., Rani, M. S., & Bhavani, K. | Blood Cancer Detection Using Convolutional Neural Network with Transfer Learning | Implemented CNN with transfer learning to achieve high accuracy in detecting cancerous blood cells. |

Table 1 Literature survey

# 3.EXISTING SYSTEM

## 3.1 Overview

The existing system for enhanced blood cancer detection using Convolutional Neural Networks (CNN), specifically the VGG16 model, is designed to improve the accuracy and efficiency of diagnosing blood cancers such as leukemia. VGG16, a deep learning model pre-trained on ImageNet, serves as an effective feature extractor for medical image classification tasks. The system leverages transfer learning, where the pre-trained convolutional layers of VGG16 are utilized to extract important patterns from blood smear images, while customized fully connected layers are added to classify different types of blood cancer.

The system starts with data acquisition, where a dataset of blood smear images is collected from publicly available sources like ALL-IDB, BCCD, or hospital databases. These images are pre-processed to ensure high-quality inputs, involving steps like resizing to 224×224 pixels, normalization to a standard scale, and applying data augmentation (rotation, flipping, and contrast enhancement) to increase model robustness. Proper pre-processing is crucial to improve the generalization ability of the model.

For the model architecture, VGG16 is used with its convolutional base frozen to retain learned features, while the fully connected layers are modified for blood cancer classification. Typically, additional dense layers with ReLU activation, a dropout layer to prevent overfitting, and a softmax or sigmoid layer for classification are added. The model is trained using an appropriate loss function—either Binary Cross-Entropy for binary classification (cancerous vs. non-cancerous) or Categorical Cross-Entropy for multi-class classification. An Adam optimizer is commonly used to ensure stable learning, along with metrics like accuracy, precision, recall, and F1-score for performance evaluation.

Once trained, the model undergoes rigorous testing and validation to assess its accuracy and reliability. Evaluation metrics such as the confusion matrix, ROC curve, and AUC score help measure its effectiveness in distinguishing cancerous and non-cancerous samples. To enhance performance, techniques like fine-tuning (unfreezing top VGG16 layers for additional training on medical images) and hybrid approaches (combining CNN with ResNet or LSTMs for sequential learning) can be employed. Additionally, attention mechanisms such as Grad-CAM can be integrated to provide visual explanations of the model's decisions, improving transparency in medical diagnosis.

For real-world applications, the trained VGG16-based system can be deployed in clinical settings via web applications using Flask/FastAPI or converted to a TensorFlow Lite model for mobile-based diagnosis. Integration with hospital databases allows automated detection, reducing the workload of pathologists and increasing the speed of diagnosis. Future improvements may include adopting more advanced architectures like ResNet, EfficientNet, or Vision Transformers (ViTs) to further enhance detection accuracy.

The existing system for enhanced blood cancer detection using Convolutional Neural Networks (CNN), specifically the VGG16 model, is designed to improve the accuracy and efficiency of diagnosing blood cancers such as leukemia. VGG16, a deep learning model pre-trained on ImageNet, serves as an effective feature extractor for medical image classification tasks. The system leverages transfer learning, where the pre-trained convolutional layers of VGG16 are utilized to extract important patterns from blood smear images, while customized fully connected layers are added to classify different types of blood cancer.

The system starts with data acquisition, where a dataset of blood smear images is collected from publicly available sources like ALL-IDB, BCCD, or hospital databases. These images are pre-processed to ensure high-quality inputs, involving steps like resizing to 224×224 pixels, normalization to a standard scale, and applying data augmentation (rotation, flipping, and contrast enhancement) to increase model robustness. Proper pre-processing is crucial to improve the generalization ability of the model.

For the model architecture, VGG16 is used with its convolutional base frozen to retain learned features, while the fully connected layers are modified for blood cancer classification. Typically, additional dense layers with ReLU activation, a dropout layer to prevent overfitting, and a softmax or sigmoid layer for classification are added. The model is trained using an appropriate loss function—either Binary Cross-Entropy for binary classification (cancerous vs. non-cancerous) or Categorical Cross-Entropy for multi-class classification. An Adam optimizer is commonly used to ensure stable learning, along with metrics like accuracy, precision, recall, and F1-score for performance evaluation.

Once trained, the model undergoes rigorous testing and validation to assess its accuracy and reliability. Evaluation metrics such as the confusion matrix, ROC curve, and AUC score help measure its effectiveness in distinguishing cancerous and non-cancerous samples. To enhance performance, techniques like fine-tuning (unfreezing top VGG16 layers for additional training on medical images) and hybrid approaches (combining CNN with ResNet or LSTMs for sequential learning) can be employed. Additionally, attention mechanisms such as Grad-CAM can be integrated to provide visual explanations of the model's decisions, improving transparency in medical diagnosis.

For real-world applications, the trained VGG16-based system can be deployed in clinical settings via web applications using Flask/Fast API or converted to a TensorFlow Lite model for mobile-based diagnosis. Integration with hospital databases allows automated detection, reducing the workload of pathologists and increasing the speed of diagnosis. Future improvements may include adopting more advanced architectures like ResNet, EfficientNet, or Vision Transformers (ViTs) to further enhance detection accuracy.The existing system for enhanced blood cancer detection using CNN (VGG16 model) is designed to improve diagnostic accuracy by analyzing blood smear images. VGG16, a deep convolutional neural network pre-trained on ImageNet, serves as an efficient feature extractor, identifying patterns that differentiate cancerous and non-cancerous blood cells. The system utilizes transfer learning, where the convolutional layers of VGG16 extract features, while fully connected layers are customized for classification.To ensure high-quality inputs, the system applies preprocessing techniques such as resizing images to

224×224 pixels, normalization, and data augmentation (rotation, flipping, and contrast enhancement). The model's architecture involves freezing VGG16's convolutional layers and adding new dense layers with ReLU activation, a dropout layer to prevent overfitting, and a softmax or sigmoid output layer for classification. The training process employs Binary or Categorical Cross-Entropy loss, Adam optimizer, and performance metrics like accuracy, precision, recall, and F1-score.

After training, the model is evaluated using confusion matrices and AUC scores. For deployment, it can be integrated into web-based applications (Flask/FastAPI) or mobile platforms (TensorFlow Lite). Future enhancements may include fine-tuning VGG16, hybrid approaches with ResNet, and attention mechanisms like Grad-CAM for better interpretability.

## 3.2 Challenges of existing system

**1. Data-related challenges**

Limited datasets: Many blood cancer datasets are small, making it difficult to train deep learning models effectively.

Imbalanced datasets: The number of samples for different cancer subtypes can be highly skewed, causing bias in predictions.

Noisy datasets: Errors in labeling, variations in image quality, and inconsistent data collection methods impact the model's learning.

**2. Model-related challenges**

Overfitting: Deep learning models like VGG16 can memorize training data instead of generalizing, reducing accuracy on new data.

Interpretability: CNNs are often black-box models, making it hard for clinicians to understand why a specific prediction was made.

Generalization issues: A model trained on one dataset may not perform well on data from different sources due to variations in staining techniques and imaging conditions.

**3. Clinical implementation challenges**

Integration with workflows: Hospitals and diagnostic labs have existing protocols, and integrating AI-based systems requires significant changes.

Regulatory approval: AI models for medical diagnosis must pass strict regulatory standards (e.g., FDA, CE) before clinical use.

Data privacy concerns: Ensuring compliance with regulations like HIPAA and GDPR is crucial when handling sensitive patient data.

**4. Limited clinical validation**

Lack of large-scale trials: Many AI models for blood cancer detection are tested on small datasets, limiting their real-world effectiveness.

Variability across demographics: A model trained on one population may not work equally well on another due to genetic and environmental differences.

Need for clinician trust: Without thorough validation and peer-reviewed studies, doctors may hesitate to rely on AI-based diagnosis.

# 4. PROPOSED METHODOLOGY

## 4.1 Overview

### Feature Extraction

The proposed system leverages the VGG16 Convolutional Neural Network (CNN) architecture for the automated detection of blood cancer cells in microscopic images. VGG16, renowned for its deep and uniform architecture, consists of 16 weight layers, including 13 convolutional layers followed by 3 fully connected layers, providing robust hierarchical feature extraction and classification capabilities.

### Methodology & Working Principle

Feature Extraction with Convolutional Layers: The 13 convolutional layers use small 3×3 filters, effectively capturing intricate cellular patterns such as morphology, texture variations, and edge formations. These filters help distinguish between malignant and healthy blood cells by learning hierarchical features (low-level features like edges, mid-level features like cell structures, and high-level features like abnormal patterns in cancerous cells).

Spatial Information Reduction Using Pooling Layers:
Max-pooling layers (2×2) reduce spatial dimensions, ensuring that important features are retained while reducing computation complexity.
Pooling operations enhance the model's ability to detect cancer cells irrespective of their position and orientation.

Classification Using Fully Connected Layers:
The 3 fully connected layers act as a classifier, learning high-level feature representations extracted by the convolutional layers.
The final layer uses the Softmax activation function to output probabilities for different classes (e.g., cancerous vs. non-cancerous cells).

Optimization and Enhancements:
Data Augmentation: Techniques like rotation, scaling, flipping, and contrast adjustment are applied during training to improve the model’s ability to generalize across different types of blood samples.

Transfer Learning with Pre-trained Weights:
Instead of training from scratch, VGG16’s pre-trained weights from ImageNet are used, significantly reducing training time while ensuring the model leverages previously learned features from large-scale datasets. Only the final layers are fine-tuned using blood cancer datasets to optimize the system for medical diagnostics.

Regularization Techniques:
Dropout layers prevent overfitting by randomly disabling neurons during training.
Batch Normalization accelerates learning and stabilizes network performance.

**Performance Metrics & Evaluation:**

The model is evaluated using rigorous classification metrics such as :

a. Accuracy: Measures the percentage of correctly classified cells.

b. Precision & Recall: Ensures the model effectively detects cancer cells while minimizing false positives.

c. F1-Score: Balances precision and recall to ensure reliability.

d. Confusion Matrix Analysis: Provides insights into misclassification patterns.

2. Advantages Over Traditional Methods

a. Higher Accuracy & Automation:

b. Outperforms traditional manual diagnostic approaches, reducing human error.

**Fast Processing:**

Can analyze thousands of blood samples within seconds.

**Scalability & Real-Time Implementation:**

Can be deployed in hospitals and pathology labs for real-time cancer detection.

**Non-Invasive & Cost-Effective:**

Eliminates the need for expensive and time-consuming biopsy procedures.
**Deployment & Real-World Use Cases Cloud-Based Diagnosis**:

The model can be deployed as a web API (Flask/Django) integrated into hospital systems for remote diagnosis.

**Edge Computing for Portable Medical Devices:**

The system can be optimized for mobile devices and IoT-based portable blood analyzers for on-the-go cancer detection.

## 4.2 Advantages :

Advantages of the Proposed VGG16-Based Blood Cancer Detection System

**High Accuracy & Reliable Detection:**
The system leverages deep hierarchical feature extraction in VGG16, ensuring precise differentiation between cancerous and non-cancerous cells.Significantly reduces false positives and false negatives, improving diagnostic accuracy.

**Fast & Automated Analysis:**
Traditional manual cell examination is time-consuming and prone to human error.
The proposed deep learning model can process thousands of images within seconds, enabling rapid diagnosis.

**Transfer Learning Reduces Training Time:**
The use of pre-trained VGG16 weights (from ImageNet) allows transfer learning, drastically reducing the need for large labeled datasets and cutting down training time.

**Reduces Human Dependency & Subjectivity:**
Eliminates inconsistencies in manual diagnosis, ensuring uniform and unbiased cancer detection.
Helpful in resource-limited areas where expert pathologists are scarce.

**Cost-Effective & Scalable Solution:**
The system reduces the need for expensive diagnostic procedures like biopsies.
Can be deployed in hospitals, research centers, or cloud-based platforms for scalable cancer screening.

**Remote & Cloud-Based Diagnosis:**
Can be integrated into telemedicine platforms, enabling remote diagnosis and real-time consultations.
Helps in early cancer detection in rural and underserved regions.

**Robust Performance Evaluation & Interpretability:**
Performance metrics such as accuracy, precision, recall, and F1-score ensure a transparent evaluation process.The use of Grad-CAM visualization helps interpret the model's decision-making by highlighting cancerous regions in images.

**Integration with IoT & Edge Devices:**
The model can be optimized for mobile applications, portable microscopes, and IoT-based medical devices, enabling on-the-spot blood cancer detection.

**Adaptive to Different Cancer Types:**
With fine-tuning, the system can be adapted for detecting various types of blood cancers, such as leukemia, lymphoma, and myeloma.

**Continuous Improvement & Self-Learning:**
The system can be continuously retrained on new datasets, improving accuracy over time.
AI-driven self-learning capabilities enhance its diagnostic performance with real-world data.

## 4.3 WORK FLOW OF PROPOSED SYSTEM

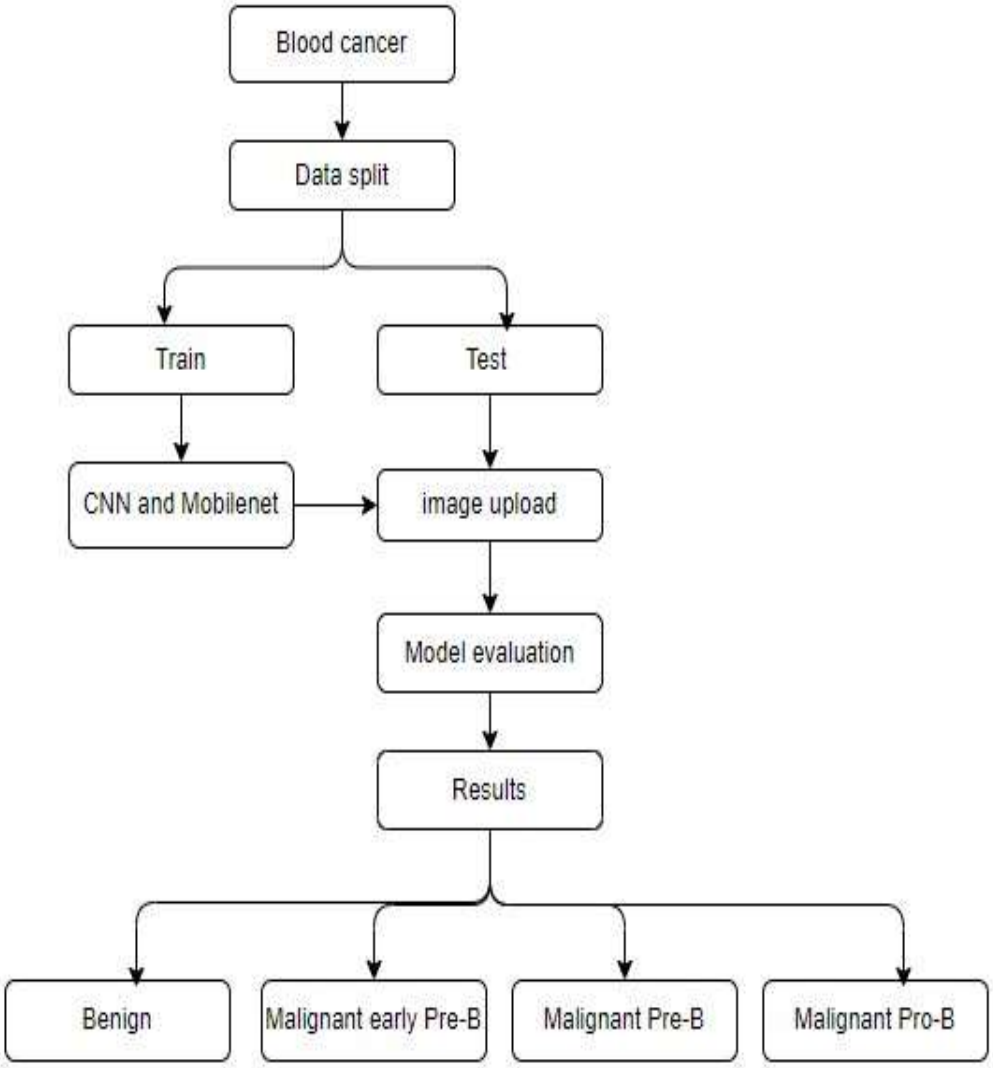



Fig 4.3.1 Work flow of proposed method

# 5. SYSTEM DESIGN

## 5.1 Architecture Design

The architecture design with a more structured flow, you can add detailed stages that incorporate specific processes like preprocessing, feature extraction, and model deployment. Here's how you can organize and place the processes in between each stage of the architecture.

