

WHITE BLOOD CELLS DETECTION BY USING CONVOLUTIONAL NEURAL NETWORK (CNN)

A Major Project Report submitted

in partial fulfillment for the award of the Degree of

Bachelor of Technology

In

Computer Science and Engineering

by

B. SRIRAM (U20NA003)

S. SABEER KHAN (U20NA030)

S. AKHIL KUMAR (U20NA034)

P. MADHAVAN (U20NA039)

Under the guidance of

Mrs. K.Amutha, M.E. (Assistant Professor)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

BHARATH INSTITUTE OF HIGHER EDUCATION AND RESEARCH

(Deemed to be University Estd u/s 3 of UGC Act, 1956)

CHENNAI 600 073, TAMILNADU, INDIA

April, 2024

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

BONAFIDE CERTIFICATE

This is to Certify that this Innovation Project Report Titled “**WHITE BLOOD CELLS DETECTION BY USING CONVOLUTIONAL NEURAL NETWORK (CNN)**” is the Bonafide Work of B.SRIRAM(U20NA003), S.SABEERKHAN(U20NA030), S.AKHILKUMAR(U20NA034), P.MADHAVAN(U20NA039) Final Year B.Tech. (CSE) who carried out the Major project work under my supervision Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report or dissertation on basis of which a degree or award conferred on an earlier occasion by any other candidate.

PROJECT GUIDE

Mrs. K.Amutha

Assistant Professor

Department of CSE

BIHER

HEAD OF THE DEPARTMENT

Dr.S. Maruthuperumal

Professor & Head

Department of CSE

BIHER

Submitted for the Project Viva-Voce held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

We declare that this Major project report titled **WHITE BLOOD CELLS DETECTION BY USING CONVOLUTIONAL NEURAL NETWORK (CNN)** submitted in partial fulfillment of the degree of **B. Tech in (Computer Science and Engineering)** is a record of original work carried out by us under the supervision of **Mrs.K.Amutha** and has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

B. SRIRAM
(U20NA003)

S. SABEER KHAN
(U20NA030)

S. AKHIL KUMAR
(U20NA034)

P. MADHAVAN
(U20NA039)

Chennai

Date:

ACKNOWLEDGMENTS

We express our sincere thanks to our beloved Honorable Chairman **Dr.S.Jagathrakshakan, M.P.**, for continuous and constant encouragement in all academic activities.

We express our deepest gratitude to our beloved President **Dr. J. Sundeep Aanand** President , and Managing Director **Dr.E.Swetha Sundeep Aanand** Managing Director for providing us the necessary facilities to complete our project.

We take great pleasure in expressing sincere thanks to **Dr.K.VijayaBaskar Raju** Pro Chancellor , **Dr.M.Sundararajan** Vice Chancellor (i/c), **Dr.S.Bhuminathan** Registrar and **Dr.R.Hariprakash** Additional Registrar, **Dr.M.Sundararaj** Dean Academics for moldings our thoughts to complete our project.

We thank our **Dr.S.Neduncheliyan** Dean, School of Computing for his encouragement and the valuable guidance.

We record indebtedness to our Head, **Dr.S.Maruthuperumal** (Head of Department), Department of Computer Science and Engineering for his immense care and encouragement towards us throughout the course of this project.

We also take this opportunity to express a deep sense of gratitude to our Supervisor **Ms.K.Amutha** (Assistant Professor) and our Project Co-ordinator **Dr.T.Veeramani** (Professor) for their cordial support, valuable information and guidance, They helped us in completing this project through various stages.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

B. SRIRAM (U20NA003)

S. SABEER KHAN (U20NA030)

S. AKHIL KUMAR (U20NA034)

P. MADHAVAN (U20NA039)

ABSTRACT

The microscopic inspection of blood smears provides diagnostic information concerning patients' health status. They use a microscope and count the percentage of the occurrence of each type of cell counted within an area of interest in smears. Obviously, this manual counting process is very tedious and slow. In addition, the cell classification and counting accuracy may depend on the capabilities and experiences of the operators. Therefore, the necessity of an automated differential counting system becomes inevitable. In this project, CNN models are used. In order to achieve good performance from deep learning methods, the network needs to be trained with large amounts of data during the training phase. We take the images of the white blood cells for the training phase and train our model on them. With this method we achieved good accuracy than traditional methods. And we can generate the results within seconds also. White blood cells account for only about 1% of your blood, but their impact is big. White blood cells are also called leukocytes. They protect you against illness and disease. Think of white blood cells as your immunity cells. In a sense, they are always at war. They flow through your bloodstream to fight viruses, bacteria, and other foreign invaders that threaten your health. When your body is in distress and a particular area is under attack.