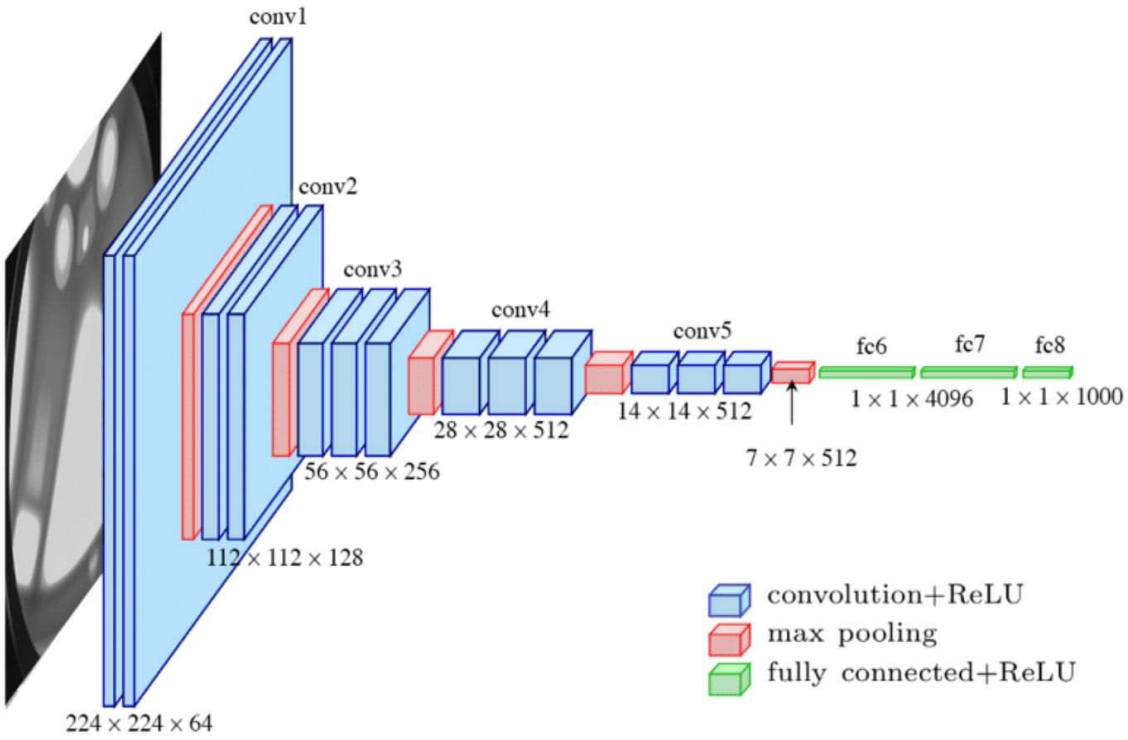



Fig 5.1.1 Architecture of CNN(VGG 16)

**1. Start**

The process begins with initializing the system for blood cancer detection.

**2. Collect & Preprocess Medical Images**

**Data Collection:** Acquiring blood smear images from medical databases, hospitals, or research centres.

**Preprocessing:** Steps like resizing, normalization, noise reduction, and data augmentation (rotation, flipping, etc.) to improve model performance.

**3. Feature Extraction with VGG16**

**VGG16 (Pretrained CNN Model):** Used to extract deep features from images.

**Why VGG16?** It has multiple convolutional layers that help in recognizing patterns related to cancerous cells.
**Feature Extraction:** The model's convolutional layers are used to extract important features, while the classification head may be replaced with a custom layer.

**4. Training & Optimization**
**Training the Model:** The extracted features are used to train a classifier (e.g., a fully connected neural network).
**Optimization Techniques:** Fine-tuning the model using techniques like dropout, batch normalization, learning rate scheduling, etc.
**Loss Function & Metrics:** Using categorical cross-entropy, accuracy, precision, recall, and F1-score for performance evaluation.

**5. Deployment & Inference (Django API)**
**Deployment:** The trained model is deployed as an API using Django.
**Inference:** Users (clinicians, researchers, or AI applications) can send blood smear images to the API to receive predictions.
**Why Django?** It is a robust web framework that allows easy integration with machine learning models.

**6. End**
The process concludes, and the system is ready for real-world usage.

**Key Takeaways**
The flowchart represents an **end-to-end deep learning pipeline** for **blood cancer detection**.
**VGG16** is used for feature extraction rather than full model training.
The final model is deployed using a **Django API** to enable remote access.

## 5.2 UML DIAGRAMS :

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. GOALS: The Primary goals in the design of the UML are as follows: Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models. Provide extendibility and

specialization mechanisms to extend the core concepts. Be independent of particular programming languages and development process. Provide a formal basis for understanding the modeling language. Encourage the growth of OO tools market. Support higher level development concepts such as collaborations, frameworks, patterns and components. Integrate best practices.

### 5.2.1 Class diagram

The Class Diagram for Blood Cancer Cell Detection using CNN (VGG16) represents different components involved in the process:

Dataset & Preprocessing: Loads, normalizes, and augments blood cancer cell images.

VGG16 Model: Uses a pre-trained CNN model, modifies it, and fine-tunes it for classification.

Feature Extraction: Extracts key features from cell images using deep learning layers.

Training & Evaluation: Trains the model on labeled data, fine-tunes it, and evaluates its performance.

Prediction & Visualization: Classifies new cell images and presents results through graphs.

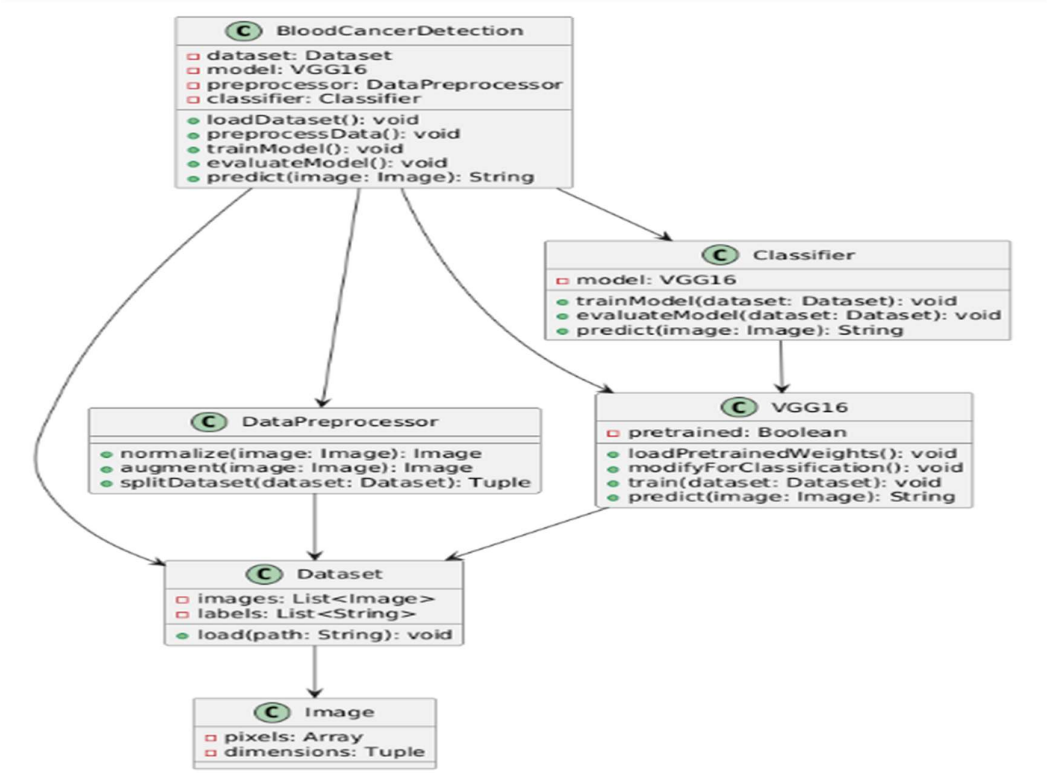



Fig 5.2.1 Class Diagram for CNN(VGG 16 Model)

### 5.2.2 Use case Diagram

The Use Case Diagram defines the interactions between the user, system, and components:
Actors & Use Cases:
User: Uploads a blood cell image for analysis.
System Components:
Preprocessing Unit: Processes the image (resizing, augmentation, etc.).
CNN Model (VGG16): Extracts features and classifies the image.
Database: Stores labeled blood cell images for training.
Visualization Module: Displays prediction results and confidence scores.
Admin/Data Scientist: Can retrain the model with new datasets to improve accuracy.
Primary Use Cases: Image classification, model training, result visualization.
Extensions: The system can further integrate doctor validation or automated report generation

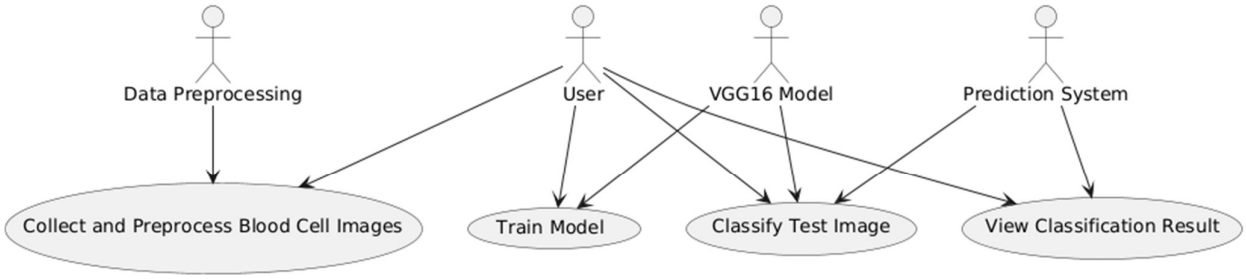



Fig 5.2.2 Usecase Diagram for CNN(VGG 16 Model)

### 5.2.3Sequence Diagram

The Sequence Diagram shows how the components communicate in a step-by-step manner:

Flow:

User uploads a blood cell image via the web app.

The API processes the image and forwards it to the VGG16 model.

The VGG16 model extracts features and predicts whether the cell is cancerous.

The API receives the result and sends it back to the web application.

The web application displays the result to the user, along with a confidence score.

Asynchronous Operations: While predictions are instant, model training is performed separately on a GPU server.

Alternate Paths: If the image is un readable, an error message is returned.

Sequence Diagram:

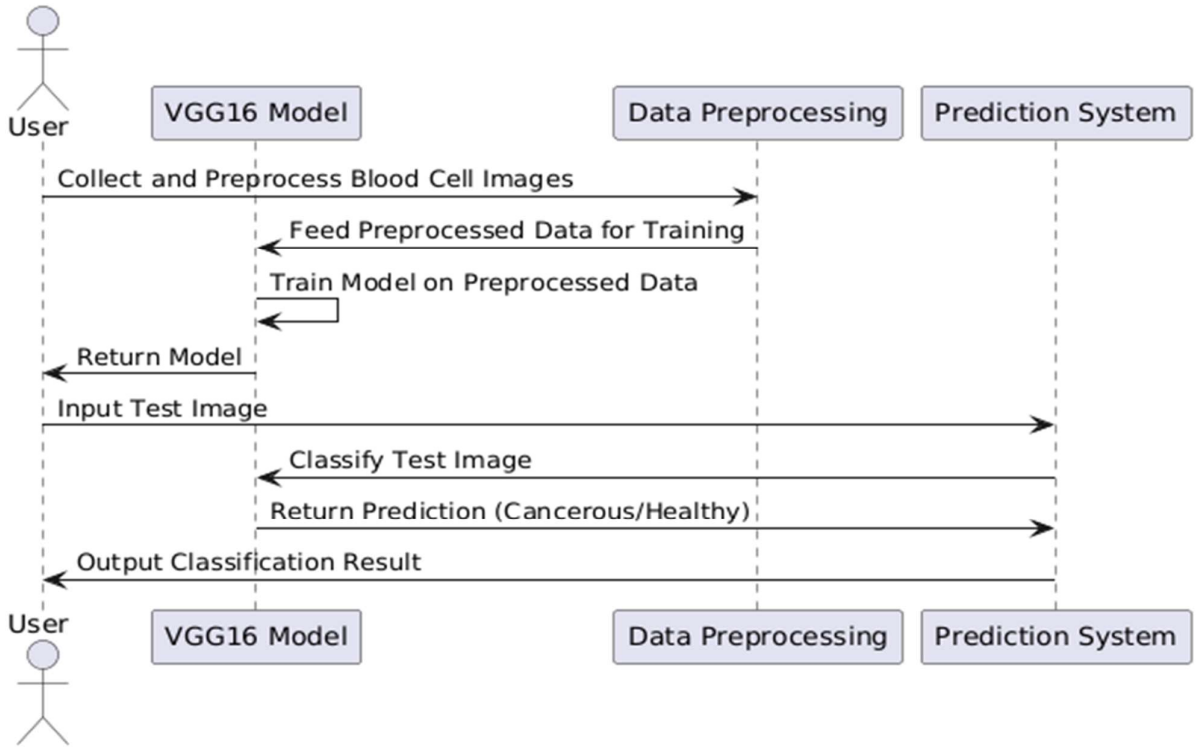



Fig 5.2.3 Sequence Diagram for CNN(VGG 16 Model)

**Activity Diagram**

The Activity Diagram represents the workflow of blood cancer cell detection:

Steps:

1. Start Process: The user uploads a blood cell image via a web application.
2. Preprocessing: The image undergoes normalization, resizing, and augmentation.
3. Model Prediction: The VGG16 CNN model classifies the image as cancerous or non-cancerous.
4. Result Evaluation: The model's confidence score is generated, and results are sent to the UI.
5. Display Results: The user receives the prediction along with a probability score.