TABLE OF CONTENT

S.no	Title	Page no
	Abstract	v
	List of Tables	viii
	List of Figures	ix
	Abbreviations	x
1	Introduction	1-10
	1.1 Introduction to the project	1
	1.2 Types of white blood cells	4
	1.3 Problem affecting white blood cells	8
2	Literature Survey	11-12
3	Proposed Methodology	13-27
	3.1 Existing system	13
	3.2 Proposed system	14
	3.3 Purpose of the system	16
	3.4 Scope of the system	18
	3.5 Python	23
	3.6 Keras	24
	3.7 Tensorflow	25
4	Experimental Results & Discussion	28-80
	4.1 System Architecture	28
	4.2 System design	30
	4.3 UML Diagram	31

	4.4 Neural networks	38
	4.5 Convolutional layer	43
	4.6 Kernal	46
	4.7 Data Preprocessing	56
	4.8 Jupyter notebook	68
5	Conclusion & Feature work	81-88
	5.1 Feature work	83
	5.2 Summary	87
	Reference	89-90
	Appendix – A	91
	Publications(paper)	92

LIST OF TABLES

Sno	Title	Page no
1	Literature Survey	12
2	Data cleaning	60

LIST OF FIGURES

Fig no	Name of the Figure	Page no
1	White blood cells	15
2	Types of white blood cells	16
3	Tensor flow architecture	38
4	System architecture	40
5	Use case diagram for developer	44
6	Use case diagram for user	44
7	Sequence diagram	45
8	Acitivity diagram	47
9	Convolutional operations	55
10	Kernel operations	57
11	Krenel types	58
12	Pool types	62
13	Fully connected layer	63
14	Model architecture	64
15	Data pre -processing	68
16	Data cleaning	69
17	Data visualization	71
18	Data transformation	73
19	Anaconda prompt	79

LIST OF ABBREVIATIONS

CNN – Convolutional Neural Network
CSF - Colony Stimulating Factor
RNN – Recurrent Neural Network

1.INTRODUCTION

1.1 INTRODUCTION TO PROJECT

White blood cells (WBCs), also called leukocytes or leucocytes, are the cells of the system that are involved in protecting the body against both infectious disease and foreign invaders. All white blood cells are produced and derived from multi potent cells in the bone known as hematopoietic. Leukocytes are found throughout the body, including the blood and lymphatic system. All white blood cells have nuclei which distinguishes them from the other blood, the anucleated red blood cells (RBCs) and platelets. Our blood is made up of red blood cells, white blood cells, platelets, and plasma. White blood cells account for only about 1% of your blood, but their impact is big. White blood cells are also called leukocytes. They protect you against illness and disease. Think of white blood cells as your immunity cells. In a sense, they are always at war. They flow through your bloodstream to fight viruses, bacteria, and other foreign invaders that threaten your health. When your body is in distress and a particular area is under attack, white blood cells rush in to help destroy the harmful substance and prevent illness.

White blood cells are made in the bone marrow. They are stored in your blood and lymph tissues. Because some white blood cells called neutrophils have a short life less than a day, your bone marrow is always making them. The proposed CNN-based approach aims to automate the process of white blood cell classification, leveraging the power of deep learning to enhance accuracy and efficiency. The network is trained on a dataset comprising diverse white blood cell images, allowing it to learn intricate features and patterns associated with different cell types.

The proposed model not only achieves high classification accuracy but also exhibits resilience to variations in cell morphology and staining techniques. This research contributes to the ongoing efforts in leveraging deep learning techniques for medical image analysis, particularly in the field of hematology. The automated classification of white blood cells through CNNs holds promise for improving diagnostic processes, ultimately benefiting healthcare professionals and patients alike. The automation of blood smear analysis through CNN models marks a significant advancement in medical diagnostics. By leveraging deep learning techniques, this project aims to alleviate the painstaking task of manual cell counting, which is not only time-consuming but also prone to human error. With the utilization of

convolutional neural networks (CNNs), trained on a large dataset of white blood cell images, the system achieves enhanced accuracy compared to conventional methods.

White blood cells, comprising only about 1% of the blood volume, play a crucial role in immune defense. Referred to as leukocytes, they serve as the frontline warriors against pathogens, tirelessly patrolling the bloodstream to identify and neutralize foreign invaders. During times of infection or distress, white blood cells mobilize swiftly to the affected sites, mounting a coordinated defense to combat harmful substances and prevent the onset of illness.

The implementation of an automated system for differential white blood cell counting not only streamlines the diagnostic process but also enhances the efficiency of healthcare delivery. By generating results within seconds, this technology facilitates prompt decision-making by healthcare professionals, leading to timely interventions and improved patient outcomes.

Moreover, the integration of CNN models in medical diagnostics signifies a paradigm shift towards precision medicine, where data-driven approaches enable personalized patient care. Through continuous refinement and validation, these AI-powered systems hold the promise of revolutionizing healthcare practices, ensuring accurate diagnoses and tailored treatment strategies for individuals across diverse clinical settings

In essence, the convergence of artificial intelligence and medical science heralds a new era in healthcare, where innovative technologies empower clinicians with enhanced diagnostic capabilities and pave the way for more effective disease management strategies. The ultimate aim is to create an automated system assisting medical professionals in diagnosing diseases based on blood cell morphology. **Data Collection:** Assemble a diverse and well-labeled dataset of white blood cell images, including neutrophils, lymphocytes, monocytes, eosinophils, and basophils. . White blood cells are made in the bone marrow. They are stored in your blood and lymph tissues. Because some white blood cells called neutrophils have a short life less than a day, your bone marrow is always making them. The proposed CNN-based approach aims to automate the process of white blood cell classification, leveraging the power of deep learning to enhance accuracy and efficiency.

White blood cells are made in the bone marrow. They are stored in your blood and lymph tissues. Because some white blood cells called neutrophils have a short lifeless than a day, your bone marrow is always making them. The primary goal of this project is to develop a deep learning model using CNNs to categorize white blood cells into different subtypes. The model will be trained on a labeled dataset of blood cell images to recognize patterns and features indicative of specific white blood cell types. The ultimate aim is to create an automated system assisting medical professionals in diagnosing diseases based on blood cell morphology

.Data Collection: Assemble a diverse and well-labeled dataset of white blood cell images, including neutrophils, lymphocytes, monocytes, eosinophils, and basophils. Data Preprocessing: Normalize and augment the dataset to enhance the model's ability to generalize, involving resizing, cropping, and introducing variations to the images.Design a CNN architecture suitable for image classification. Experiment with various layers, filter sizes, and activation functions to optimize performance. Train the CNN on the prepared dataset using a suitable optimization algorithm.

Monitor the model's performance on validation data to prevent overfitting.Assess the model on a separate test dataset to determine its accuracy, precision, recall, and F1-score for each white blood cell subtype. Refine the model based on evaluation results to improve its performance and generalization capabilities. trained CNN model capable of accurately classifying white blood cells into distinct subtypes. Enhanced efficiency in diagnosing diseases related to abnormalities in white blood cell counts.Reduction in manual effort and potential for human error in traditional blood cell classification methods.

Automating white blood cell classification through deep learning has the potential to revolutionize medical diagnostics. Rapid and accurate identification of white blood cell subtypes can aid in early disease detection and timely intervention, ultimately improving patient outcomes. White blood cells are made in the bone marrow. They are stored in your blood and lymph tissues. Because some white blood cells called neutrophils have a short life less than a day, your bone marrow is always making them. The proposed CNN-based approach aims to automate the process of white blood cell classification, leveraging the power of deep learning to enhance accuracy and efficiency.