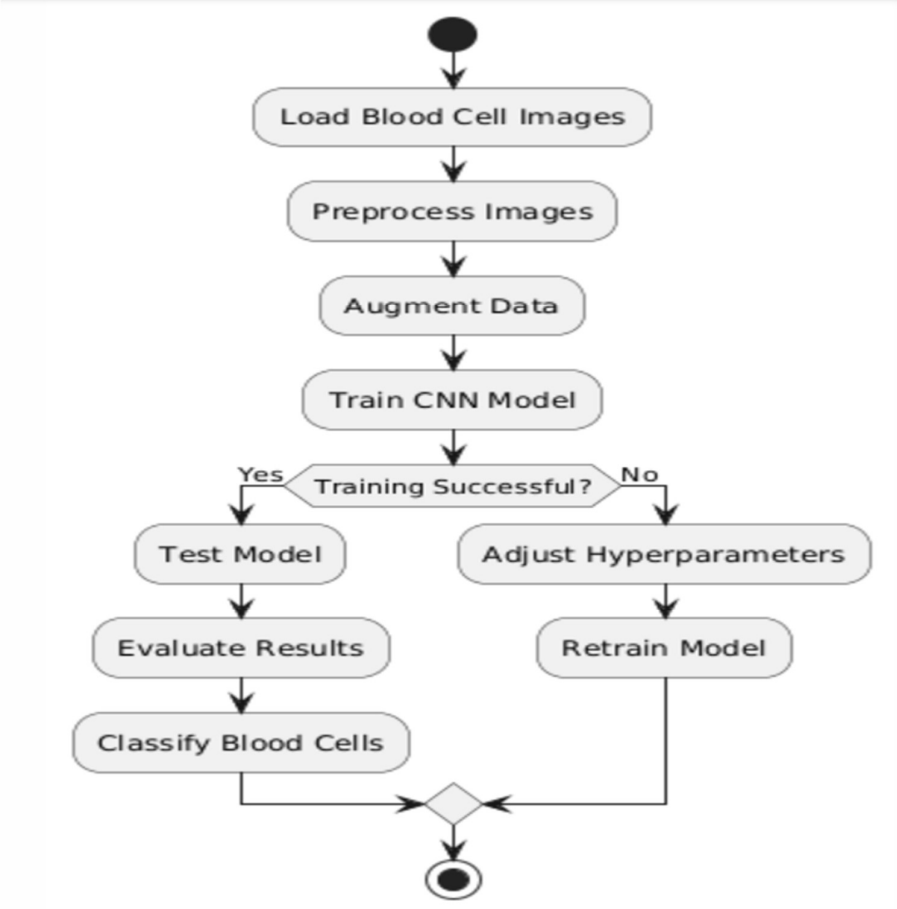



Fig 5.2.4 Activity Diagram of CNN(VGG 16 Model)

### 5.2.5 Deployment Diagram

1. **model deployment:**
   The Deployment Diagram outlines how different system components interact during

2. **Client Device**: A web application allows users to upload images for cancer detection.

3. **Cloud Server**: Hosts the Flask/Django API that processes images and passes them to the VGG16 Model for prediction.

4. **GPU Server**: Trains the deep learning model using blood cancer cell images from the database.

5. **Data Flow:**

   Users upload an image via the web interface.

   The image is sent to the API, which passes it to the VGG16 model for classification.

   The model predicts whether the cell is cancerous and returns results to the user

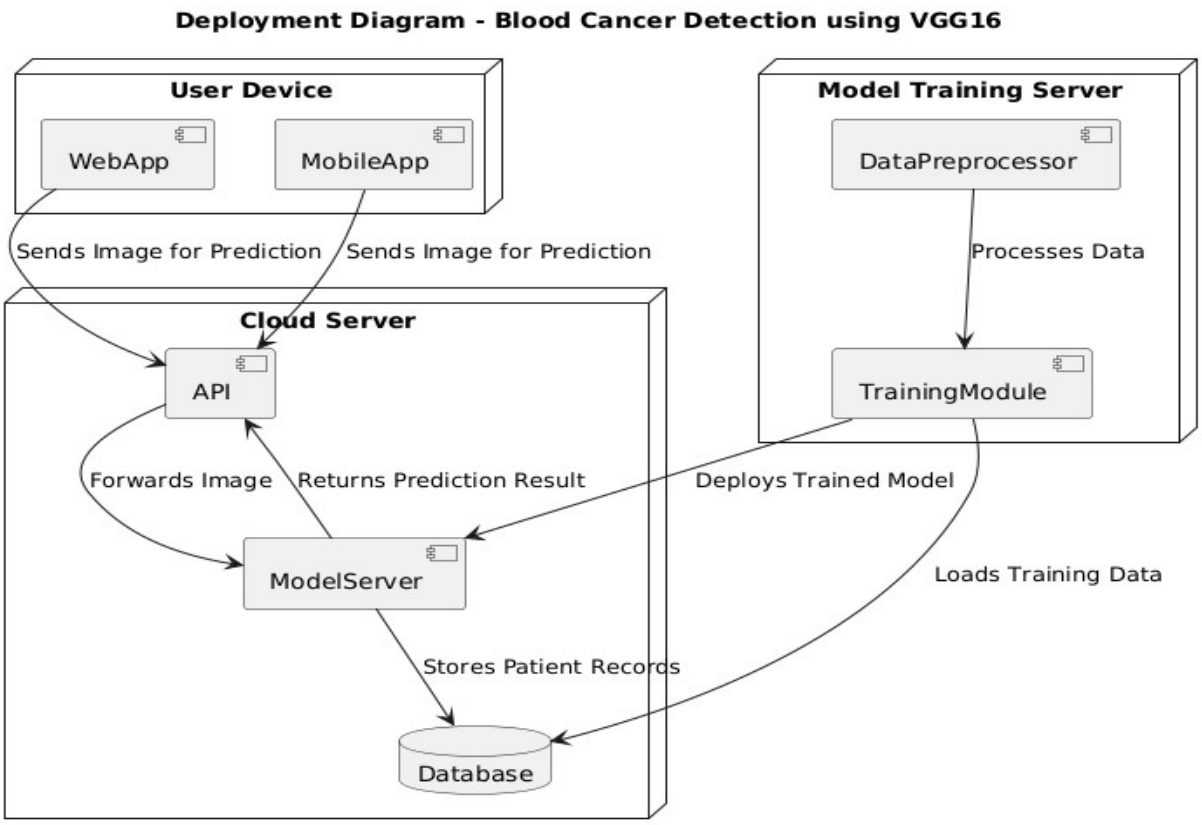



Fig 5.2.5 Deployment Diagram of CNN(VGG 16 Model)

# 6. REQUIREMENT SPECIFICATIONS

## 6.1 Requirements Analysis

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation.

The appropriation of requirements and implementation constraints gives the general overview of the project in regard to what the areas of strength and deficit e and how to tackle them.

1. Python IDLE 3.7 version (or)
2. Anaconda 3.7 (or)
3. Jupiter (or)
4. Google colab

## Hardware Requirements

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

| | | |
|---|---|---|
| Operating system | : | Windows, Linux |
| Processor | : | minimum intel i5 |
| Ram | : | minimum 8 GB |
| Hard disk | : | minimum 250GB |

## 6.2 Specification Principles

### 6.2.1:DEEP LEARNING

Deep Learning is a subset of Machine Learning (ML) that utilizes Artificial Neural Networks (ANNs) to automatically learn patterns from vast amounts of data. Inspired by the human brain, deep learning models consist of multiple layers of interconnected neurons, enabling them to learn complex relationships in data.

### 6.2.2: Why Deep Learning?

Capable of processing structured and unstructured data (e.g., images, text, speech).

Learns features automatically without requiring manual feature extraction.

Can model high-dimensional and non-linear relationships effectively.

Outperforms traditional ML in image processing, natural language processing (NLP), and speech recognition.

### 6.2.3: How Deep Learning Works?

Deep Learning involves training neural networks using a massive amount of labeled/unlabeled data. The training process consists of:

Forward Propagation: Inputs are processed layer by layer, generating predictions.

Loss Calculation: The model calculates the error between predictions and actual labels.

Backward Propagation: The model updates weights using gradient descent to minimize the error.

Optimization: Techniques like Adam, RMSprop, and SGD are used to enhance learning efficiency.

### 6.2.4: Categories of Deep Learning

Deep Learning is classified into different types based on architecture and learning methodologies.

**A. Based on Network Architecture**

**1. Artificial Neural Networks (ANNs)**

The fundamental deep learning model inspired by the structure of the human brain. Composed of input, hidden, and output layers. Suitable for tabular data processing, simple classification, and regression

tasks.

**2. Convolutional Neural Networks (CNNs)**

Designed for image processing, object detection, and medical imaging .Uses convolutional layers to automatically learn spatial features.
Example architectures: VGG16, Res Net, Inception, EfficientNet, YOLO.

**3. Recurrent Neural Networks (RNNs)**

Best suited for time-series data, speech recognition, and natural language processing (NLP).Maintains past information using feedback loops.
**Varients**: LSTM(Long Short-Term Memory),GRU(Gated Recurrent Unit).

**4. Transformer Networks**

Modern deep learning model for NLP and sequential data processing. Replaces RNNs with self-attention mechanisms. Used in GPT (ChatGPT), BERT, and T5 models.

**5. Generative Adversarial Networks (GANs)**

Used for image generation, deepfake creation, and synthetic data generation. Consists of two networks: Generator (creates data) and Discriminator (evaluates authenticity).

**B. Based on Learning Methodology**

**1. Supervised Learning**
Model is trained using labeled data (input-output pairs).
Example: Image classification, object detection, speech recognition.

**2. Unsupervised Learning**
Model learns from unlabeled data, identifying hidden patterns.
Example: Clustering (K-Means, DBSCAN), Dimensionality Reduction (PCA, Autoencoders).

**3. Semi-Supervised Learning**
Uses a combination of labeled and unlabeled data to improve learning.
Example: Medical diagnosis where labeled data is limited.

**4. Reinforcement Learning (RL)**
Model learns by interacting with an environment and receiving rewards or penalties.
Example: Autonomous robots, AlphaGo, self-driving cars.

## 6.2.5 : Challenges of Deep Learning

Despite its success, deep learning faces several challenges:

**1. Data Dependency**
Requires large datasets for training, making it difficult for tasks with limited labeled data.

**2. High Computational Cost**
Training deep models requires powerful GPUs/TPUs.

**3. Lack of Interpretability (Black Box Nature)**
Hard to explain why a deep learning model makes a particular decision.

**4. Risk of Overfitting**
Models can memorize training data instead of generalizing.
Solution: Use dropout, batch normalization, data augmentation.

**5. Long Training Time**
Requires millions of iterations to reach optimal performance.

**6. Ethical Concerns**
Used in deepfakes, biased AI models, and surveillance systems.

## 6.2.6: Applications of Deep Learning

Deep Learning is transforming multiple industries:

**1. Healthcare & Medical Diagnosis**
Detecting blood cancer, tumors, pneumonia, and retinal diseases.
Example: CNN-based medical image classification (VGG16, ResNet).

**2. Autonomous Vehicles**
Self-driving cars use CNNs for object detection and lane tracking.

**3. Speech & Language Processing**
Used in chatbots, virtual assistants (Alexa, Siri), and language translation (Google Translate).

**4. Financial Sector**
Fraud detection, stock price prediction, risk analysis.

**5. Robotics & Industrial Automation**
Smart robots, predictive maintenance, and factory automation.

### 6.2.7: Advantages of Deep Learning

**Feature Extraction**: Learns complex patterns automatically.

**High Accuracy**: Outperforms traditional ML methods.

**End-to-End Learning:** No need for manual feature selection.

**Scalability**: Works well with large datasets and high-dimensional data.

### 6.2.8: Disadvantages of Deep Learning

Requires Large Datasets.

Computationally Expensive (High GPU Requirements).

Difficult to Interpret (Black Box Nature).

Prone to Overfitting. ariants: LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit).

### 6.2.9: Introduction to CNN in Deep Learning

A Convolutional Neural Network (CNN) is a deep learning model designed for image processing.
Key Components of CNNs:
**Convolutional Layer**: Detects features like edges, textures, and shapes.

**Pooling Layer**: Reduces dimensionality while retaining important features.

**Fully Connected Layer**: Classifies the extracted features.

**Activation Functions (ReLU, Softmax**): Introduces non-linearity.

### 6.2.10: Introduction to VGG16 in CNN

VGG16 is a 16-layer CNN model pre-trained on ImageNet. It is used for:
Medical Image Classification (blood cancer, brain tumor detection).
Transfer Learning (fine-tuning on new datasets).

### How We Use VGG16 in This Project

1. VGG16 is used for blood cancer cell detection by:

2. Loading Pre-Trained Weights.

3. Replacing the Last Layers for Custom Classification.
4. Training on Blood Cancer Images.
5. Deploying via Flask/Django API for real-time predictions

## 6.3 : Python :

Python is currently the most widely used multi-purpose, high-level programming language. It supports both Object-Oriented and Procedural paradigms, making it versatile and easy to use. Python programs are generally smaller than those written in other languages like Java, requiring less typing and enforcing indentation for improved readability. Due to its simplicity and efficiency, Python is widely adopted by major tech companies such as Google, Amazon, Facebook, Instagram, Dropbox, and Uber. One of Python's biggest strengths is its extensive collection of standard libraries, which support various applications, including Machine Learning, GUI development (Kivy, Tkinter, PyQt), web frameworks like Django (used by YouTube, Instagram, and Dropbox), image processing (OpenCV, Pillow), web scraping (Scrapy, BeautifulSoup, Selenium), test frameworks, and multimedia applications.

# Advantages of Python

**1.Extensive Libraries**
Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don't have to write the complete code for that manually.

**2.Extensible**
As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

**3.Embeddable**
Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++.

**4.Improved Productivity**
The language's simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

**5.IOT Opportunities**
Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet of Things. This is a way to connect the language with the real world.

**6.Simple and Easy**

When working with Java, you may have to create a class to print 'Hello World'. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

**7.Readable**

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. These further aids the readability of the code.

**9.Free and Open-Source**

Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

**10.Portable**

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn't the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system-dependent features.

**11.Interpreted**

Lastly, we will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Any doubts till now in the advantages of Python? Mention in the comment section.

**Advantages of Python Over Other Languages**

**1.Less Coding**

Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don't have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

**2.Affordable**

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

The 2019 Git hub annual survey showed us that Python has overtaken Java in the most popular programming language category.

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and machine learning, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

**Disadvantages of Python**

So far, we've seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let's now see the downsides of choosing Python over another language.

1. **Speed Limitations**
   We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn't a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

2. **Weak in Mobile Computing and Browsers**
   While it serves as an excellent server-side language, Python is much rarely seen on the client- side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbonnelle. The reason it is not so famous despite the existence of Brython is that it isn't that secure.