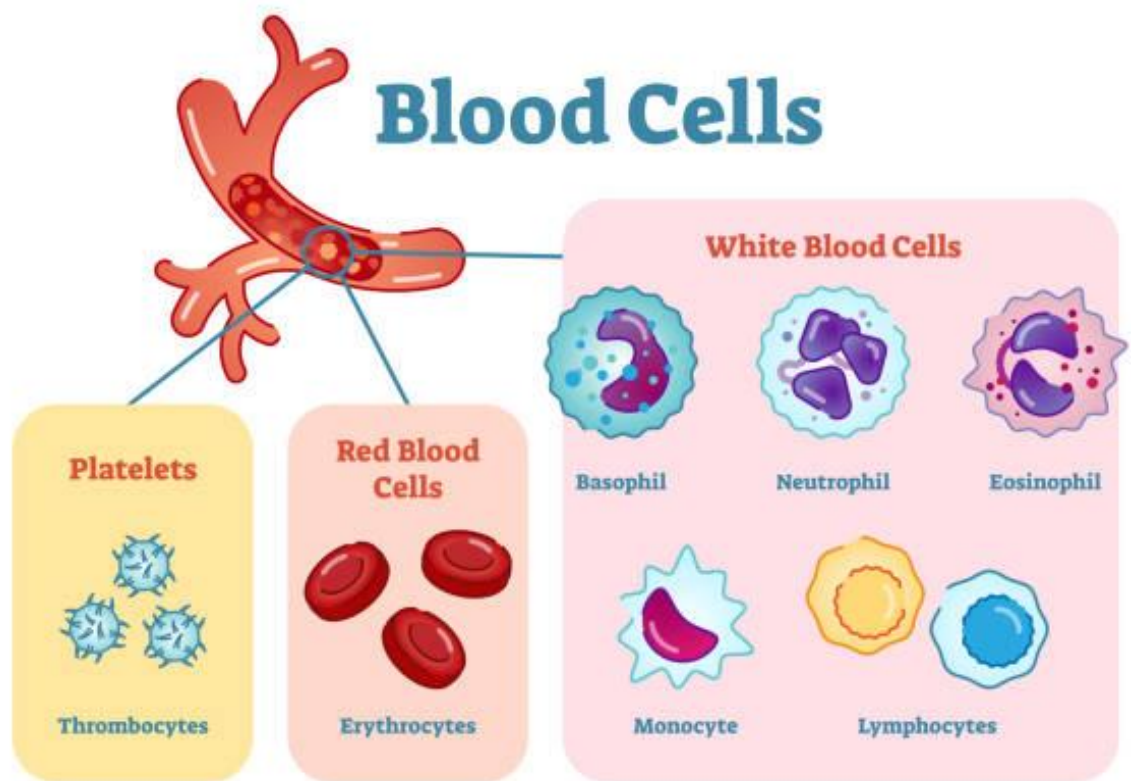


Fig 1.: white blood cells

1.2 Types of white blood cells

Among your white blood cells are:

- **Monocytes.** They have a longer lifespan than many white blood cells and help to break down bacteria. In the context of the previous discussion on Convolutional Neural Networks (CNNs) for white blood cell classification, monocytes would be one of the cell types that the model aims to accurately identify and classify. The CNN's ability to extract intricate features from images helps it distinguish monocytes from other white blood cell types based on their distinct morphological characteristics.
- **Lymphocytes.** They create antibodies to fight against bacteria, viruses, and other potentially harmful invaders. Lymphocytes are another important type of white blood cell that forms a key component of the immune system. They play a central role in the body's defense against infections and diseases, including viral and bacterial infections. Lymphocytes are characterized by a large, rounded nucleus and can be further classified into subtypes such as T cells, B cells, and natural killer (NK) cells, each with specific functions in immune response.