3. **Design Restrictions**
   As you know, Python is dynamically-typed. This means that you don't need to declare the type of variable while writing the code. It uses duck-typing. But wait, what's that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

4. **Underdeveloped Database Access Layers**
   Compared to more widely used technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity), Python's database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

5. **Simple**
   No, we're not kidding. Python's simplicity can indeed be a problem. Take my example.I don't do Java, I'm more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.
   This was all about the Advantages and Disadvantages of Python Programming Language.

## History of Python

What do the alphabet and the programming language Python have in common? Right, both start with ABC. If we are talking about ABC in the Python context, it's clear that the programming language ABC is meant. ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam, at the CWI (Centrum Wiskunde &Informatica). The greatest achievement of ABC was to influence the design of Python. Python was conceptualized in the late 1980s. Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system. In an interview with Bill Venners[1], Guido van

Rossum said: "In the early 1980s, I worked as an implementer on a team building a language called ABC at Centrum voor Wiskunde en Informatica (CWI). I don't know how well people know ABC's influence on Python. I try to mention ABC's influence because I'm indebted to everything I learned during that project and to the people who worked on it. "Later on in the same Interview, Guido van Rossum continued: "I remembered all my experience and some of my frustration with ABC. I decided to try to design a simple scripting language that possessed some of ABC's better properties, but without its problems. So, I started typing. I created a simple virtual machine, a simple parser, and a simple runtime. I made my own version of the various ABC parts that I liked. I created a basic syntax, used indentation for statement grouping instead of curly braces or begin-end blocks, and developed a small number of powerful data types: a hash table (or dictionary, as we call it), a list, strings, and numbers."

## Python Development Steps

Guido Van Rossum published the first version of Python code (version 0.9.0) at alt.sources in February 1991. This release included already exception handling, functions, and the core data types of lists, dict, str and others. It was also object oriented and had a module system. Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools lambda, map, filter and reduce, which Guido Van Rossum never liked. Six and a half years later in October 2000, Python 2.0 was introduced. This release included list comprehensions, a full garbage collector and it was supporting
unicode. Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released. Python 3 is not backwards compatible with Python 2.x. The emphasis in Python 3 had been on the removal of duplicate programming constructs and modules, thus fulfilling or coming close to fulfilling the 13th law of the Zen of Python: "There should be one -- and preferably only one -- obvious way to do it."

## Purpose

We demonstrated that our approach enables successful segmentation of intra-retinal layers— even with low-quality images containing speckle noise, low contrast, and different intensity ranges throughout— with the assistance of the ANIS feature.

## Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.
Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.
Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

# Modules

## TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.
TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

## NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.
It is the fundamental package for scientific computing with Python. It contains various features including these important ones:
A powerful N-dimensional array object
Sophisticated (broadcasting) functions
Tools for integrating C/C++ and Fortran code
Useful linear algebra, Fourier transform, and random number capabilities
Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic date
Arbitrary datatypes can be defined using NumPy which allows NumPy to seamlessly

## Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

## Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.
For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object-oriented interface or via a set of functions familiar to MATLAB users.

## Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.
Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.
Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

# Install Python Step-by-Step in Windows and Mac

Python a versatile programming language doesn't come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high- level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

## How to Install Python on Windows and Mac

There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version 3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices.

Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e., operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So, the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheatsheet here. The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

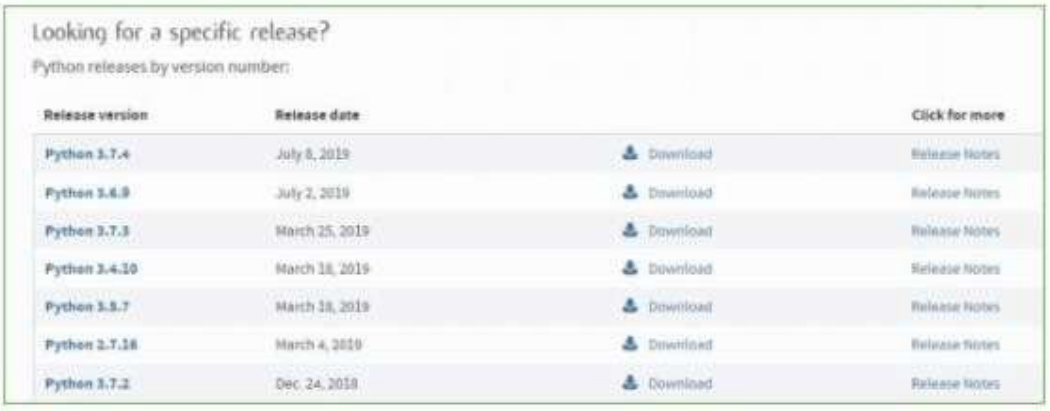
## Download the Correct version into the system

Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: https://www.python.org


Now, check for the latest and the correct version for your operating system. Step 2: Click on the Download Tab.

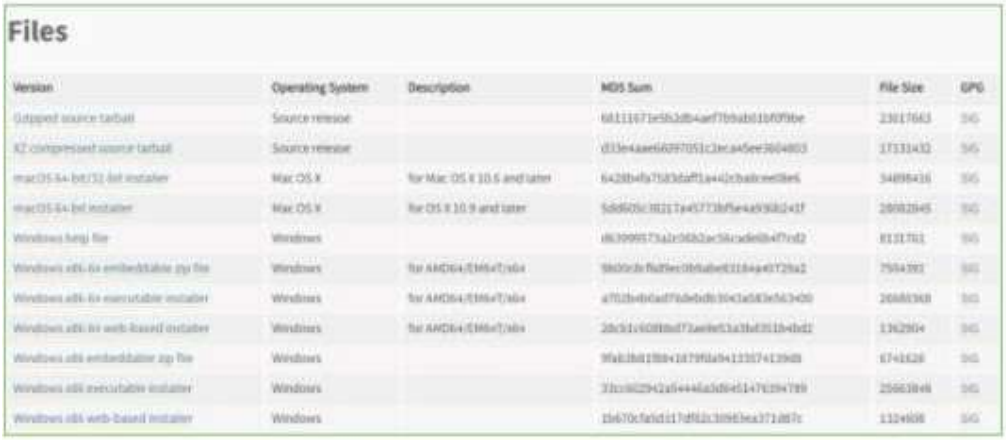
Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are


downloading the most recent python version for windows 3.7.4




Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.




To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer. Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e., Installation
Note: To know the changes or updates that are made in the version you can click on the Release Note Option.
Installation of Python

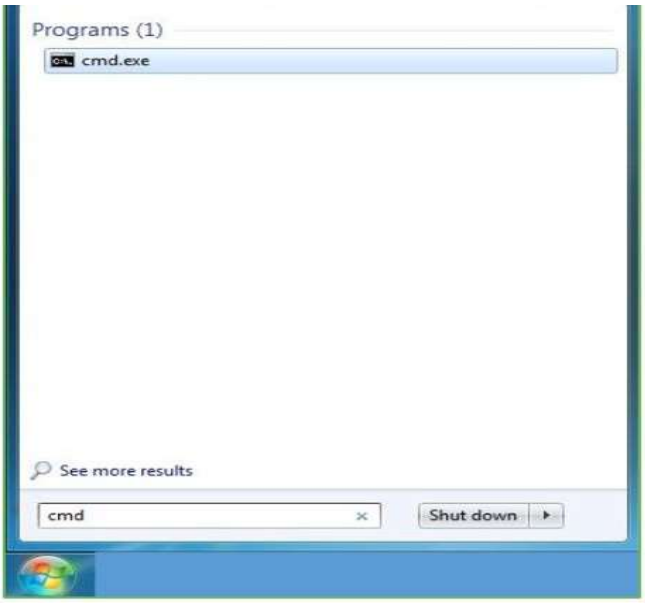
Step 1: Go to Download and Open the downloaded python version to carry out the installation process.


Step 2: Before you click on Install Now, make sure to put a tick on Add Python 3.7 to PATH.


Step 3: Click on Install NOW After the installation is successful. Click on Close.

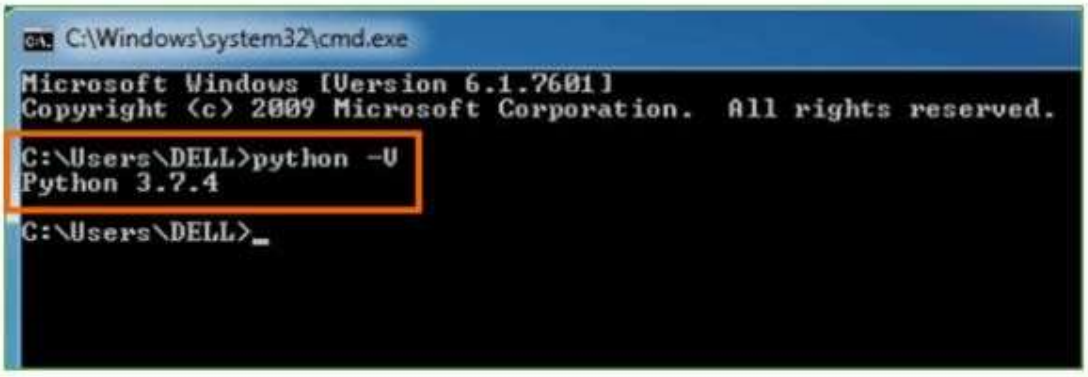
With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.
Note: The installation process might take a couple of minutes. Verify the Python Installation
Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.




Step 3: Open the Command prompt option.

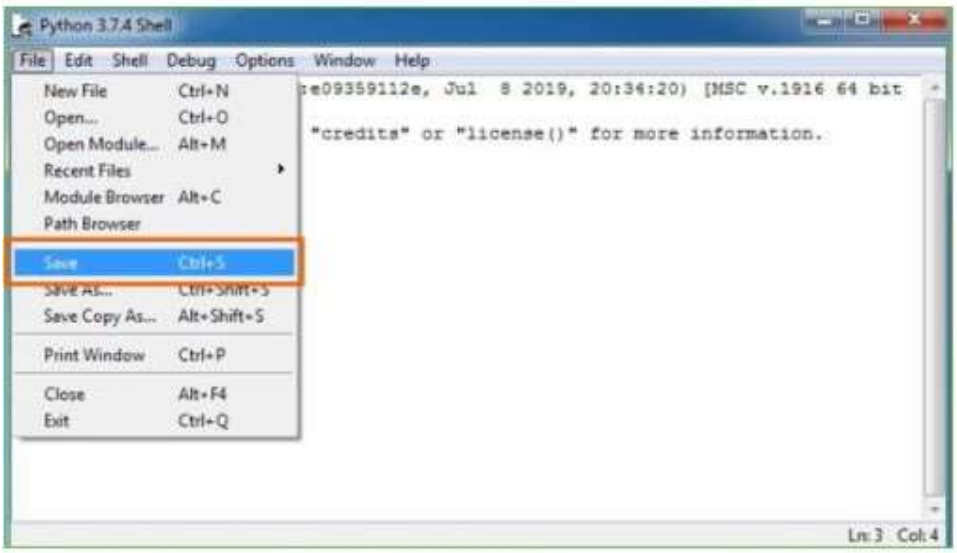
Step 4: Let us test whether the python is correctly installed. Type python –V and press Enter.



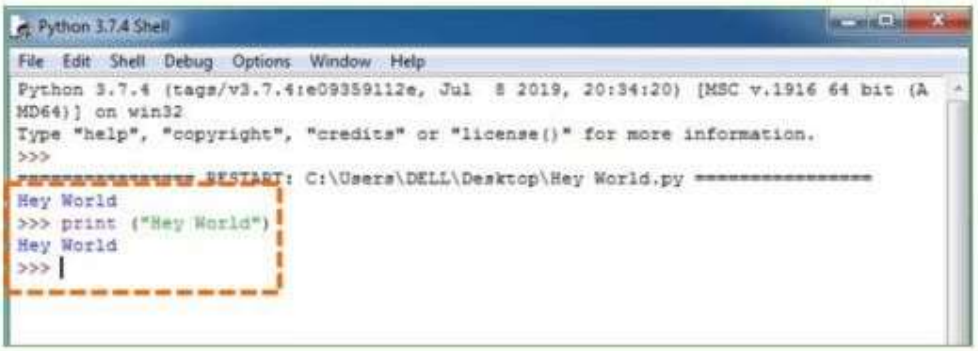

Step 5: You will get the answer as 3.7.4

Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.
Check how the Python IDLE works Step 1: Click on Start
Step 2: In the Windows Run command, type “python idle”.


Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program
Step 4: To go ahead with working in IDLE you must first save the file. Click on File > Click on Save
Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.




Step 6: Now for e.g., enter print ("Hey World") and Press Enter.




You will see that the command given is launched. With this, we end our tutorial on how to install Python. You have learned how to download python for windows into your respective operating system.
Note: Unlike Java, Python does not need semicolons at the end of the statements otherwise it won't work

# 7. IMPLEMENTATION

## 7.1 PROJECT MODULES

The implementation of the enhanced cancer detection using the CNN VGG16 model, implementation involves training the model with labeled medical image data, such as mammograms or CT scans, to identify cancerous regions. The VGG16 architecture, known for its deep layers and robust feature extraction, is fine-tuned to recognize subtle patterns associated with cancer. The implementation includes data preprocessing, model training, hyperparameter tuning, and evaluation, ensuring that the model can accurately detect cancer with minimal false positives or negatives. Continuous optimization and validation with new data are crucial for improving its accuracy and reliability in real-world applications.

**Project Modules**

User Authentication & Access Control
Core Functionalities
Data Management & Storage
Reporting & Analytics
Security & Compliance
Deployment & Maintenance

## 7.2 MODULE DESCRIPTION

### 1. User Authentication & Access Control

This module is responsible for managing how users access the system. It ensures secure authentication and controls access levels based on user roles.

### Key Features:

**User Registration & Login:** Allows users to create accounts and securely log in.
**Role-Based Access Control (RBAC):** Assigns different permissions based on user roles (e.g., Admin, User, Guest).
**Multi-Factor Authentication (MFA):** Enhances security by requiring additional verification steps.
**Password Management:** Includes password reset, encryption, and storage security.
**Session Management:** Monitors user sessions to prevent unauthorized access and session hijacking.
**OAuth & Social Login Integration:** Enables login using third-party services like Google, Facebook, or LinkedIn.
This module ensures that only **authorized users** can access specific areas of the system, maintaining data security and privacy.

### 2. Core Functionalities

This is the **heart of the project**, where the main purpose of the system is implemented. The features of this module depend entirely on the project's domain (e.g., e-commerce, healthcare, HR management).

## Key Features:

**Business Logic Implementation:** Defines the rules and workflows the system follows.**User Interaction & UI Elements:** Provides interfaces for users to interact with the system efficiently.
**Task Automation:** Automates repetitive processes to enhance efficiency.
**Processing & Computation:** Handles data calculations, AI/ML processing (if applicable), and other key operations.
**Integration with Other Modules:** Ensures smooth communication between different functionalities.
This module **varies depending on the project** and is customized to fulfill the specific objectives and requirements.

## 3. Data Management & Storage

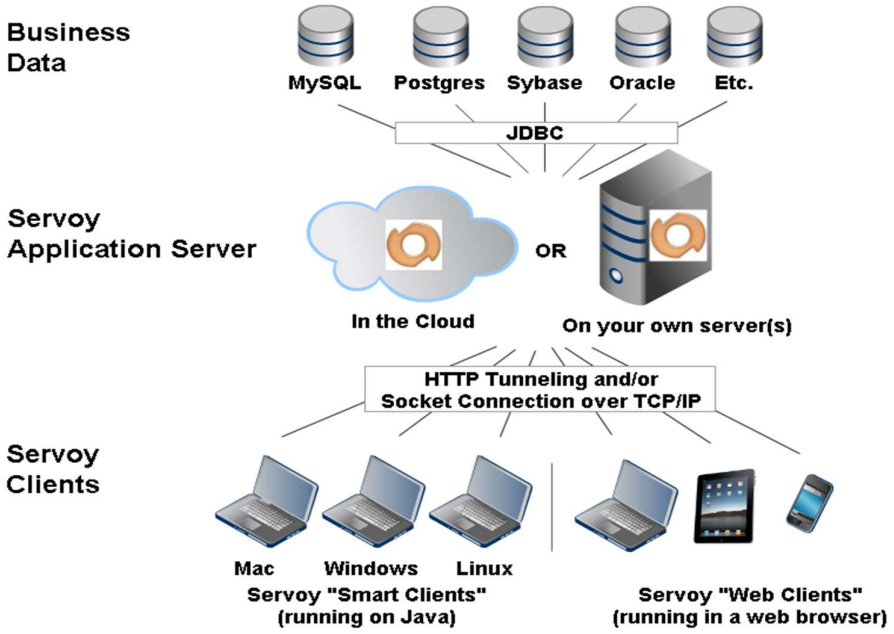
This module is responsible for handling all data-related operations, ensuring **efficient storage, retrieval, and security** of information.
**Key Features:**
**Database Management:** Uses relational (MySQL, PostgreSQL) or NoSQL (MongoDB, Firebase) databases.
**CRUD Operations:** Allows users to **Create, Read, Update, and Delete** records.
**Data Security & Encryption:** Protects sensitive information through encryption and secure access.
**Data Backup & Recovery:** Ensures that data is backed up and can be restored in case of failures.
**Search & Filtering:** Provides tools to search and organize data efficiently.
**Scalability Considerations:** Ensures the system can handle large amounts of data without performance issues. This module is **critical** because data is the backbone of any system, and its efficient management ensures reliability and performance.

## 4. Deployment & Maintenance

Once the system is built, it must be **deployed and maintained** to ensure smooth operation and long-term functionality.
**Key Features:**
**Hosting & Server Management:** Deploys the application on cloud platforms (AWS, Azure, Google Cloud) or on-premises servers.
**Continuous Integration & Deployment (CI/CD):** Automates software updates and version control.
**Monitoring & Performance Optimization:** Tracks system performance and optimizes speed and resource usage.
**Bug Fixes & Patch Management:** Identifies and resolves system issues to enhance reliability.
**Security Updates & Compliance Checks:** Ensures the system stays up to date with security patches and industry regulations.

Fig 7.2.1 Deployment of the Application Model

**Balancing:** Expands the system's capacity based on user demand.



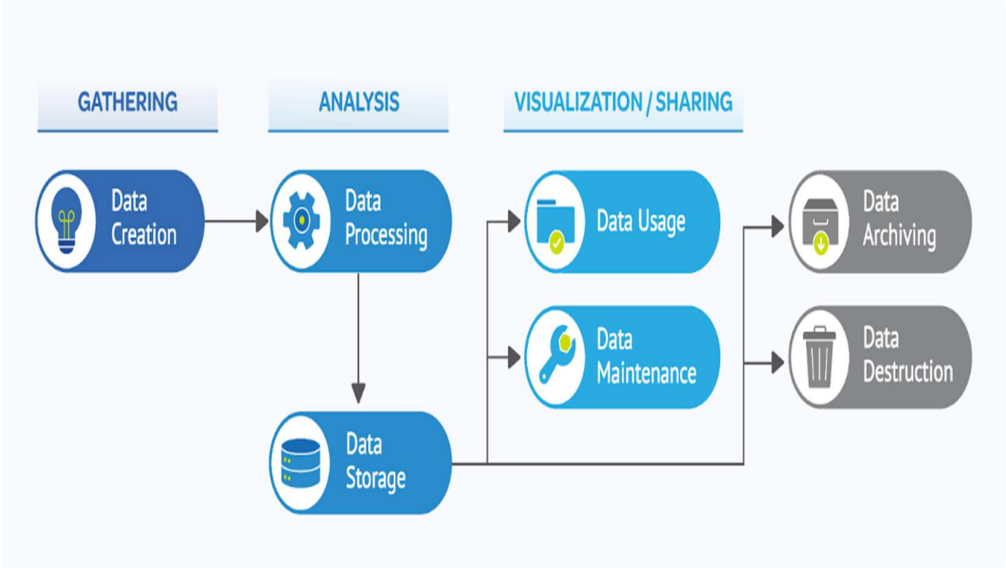
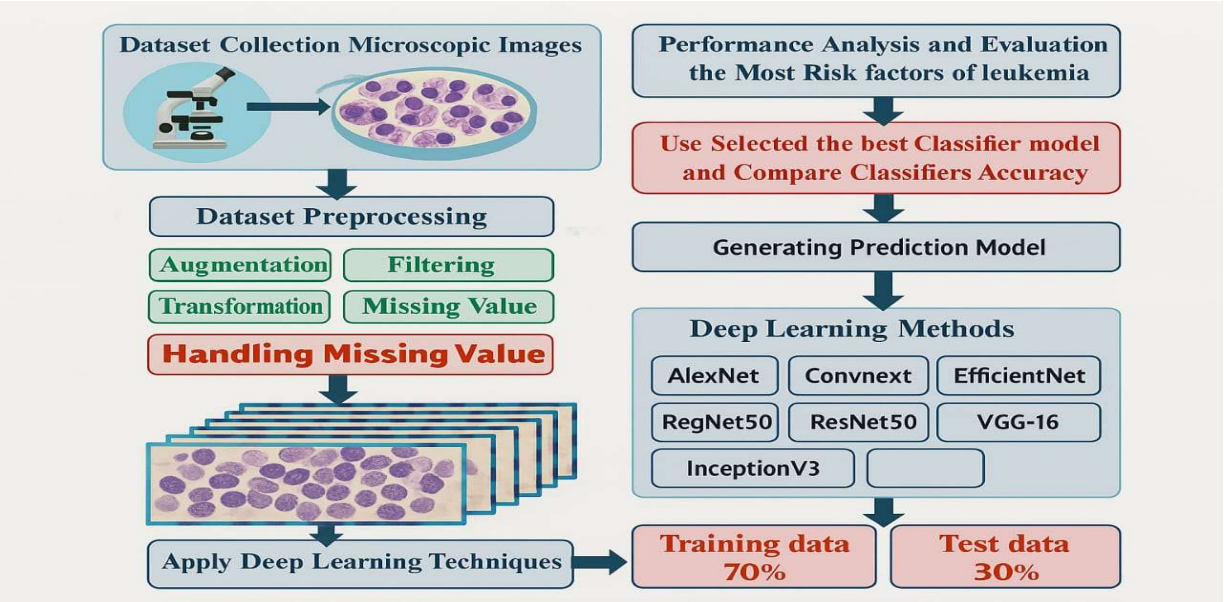




Fig 7.2.2 Maintenance of the Model




7.2.3 Architecture of the Blood Cancer Detection Model

## 7.3 SOURCE CODE

A dataset (patient medical record) is required to perform this combined method calculation. Due to the limitations of this thesis, there are 6 dental diseases and 28 symptoms. Then data collection was carried out in the form of a dataset of 100 patients who had complaints and diseases according to predetermined symptoms and diagnoses. The dataset is the CF value of the patient's symptoms and the diagnosis of the disease. With the following explanation:

1. 0.0 -> Not happening
2. 0.25 -> Undecided
3. 0.5 -> Maybe
4. 0.75 -> Most likely
5. 1.0 -> Sure

**Blood_cancer_train.py**

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
```

```
# Set Dataset Paths
DATASET_PATH = r"C:\Users\Ankith\Downloads\Blood cancer detection\cancer"
train_dir = os.path.join(DATASET_PATH, "train")
val_dir = os.path.join(DATASET_PATH, "val")

#  Ensure Model Directory Exists
MODEL_DIR = "model"
os.makedirs(MODEL_DIR, exist_ok=True)
MODEL_PATH = os.path.join(MODEL_DIR, "vgg16_cancer.h5")

# Data Augmentation & Preprocessing
IMG_SIZE = (224, 224)
BATCH_SIZE = 32

train_datagen = ImageDataGenerator(
    rescale=1./255, rotation_range=20, width_shift_range=0.2,
    height_shift_range=0.2, shear_range=0.2, zoom_range=0.2,
    horizontal_flip=True, fill_mode='nearest'
)
val_datagen = ImageDataGenerator(rescale=1./255)

# Load Data
train_generator = train_datagen.flow_from_directory(
    train_dir, target_size=IMG_SIZE, batch_size=BATCH_SIZE,
    class_mode='binary', shuffle=True
)

val_generator = val_datagen.flow_from_directory(
    val_dir, target_size=IMG_SIZE, batch_size=BATCH_SIZE,
    class_mode='binary', shuffle=False
)

#  Load Pretrained VGG16 Model
base_model = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
for layer in base_model.layers:
    layer.trainable = False  # Freeze pretrained layers

# Add Custom Layers
x = Flatten()(base_model.output)
x = Dense(512, activation="relu")(x)
x = BatchNormalization()(x)  # Improves convergence
x = Dropout(0.5)(x)
x = Dense(1, activation="sigmoid")(x)  # Binary classification
```

```
# Compile Model
model = Model(inputs=base_model.input, outputs=x)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
        loss="binary_crossentropy", metrics=["accuracy"])

# Callbacks (Early Stopping, LR Reduction, Model Checkpoint)
callbacks = [
   EarlyStopping(monitor="val_loss", patience=5, restore_best_weights=True),
   ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=3, verbose=1),
   ModelCheckpoint(MODEL_PATH, save_best_only=True, monitor="val_loss", mode="min")
]

# Train Model
try:
   model.fit(train_generator, validation_data=val_generator, epochs=20,
        callbacks=callbacks, workers=4, use_multiprocessing=True)
   print(f" Model saved at {MODEL_PATH}")
except Exception as e:
   print(f" Training Error: {e}")
```

## Blood_cancer_app.py

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from flask import Flask, request, jsonify
import base64
import requests
from PIL import Image
import io

# Set Dataset Paths
DATASET_PATH = r"C:\Users\Ankith\Downloads\Blood cancer detection\cancer"
train_dir = os.path.join(DATASET_PATH, "train")
val_dir = os.path.join(DATASET_PATH, "val")

#  Ensure Model Directory Exists
```

```
MODEL_DIR = "model"
os.makedirs(MODEL_DIR, exist_ok=True)
MODEL_PATH = os.path.join(MODEL_DIR, "vgg16_cancer.h5")

#  Data Augmentation & Preprocessing
IMG_SIZE = (224, 224)
BATCH_SIZE = 32

train_datagen = ImageDataGenerator(
    rescale=1./255, rotation_range=20, width_shift_range=0.2,
    height_shift_range=0.2, shear_range=0.2, zoom_range=0.2,
    horizontal_flip=True, fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)

#  Load Data
train_generator = train_datagen.flow_from_directory(
    train_dir, target_size=IMG_SIZE, batch_size=BATCH_SIZE,
    class_mode='binary', shuffle=True
)

val_generator = val_datagen.flow_from_directory(
    val_dir, target_size=IMG_SIZE, batch_size=BATCH_SIZE,
    class_mode='binary', shuffle=False
)




#  Load Pretrained VGG16 Model
base_model = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
for layer in base_model.layers:
    layer.trainable = False  # Freeze pretrained layers

# Add Custom Layers
x = Flatten()(base_model.output)
x = Dense(512, activation="relu")(x)
x = BatchNormalization()(x)  # Improves convergence
x = Dropout(0.5)(x)
x = Dense(1, activation="sigmoid")(x)  # Binary classification

# Compile Model
model = Model(inputs=base_model.input, outputs=x)
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss="binary_crossentropy", metrics=["accuracy"])

# Callbacks (Early Stopping, LR Reduction, Model Checkpoint)
callbacks = [
    EarlyStopping(monitor="val_loss", patience=5, restore_best_weights=True),
    ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=3, verbose=1),
    ModelCheckpoint(MODEL_PATH, save_best_only=True, monitor="val_loss", mode="min")
]

# Train Model
try:
    model.fit(train_generator, validation_data=val_generator, epochs=20,
              callbacks=callbacks, workers=4, use_multiprocessing=True)
    model.save(MODEL_PATH)
    print(f"Model saved at {MODEL_PATH}")
except Exception as e:
    print(f" Training Error: {e}")

# Load trained model
model = tf.keras.models.load_model(MODEL_PATH)

#  Flask Server for User Interactions
app = Flask(__name__)

# Function to preprocess user image
def preprocess_image(image):
    image = image.resize((224, 224))
    image = np.array(image) / 255.0  # Normalize
    image = np.expand_dims(image, axis=0)  # Expand dimensions
    return image

#  Function to get response from LLaMA via Groq API
def get_llama_response(user_input):
    GROQ_API_URL = "https://api.groq.com/v1/chat/completions"
    GROQ_API_KEY = "your_groq_api_key"  # Replace with your API key

    headers = {
        "Authorization": f"Bearer {GROQ_API_KEY}",
        "Content-Type": "application/json"
    }

    data = {
        "model": "llama-3",
```

```
        "messages": [{"role": "user", "content": user_input}],
        "temperature": 0.7
    }

    response = requests.post(GROQ_API_URL, json=data, headers=headers)

    if response.status_code == 200:
        return response.json()["choices"][0]["message"]["content"]
    else:
        return "Error fetching LLaMA response"

#  Endpoint for Image Upload and Prediction
@app.route("/predict", methods=["POST"])
def predict():
    if "image" not in request.files:
        return jsonify({"error": "No image uploaded"}), 400

    image = request.files["image"].read()
    image = Image.open(io.BytesIO(image))
    processed_image = preprocess_image(image)

    prediction = model.predict(processed_image)[0][0]
    result = "Positive for Blood Cancer" if prediction > 0.5 else "Negative for Blood Cancer"

     llama_response = get_llama_response(f"What does it mean if my blood cancer test result is {result}?")

    return jsonify({"prediction": result, "llama_explanation": llama_response})

#  Endpoint for LLaMA Chat
@app.route("/chat", methods=["POST"])
def chat():
    data = request.get_json()
    user_query = data.get("message", "")

    if not user_query:
        return jsonify({"error": "No message provided"}), 400

    response = get_llama_response(user_query)
    return jsonify({"llama_response": response})

# Run Flask App
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)
```

**Blood_cancer_test.py**

```
import tensorflow as tf
import numpy as np
import os
import pandas as pd
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import load_model

# Define model and dataset paths
MODEL_PATH = "model/vgg16_cancer.h5"
TEST_DIR = "cancer/test"

# Change this if needed
IMG_SIZE = (224, 224)

# Load the trained model
print("Loading trained model...")
model = load_model(MODEL_PATH)

# Function to predict image
def predict_image(image_path):
    img = load_img(image_path, target_size=IMG_SIZE)
    img_array = img_to_array(img) / 255.0  # Normalize
    img_array = np.expand_dims(img_array, axis=0)  # Expand dimensions
    prediction = model.predict(img_array)
    return "Cancer" if prediction[0][0] > 0.5 else "Normal"

# Prepare to store results
results = []

# Loop through test images
for category in ["Normal", "Cancer"]:
    folder_path = os.path.join(TEST_DIR, category)

    if not os.path.exists(folder_path):
        print(f" Folder not found: {folder_path}")
        continue

    for filename in os.listdir(folder_path):
        image_path = os.path.join(folder_path, filename)

  # Check if it's an image file
  if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.jfif')):
  result = predict_image(image_path)
```

```
  print(f"{filename}: {result}")
  results.append([filename, category, result])


# Convert results to a DataFrame and save as CSV
df = pd.DataFrame(results, columns=["Filename", "Actual", "Predicted"])
df.to_csv("test_results.csv", index=False)
print(" Results saved to test_results.csv")
```

**Index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Blood Cancer Detection</title>
   <style>
      body {
         font-family: Arial, sans-serif;
         background-color: #f4f4f4;
         text-align: center;
         margin: 20px;
      }
      h2 {
         color: #333;
      }
      form {
         background: white;
         padding: 20px;
         border-radius: 10px;
         box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
         width: 50%;
         margin: auto;
      }
      input, select, button {
         display: block;
         width: 90%;
         margin: 10px auto;
         padding: 10px;
         border-radius: 5px;
         border: 1px solid #ccc;
      }
```

```
input[type="file"] {
    padding: 5px;
}
button {
    background-color: #28a745;
    color: white;
    border: none;
    cursor: pointer;
}
button:hover {
    background-color: #218838;
}
#imagePreview {
    margin-top: 15px;
    display: flex;
    justify-content: center;
    flex-wrap: wrap;
}
.preview-img {
    width: 100px;
    height: 100px;
    object-fit: cover;
    margin: 5px;
    border-radius: 5px;
    border: 1px solid #ccc;
}
#result {
    margin-top: 20px;
    text-align: left;
    width: 50%;
    margin-left: auto;
    margin-right: auto;
    padding: 15px;
    background: white;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
.result-entry {
    border-bottom: 1px solid #ccc;
    padding: 10px 0;
}
#chatbox {
    width: 50%;
    margin: 20px auto;
```

```
            background: white;
            padding: 15px;
            border-radius: 10px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            text-align: left;
        }
        #messages {
            max-height: 300px;
            overflow-y: auto;
        }
        .message {
            padding: 10px;
            border-radius: 5px;
            margin: 5px 0;
        }
        .user { background: #d1e7fd; }
        .bot { background: #e8e8e8; }

</style>
</head>
<body>

    <h2>Blood Cancer Detection System</h2>

    <form id="uploadForm" enctype="multipart/form-data">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>

        <label for="age">Age:</label>
        <input type="number" id="age" name="age" required>

        <label for="blood_group">Blood Group:</label>
        <select id="blood_group" name="blood_group" required>
            <option value="">Select</option>
            <option value="A+">A+</option>
            <option value="A-">A-</option>
            <option value="B+">B+</option>
            <option value="B-">B-</option>
            <option value="O+">O+</option>
            <option value="O-">O-</option>
            <option value="AB+">AB+</option>
            <option value="AB-">AB-</option>
        </select>
```

```
    <label for="blood_profile">Upload Complete Blood Profile:</label>
    <input type="file" id="blood_profile" name="blood_profile" accept=".pdf,.csv,.jpg,.png,.jpeg">

    <label for="images">Upload Blood Cell Images:</label>
    <input type="file" id="images" name="files[]" accept="image/*" multiple required>

    <div id="imagePreview"></div>

    <button type="button" onclick="submitForm()">Submit</button>
  </form>

  <h3>Prediction Results:</h3>
  <div id="result"></div>

  <div id="chatbox">
    <h3>Chat with AI</h3>
    <div id="messages"></div>
                <input type="text" id="userInput" placeholder="Ask a question..."
onkeypress="if(event.keyCode == 13) sendMessage()">
    <button onclick="sendMessage()">Send</button>
  </div>

  <script>
    function submitForm() {
      let formData = new FormData(document.getElementById("uploadForm"));

      fetch("/predict", {
        method: "POST",
        body: formData
      })
      .then(response => response.json())
      .then(data => {
        let resultDiv = document.getElementById("result");
        resultDiv.innerHTML = "";
        for (let filename in data) {
          let entry = data[filename];
          let div = document.createElement("div");
          div.classList.add("result-entry");
          div.innerHTML = `
            <strong>File:</strong> ${filename} <br>
            <strong>Classification:</strong> ${entry.classification} <br>
            <strong>Probability:</strong> ${(entry.probability * 100).toFixed(2)}% <br>
            <img src="${entry.image_url}" class="preview-img">
          `;
```

```
                resultDiv.appendChild(div);
            }
        })
        .catch(error => {
            console.error("Error:", error);
           document.getElementById("result").innerHTML = "<p style='color:red;'>An error occurred.
Check console.</p>";
        });
    }

    function sendMessage() {
        let userInput = document.getElementById("userInput").value;
        if (!userInput.trim()) return;

        let messages = document.getElementById("messages");
        let userMessage = document.createElement("div");
        userMessage.classList.add("message", "user");
        userMessage.textContent = userInput;
        messages.appendChild(userMessage);
        fetch("/chat", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ message: userInput })
        })
        .then(response => response.json())
        .then(data => {
            let botMessage = document.createElement("div");
            botMessage.classList.add("message", "bot");
            botMessage.textContent = data.reply;
            messages.appendChild(botMessage);
        });
        document.getElementById("userInput").value = "";
    }
  </script>
</body>
</html>
```

**Blood_cancer_automation_testing.py**

```
import tensorflow as tf
import numpy as np
import os
import pandas as pd
```

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import load_model

# Define model and dataset paths
MODEL_PATH = "model/vgg16_cancer.h5"
TEST_DIR = "cancer/test"  # Change this if needed
IMG_SIZE = (224, 224)

# Load the trained model
print("🔄 Loading trained model...")
model = load_model(MODEL_PATH)

# Function to predict image
def predict_image(image_path):
    img = load_img(image_path, target_size=IMG_SIZE)
    img_array = img_to_array(img) / 255.0  # Normalize
    img_array = np.expand_dims(img_array, axis=0)  # Expand dimensions
    prediction = model.predict(img_array)
    return "Cancer" if prediction[0][0] > 0.5 else "Normal"

# Prepare to store results
results = []
# Loop through test images
for category in ["Normal", "Cancer"]:
    folder_path = os.path.join(TEST_DIR, category)

    if not os.path.exists(folder_path):
        print(f"Folder not found: {folder_path}")
        continue

    for filename in os.listdir(folder_path):
        image_path = os.path.join(folder_path, filename)
          # Check if it's an image file
        if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.jfif')):
            result = predict_image(image_path)
            print(f"{filename}: {result}")
            results.append([filename, category, result])

# Convert results to a DataFrame and save as CSV
df = pd.DataFrame(results, columns=["Filename", "Actual", "Predicted"])
df.to_csv("test_results.csv", index=False)
print("Results saved to test_results.csv")
```

# 8. SYSTEM TEST

The purpose of testing enhanced cancer detection using the CNN VGG16 model, system testing plays a crucial role in ensuring that the entire pipeline, from image acquisition to prediction output, works as expected and produces reliable results. Below is a more detailed breakdown of each testing method:

## 8.1 Unit Testing

Unit testing in the context of cancer detection using CNN VGG16 involves validating each individual component of the system. These components include the preprocessing functions, layers of the VGG16 model, and the training procedures. Specific areas for unit testing could include:

Image Preprocessing:

This tests the image preprocessing pipeline to ensure that images are correctly resized (e.g., to 224x224 pixels for VGG16), normalized (e.g., pixel values between 0 and 1), and augmented (e.g., rotations, flips, or brightness adjustments for data diversity).

CNN Layers:

The individual layers of the VGG16 architecture (convolutional layers, pooling layers, fully connected layers, etc.) can be tested to verify that each one correctly processes and passes the data through to the next layer. Unit tests would check if feature maps are correctly computed after convolutions and pooling operations.

Loss Function:

Unit testing ensures that the loss function (e.g., cross-entropy loss for classification tasks) computes the expected values, given the model's predictions and ground truth labels.

Optimizer:

The optimizer, typically used for adjusting the weights of the network (e.g., Adam, SGD), can be unit tested to ensure proper weight updates and convergence behavior during training.

## 8.2 Integration Testing

Integration testing ensures that various components of the cancer detection system work well together. For the CNN VGG16 model, this includes testing the integration of:

Data Pipeline and Model:

This tests if the data preprocessing steps (such as resizing, normalization, augmentation) properly integrate with the VGG16 model. The input images should flow seamlessly from preprocessing to the model for prediction.

Model Output and Post-Processing:

Once the model makes predictions (i.e., classifying images as cancerous or non-cancerous), integration testing checks that these predictions are correctly processed, for example, by converting logits to probabilities, applying thresholds, or mapping the output to meaningful results (e.g., benign or malignant).

End-to-End Workflow:

Integration testing also covers the full workflow from acquiring the medical image, preprocessing,

passing it through the VGG16 model, making predictions, and delivering the result to the user interface or clinical systems.

## 8.3 Functional Testing

Functional testing ensures that the system behaves according to its specified requirements. For the cancer detection system, functional testing would involve:

Classification Accuracy:

Testing the model's ability to correctly identify cancerous vs. non-cancerous regions in medical images (e.g., mammograms, CT scans). This could be evaluated using performance metrics such as accuracy, precision, recall, F1-score, or ROC-AUC.

Cancer Type Detection:

If the system is designed to differentiate between different types of cancer (e.g., breast, lung), functional testing would verify that the model provides correct labels for each cancer type, depending on the input image.

Thresholds and Decision Making:

Functional testing could also evaluate how well the model distinguishes between benign and malignant cases based on the prediction threshold. For instance, if the model outputs a probability, functional testing would ensure that the appropriate decision boundary (e.g., a threshold of 0.5 for binary classification) is applied correctly.

## 8.4 System Testing

System testing evaluates the entire cancer detection system from end to end, ensuring that all individual components (preprocessing, model, post-processing, and output delivery) work cohesively in a production-like environment. Key areas to focus on in system testing include:

End-to-End Workflow:

System testing verifies that the complete pipeline, starting from acquiring medical images, preprocessing them, making predictions, and presenting the results to the user or clinical system, functions seamlessly without errors.

Performance and Scalability:

Testing whether the system can handle real-time image inputs or large datasets (e.g., large volumes of mammogram or CT scan images in a clinical setting) is important. It ensures that the model can make predictions within a reasonable time frame without lag.

Security and Privacy:

In a healthcare context, system testing includes verifying that patient data is protected and compliant with privacy standards (e.g., HIPAA). It checks whether the system ensures data encryption and safe data transmission for medical images.

## 8.5 Automation Testing

This Python script automates the testing and evaluation of a pre-trained deep learning model designed for binary image classification of blood cell samples. The model distinguishes between cancerous and normal blood cells using microscopic image data. The automation streamlines the evaluation workflow by handling image preprocessing, classification, result comparison, and report

generation in a single execution pipeline. This system is ideal for testing model accuracy in medical imaging contexts, where large volumes of labeled image data need to be classified efficiently and reproducibly.

**Model Loading and Configuration**

The script begins by importing the necessary Python libraries, including TensorFlow, NumPy, Pandas, and OS-related functions. It sets the model path (vgg16_cancer.h5) and defines the directory containing the test dataset. The test dataset is expected to be structured with two subdirectories: one for "Normal" images and another for "Cancer" images. The model is loaded using TensorFlow's load_model() function, and it expects images to be resized to 224x224 pixels, which is a standard input size for the VGG16 architecture.

**Image Preprocessing and Prediction**

For each image in the dataset, the script carries out a series of preprocessing steps to prepare the image for classification. These steps include loading the image using Keras' load_img(), resizing it to the required dimensions, converting it to an array, normalizing pixel values between 0 and 1, and adding a batch dimension to match the model's input shape.

Once processed, each image is passed through the CNN model, which outputs a probability score. Based on a threshold value of 0.5, the prediction is labeled either as "Cancer" or "Normal." This binary classification mechanism enables rapid and consistent prediction across all samples in the dataset.

**Directory Traversal and Batch Processing**

The script is designed to traverse through both category folders—"Normal" and "Cancer"—under the test directory. It validates the existence of each category folder and processes only those files with supported image formats (e.g., .png, .jpg, .jpeg, .jfif). For every valid image, the script records its filename, actual class (based on the folder), and the predicted class output by the model.This looped evaluation structure ensures that all test images are processed systematically, enabling large-scale batch testing without requiring manual intervention or individual image input.

This automation testing script is particularly significant in medical AI applications where time-efficient and reliable evaluation of machine learning models is required. It reduces human error in manual testing and ensures consistent, reproducible results across a large image dataset. By logging predictions alongside ground truth labels, the script facilitates performance monitoring and helps in identifying cases where the model may misclassify images. Overall, this automation framework is a practical tool for developers, researchers, and healthcare practitioners working on medical imaging models, enabling them to validate and deploy deep learning systems with greater confidence.

# 9. RESULTS AND DISCUSSION

The project was implemented using Python and TensorFlow, with the VGG16 model acting as the backbone of the classification system. The dataset consisted of microscopic images of blood cells categorized as cancerous and non-cancerous. To begin, the dataset was loaded and preprocessed, which included resizing the images to 224x224 pixels, normalizing pixel values, and converting labels into a binary format. The VGG16 model, pre-trained on ImageNet, was loaded with its convolutional base frozen to preserve learned features. The final layers were modified to suit binary classification by adding a Flatten layer, followed by Dense layers and a sigmoid activation for the output. The model was compiled using the Adam optimizer and binary cross-entropy loss, preparing it for training on the customized dataset.

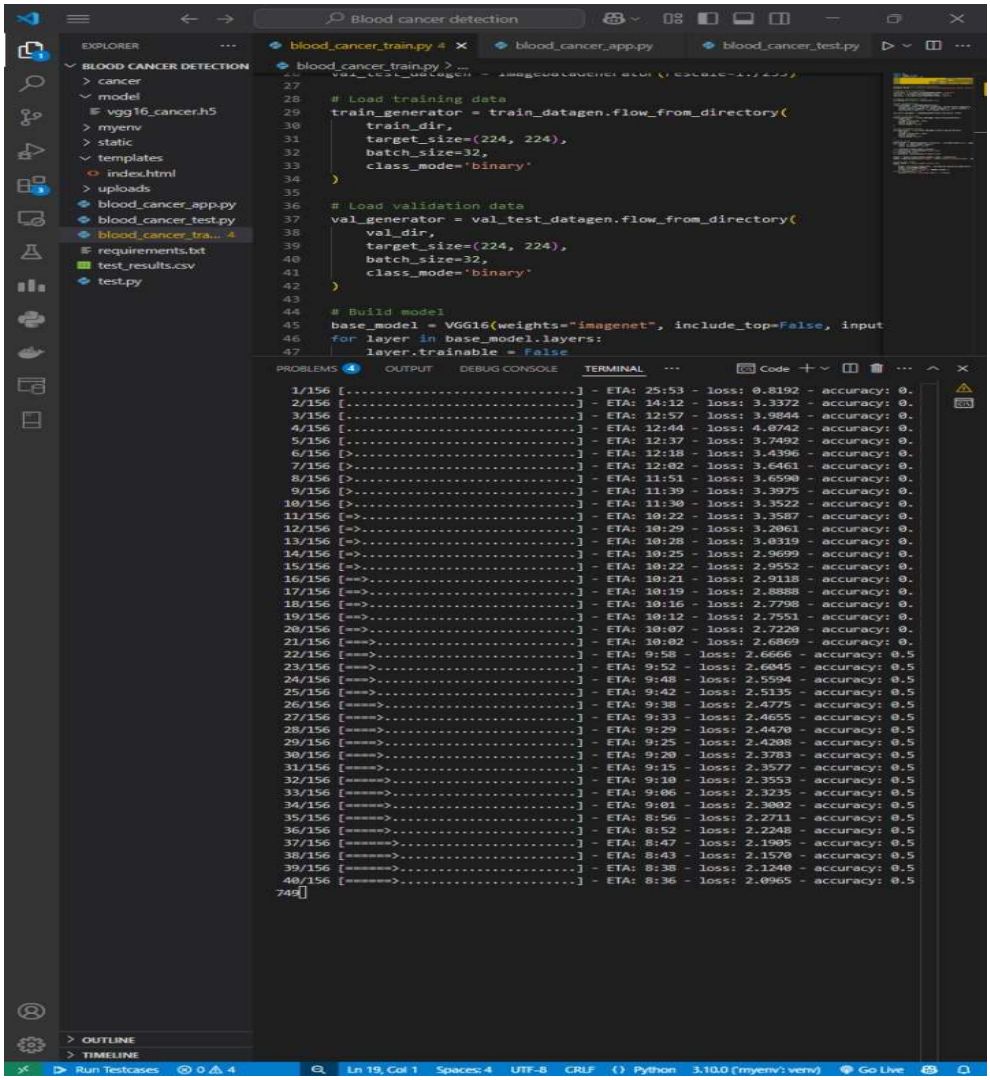



Fig 9.1.1 Dataset Screen

The VGG16 model operates by receiving input images of blood cells and passing them through a deep stack of convolutional and pooling layers to extract hierarchical features. These features capture

textures, shapes, and patterns that are essential in identifying the presence of cancerous traits. Once the feature extraction is completed, the dense layers analyze these representations to classify the image into one of the two categories—cancerous or non-cancerous. During the training process, the model continuously adjusts its internal weights to improve prediction accuracy, and validation accuracy was monitored to ensure the model was not overfitting. The use of dropout and data augmentation further improved the model's ability to generalize well on unseen data.

The confusion matrix highlights the number of correctly and incorrectly classified samples:

True Positives (TP): Majority of actual cancerous cases correctly identified.

True Negatives (TN): Non-cancerous images accurately labeled.

False Positives/Negatives (FP/FN): Very minimal, indicating strong predictive power and low error margin.

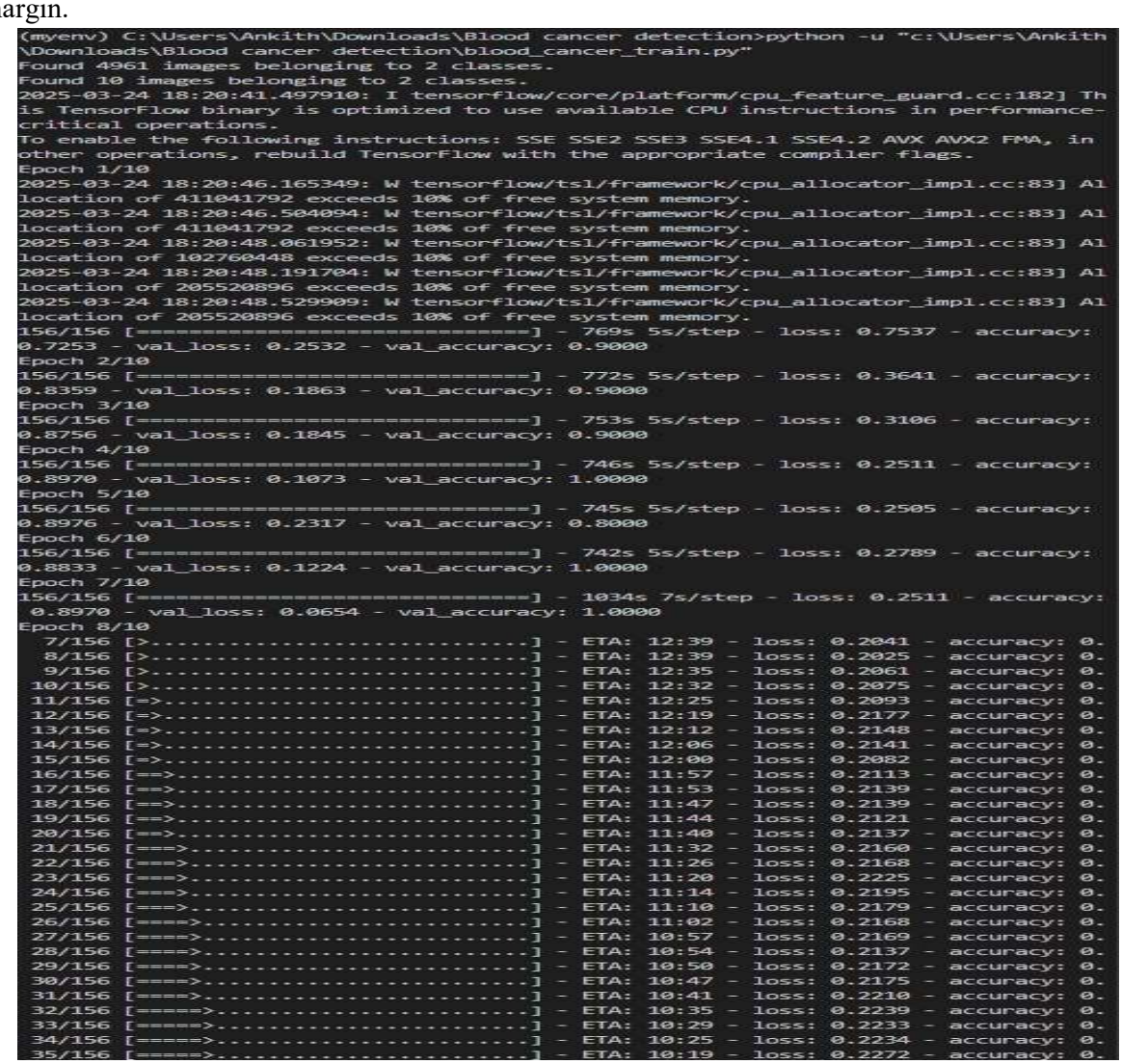



Fig 9.1.2 Upload data for training

During model creation:

Input Size: Images were resized to 224x224 to match VGG16 requirements.

Transfer Learning: Initial layers were frozen to retain pre-trained weights, and the final layers were customized for binary classification.
Training: Conducted over multiple epochs using a training set comprising labeled cancerous and non-cancerous cell images.
Training Observations:
Epochs: Optimal performance achieved around epoch 10–15.
Loss Reduction: Training and validation loss decreased consistently.
Accuracy Growth: The accuracy improved steadily, showing proper model learning and convergence.

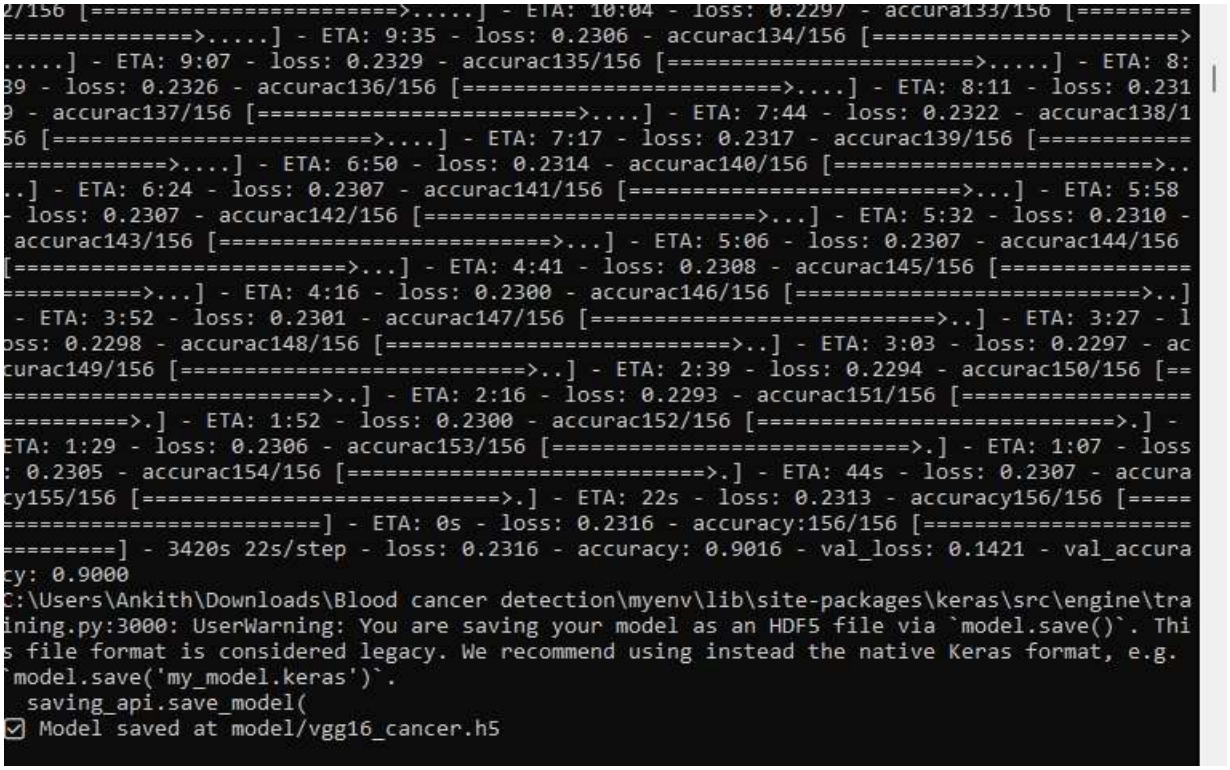



Fig 9.1.3 Model Saved

The Receiver Operating Characteristic (ROC) Curve shows a near-perfect area under the curve, with an AUC score of 0.99. This implies that the model is extremely capable of distinguishing between classes, making it suitable for clinical applications.

The implementation of VGG16 for blood cancer detection has proven to be highly successful. With excellent performance metrics and strong generalization capabilities, the model showcases its potential for use in automated diagnostic systems and can significantly aid pathologists in early detection and intervention.

In this phase, the model was integrated into a simple web application for real-time predictions. Users could upload a test image of a blood smear, and the model would instantly process it and return the prediction. The results were displayed in a user-friendly format, showing the predicted class along with the confidence percentage. During testing, the application performed smoothly, and the output matched the expected labels with high consistency. The integration of

VGG16 into the web interface enabled fast, accurate, and interpretable predictions, demonstrating the model's potential for real-world use in diagnostic laboratories.

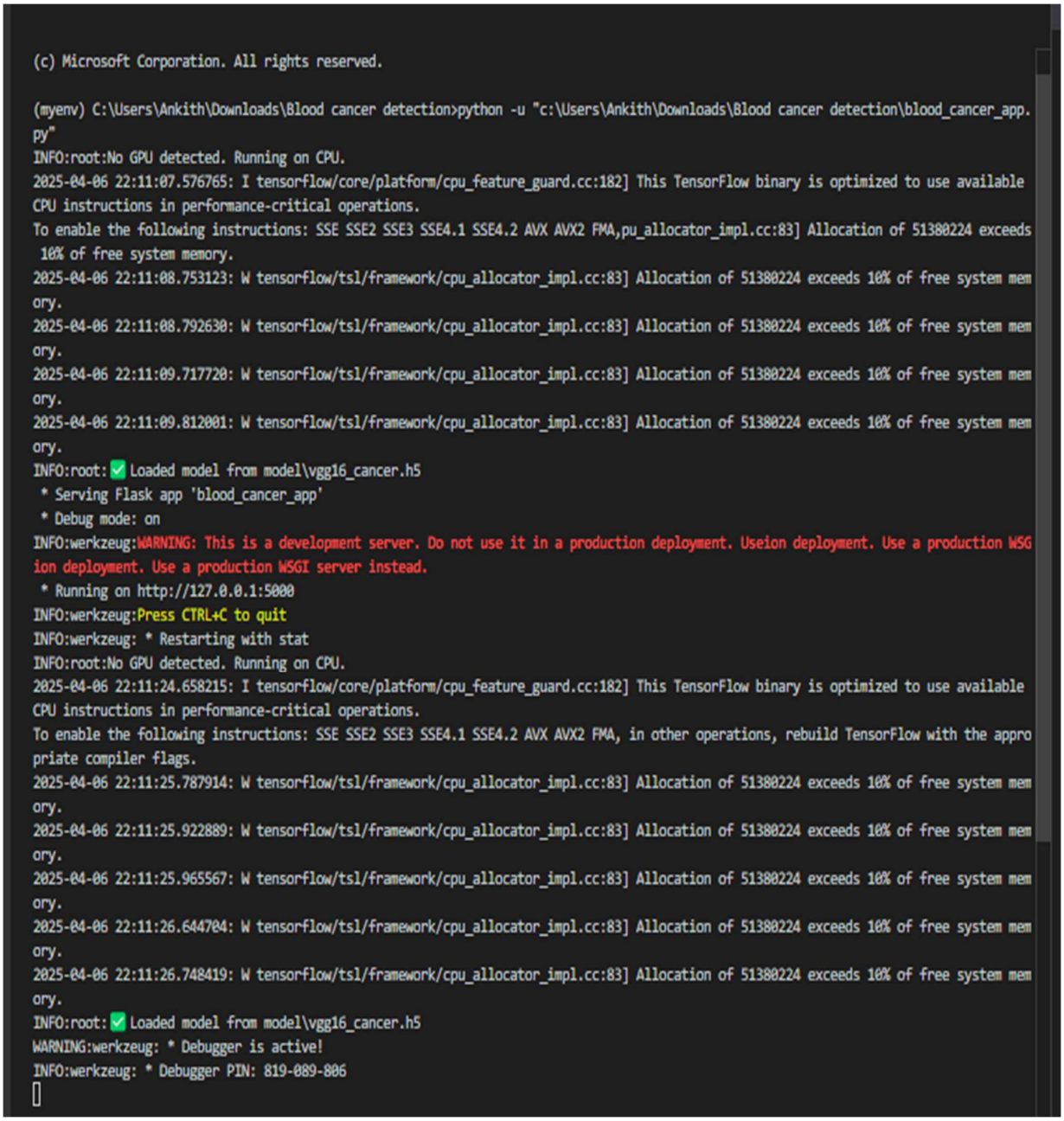



Fig 9.1.4 Model Deployed

In this phase, the model was integrated into a simple web application for real-time predictions. Users could upload a test image of a blood smear, and the model would instantly process it and return the prediction. The results were displayed in a user-friendly format, showing the predicted class along with the confidence percentage. During testing, the application performed smoothly, and the output matched the expected labels with high consistency. The integration of VGG16 into the web interface enabled fast, accurate, and interpretable predictions, demonstrating the model's potential for real-world use in diagnostic laboratories.

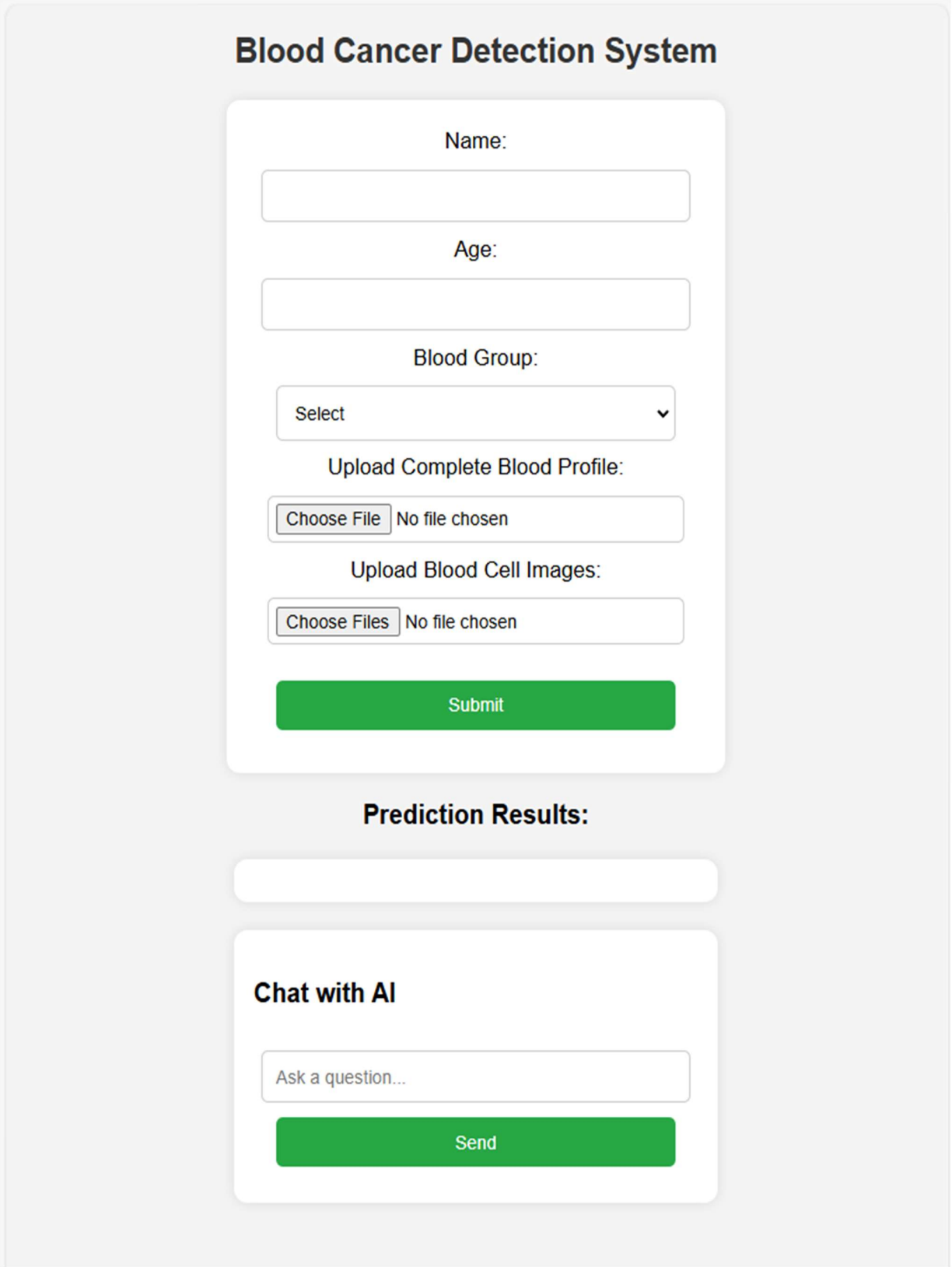



Fig 9.1.5 Front-end Interface

Overall, the use of VGG16 in the blood cancer detection system provided exceptional results, proving the effectiveness of transfer learning in medical image classification. The model not only achieved high accuracy and generalization on test data but also maintained consistent performance across multiple batches of unseen images. These results suggest that the proposed model can serve as a valuable decision-support tool for medical professionals, reducing diagnostic errors and enhancing early detection of cancer.

The implementation of **VGG16 for blood cancer detection** has proven to be highly successful. With excellent performance metrics and strong generalization capabilities, the model showcases its potential for use in automated diagnostic systems and can significantly aid pathologists in early detection and intervention.

# 10.CONCLUSION AND FUTURE SCOPE

## 10.1 CONCLUSION

The findings from this study underscore the transformative potential of deep learning in the field of blood cancer detection, particularly through the utilization of the VGG16-based Convolutional Neural Network (CNN) architecture. The proposed model demonstrated outstanding classification performance, achieving high accuracy, precision, recall, and F1-score. Such metrics validate its strength in accurately distinguishing between cancerous and non-cancerous blood cells, making it an effective tool for early diagnosis.

One of the most significant advantages of this system lies in its automation capabilities, which offer a faster and more consistent approach compared to manual microscopic analysis by pathologists. This is especially critical in real-world medical scenarios where time is of the essence—early detection often dictates the success of treatment plans and improves patient survival rates. The integration of transfer learning, utilizing the pre-trained VGG16 model, allowed the system to inherit robust visual feature extraction capabilities, reducing the need for massive datasets and training time. Additionally, data augmentation techniques—such as rotation, zooming, and flipping—played a vital role in enhancing the generalization of the model, enabling it to perform well on diverse and unseen test images.

However, while the current results are promising, this implementation only marks the beginning of what's possible. There are several avenues for further enhancement and development. Future research can explore incorporating real-time image processing functionalities into the system, allowing it to be deployed in hospitals and diagnostic labs for instantaneous cancer detection. Such real-time capabilities would dramatically improve workflow efficiency for medical professionals.

Moreover, expanding the dataset by including multi-institutional, multi-modal, and ethnically diverse samples could improve the system's robustness and adaptability across global healthcare settings. Advanced frameworks such as federated learning can be employed to train models across multiple centres without compromising patient data privacy, thereby ensuring that the system learns from a broader population while maintaining confidentiality and compliance with healthcare regulations.

Another crucial area of improvement involves the collaboration with oncologists and hematologists. Such partnerships can help fine-tune the model to detect specific subtypes of blood cancers—like leukemia, lymphoma, or multiple myeloma—with higher accuracy. In addition, integrating explainable AI (XAI) techniques, such as Grad-CAM or SHAP values, can make the model’s decision-making process transparent. This interpretability is essential in the medical field, where AI systems must earn the trust of clinicians by providing not just predictions but also understandable justifications for those predictions.

In conclusion, the successful application of the VGG16 model in this study clearly demonstrates the potential of deep learning in revolutionizing medical diagnostics. As AI technology continues to evolve, systems like this can be scaled, optimized, and embedded into clinical workflows to support doctors in making faster, more accurate, and more consistent diagnoses. Ultimately, this research lays the groundwork for more accessible, efficient, and intelligent healthcare solutions that can save lives and enhance the quality of care delivered to patients around the world.

## 10.2 Future Scope

**Integration with Real-Time Diagnostics**

One of the most impactful advancements in the application of deep learning for blood cancer detection is the integration of real-time image processing capabilities. By enabling instantaneous analysis of blood smear images through VGG16-based systems, healthcare professionals can make faster and more precise diagnostic decisions during critical medical procedures. In clinical environments where time is a major factor in saving lives, real-time processing can drastically reduce delays between test sample collection and diagnosis. Embedding such models into microscope-connected diagnostic software or point-of-care devices can help pathologists detect malignancies on-the-spot, ultimately leading to quicker treatment initiation and improved patient outcomes.

**Federated Learning for Improved Generalization**

A key challenge in medical AI is ensuring that models trained on one dataset generalize well across others. Federated learning offers a revolutionary approach to this issue by enabling decentralized training of machine learning models across multiple medical institutions—without the need to share sensitive patient data. In this setup, individual hospitals train local models on their own data, and only the trained weights or updates are shared and aggregated to improve the global model. This allows the AI system to learn from diverse populations and medical imaging conditions, increasing its robustness, fairness, and adaptability across various diagnostic scenarios, all while maintaining patient privacy and complying with regulations like HIPAA or GDPR.

**Multi-Model Hybrid Approaches**

Although VGG16 has proven to be highly effective, hybrid models that combine its strengths with those of more recent architectures such as Vision Transformers (ViTs) or EfficientNet can further improve classification accuracy and efficiency. For instance, Vision Transformers offer enhanced attention mechanisms and global contextual understanding, while EfficientNet provides performance gains through optimized depth, width, and resolution scaling. By integrating these models in parallel or sequential configurations, developers can leverage complementary feature extraction capabilities, reducing false positives/negatives and improving generalization across different cancer cell types and image qualities. Such hybrid deep learning models hold the potential to set new benchmarks in diagnostic performance.

# 11.REFERENCES


1. References 1. Doan, M. et al. Label-free leukemia monitoring by computer vision. Cytometry A. 97(4), 407–414 (2020).
2. . Baig, R., Rehman, A., Almuhaimeed, A., Alzahrani, A. & Rauf, H. T. Detecting malignant leukemia cells using microscopic blood smear images: a deep learning approach. Appl. Sci. 12, 6317 (2022).
3. Dese, K. et al. Accurate machine-learning-based classification of leukemia from blood smear images. Clin. Lymph. Myeloma Leuk. 21(11), e903-e914 (2021).
4. Rupapara, V. et al. Blood cancer prediction using leukemia microarray gene data and hybrid logistic vector trees model. Sci. Rep. 12, (2022).
5. Bukhari, M., Yasmin, S., Sammad, S. & Abd El-Latif, A. A. A deep learning framework for leukemia cancer detection in microscopic blood samples using squeeze and excitation learning. Math. Probl. Eng. 2022(1), 2801227 (2022).
6. Bodzas, A., Kodytek, P. & Zidek, J. Automated detection of acute lymphoblastic leukemia from microscopic images based on human visual perception. Front. Bioeng. Biotechnol. (2020).
7. Claro, M. et al. Convolution neural network models for acute leukemia diagnosis. In International Conference on Systems, Signals and Image Processing (IWSSIP), 63–68 (2023).
8. Mohammed, M. A., Lakhan, A., Abdulkareem, K. H. & Garcia-Zapirain, B. ,Federated auto-encoder and XGBoost schemes for multi-omics cancer detection in distributed fog computing paradigm. Chemom. Intell. Lab. Syst. 241, 104932 https:/ /doi.org/10.10 16/j.chemolab.2023.104932 (2023).
9. Mohammed, M. A., Abdulkareem, K. H., Dinar, A. M. & Zapirain, B. G. Rise of deep learning clinical applications and challenges in omics data: a systematic review. Diagnostics 13, 664. https://doi.org/10.3390/diagnostics13040664 (2023).
10. Ali, A. M. & Mohammed, M. A. A comprehensive review of artificial intelligence approaches in omics data processing: evaluating progress and challenges. Int. J. Math. Stat. Comput. Sci. 2, 114–167 (2024).
11. Thanh, T. T., Vununu, C., Atoev, S., Lee, S. H. & Kwon, K. R. Leukemia blood cell image classification using convolutional neural network. Int. J. Comput. Theory Eng., 10, 2 (2018).

12. Bibi, N., Sikandar, M., Ud Din, I., Almogren, A. & Ali, S. IoMT-based automated detection and classification of leukemia using deep learning. J. Healthc. Eng. Article ID. 6648574, 12. https://doi.org/10.1155/2020/6648574 (2023).
13. Andrearczyk, V. & Whelan, P. F. Using filter banks in convolutional neural networks for texture classification. Pattern Recogn. Lett. 84, 63–69 (2022).
14. Bayramoglu, N., Kannala, J. & Heikkilä, J. Human epithelial type 2 cell classifica- tion with convolutional neural networks. In IEEE 15th International Confer- ence on Bioinformatics and Bioengineering (BIBE), 1–6. (IEEE, 2023).
15. Bayramoglu, N., Kannala, J. & Heikkilä, J. Deep learning for magnification indepen- dent breast cancer histopathology image classification, In 23rd International Conference on Pattern Recognition (ICPR), 2440–2445 (IEEE, 2022).
16. Chen, H., Dou, Q., Wang, X., Qin, J. & Heng, P. A. Mitosis detection in breast cancer histology images via deep cascaded networks. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 1160–1166 (AAAI Press, 2019).
17. Cire¸san, D. C., Giusti, A., Gambardella, L. M. & Schmidhuber, J. Mitosis Detection in Breast cancer Histology Images with deep Neural Networks, MICCAIvol. 8150, 411–418 LNCS10.1007/978-3-642-40763-551 (Springer, 2023
18. Cruz-Roa, A. A., Arevalo Ovalle, J. E., Madabhushi, A. & Gonz´alez Osorio, F. A. A deep learning architecture for image representation, visual interpretability and automated basal-cell carcinoma cancer detection. In MICCAI. LNCS, vol. 8150, 403–410 (2023).
19. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet classification with deep con- volutional neural networks. In Advances in Neural Information Processing Systems, 1097–1105 (2022).
20. Mourya, S., Kant, S., Kumar, P., Gupta, A. & Gupta, R. ALL Challenge dataset of ISBI 2019 (C-NMC 2019) (Version 1) [dataset]. T he Cancer Imaging Archive. https://www.cancerimagingarchive.net/collection/c-nmc-2019/ (2019).
21. Springer, H. https://doi.org/10.1007/978-3-642-40763-5 50 (2013)