- **Neutrophils.** They kill and digest bacteria and fungi. They are the most numerous type of white blood cell and your first line of defense when infection strikes. Neutrophils are a type of white blood cell, or leukocyte, that plays a crucial role in the immune system's response to infection. They are part of the innate immune system, which provides immediate and non-specific defense against a variety of pathogens, including bacteria and fungi.
- **Eosinophils.** They attack and kill parasites and cancer cells, and help with allergic responses. that plays a crucial role in the immune system, particularly in response to parasitic infections and allergic reactions. These cells have a distinctive bi-lobed nucleus and contain granules that can be stained with eosin, giving them a characteristic reddish color under certain staining techniques.

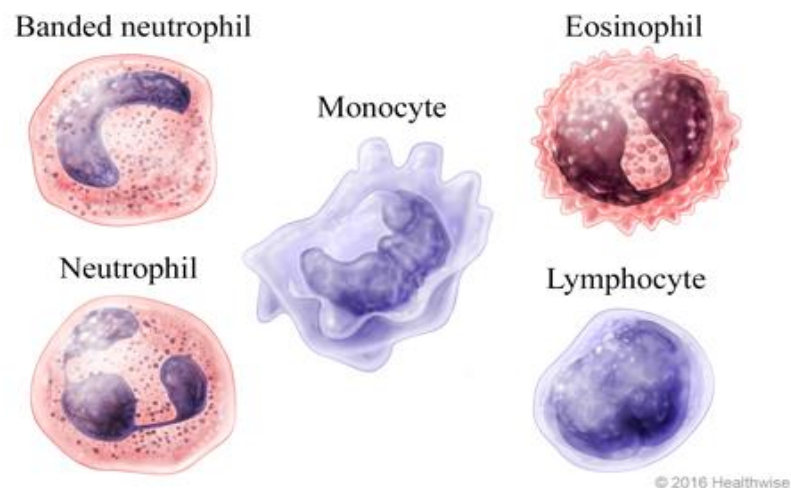


Fig .2 : types of white blood cells

The lifespan of white blood cells ranges from 13 to 20 days, after which time they are destroyed in the lymphatic system. When immature WBCs are first released from the bone marrow into the peripheral blood, they are called "bands" or "stabs." Leukocytes fight infection through a process known as phagocytosis. During phagocytosis, the leukocytes surround and destroy foreign organisms. White blood cells also produce, transport, and distribute antibodies as part of the body's immune response.

Two measurements of white blood cells are commonly done in a CBC:

The total number of white blood cells in a microliter (1×10^{-6} liters) of blood, reported as an absolute number of "X" thousands of white blood cells, and the percentage of each of the five types of white blood cells. This test is known as a differential or "diff" and is reported in

percentages in the system of structured architecture.

Normal values for total WBC and differential in adult males and females are:

- Total WBC: 4,500 - 10,000
- Bands or stabs: 3 - 5 %
- Granulocytes (or polymorphonuclears)
- Neutrophils (or segs): 50 - 70% relative value (2500-7000 absolute value)
- Eosinophils: 1 - 3% relative value (100-300 absolute value)
- Basophils: 0.4% - 1% relative value (40-100 absolute value)
- Agranulocytes (or mononuclears)
- Lymphocytes: 25 - 35% relative value (1700-3500 absolute value)
- Monocytes: 4 - 6% relative value (200-600 absolute value)

Each differential always adds up to 100%. To make an accurate assessment, consider both relative and absolute values. For example a relative value of 70% neutrophils may seem within normal limits; however, if the total WBC is 20,000, the absolute value (70% x 20,000) would be an abnormally high count of 14,000

The numbers of leukocytes changes with age and during pregnancy.

- On the day of birth, a newborn has a high white blood cell count, ranging from 9,000 to 30,000 leukocytes. This number falls to adult levels within two weeks.
- The percentage of neutrophils is high for the first few weeks after birth, but then lymphocyte predominance is seen.
- Until about 8 years of age, lymphocytes are more predominant than neutrophils.
- In the elderly, the total WBC decreases slightly.
- Pregnancy results in a leukocytosis, primarily due to an increase in neutrophils with a slight increase in lymphocytes.

Leukocytosis, a WBC above 10,000, is usually due to an increase in one of the five types of

white blood cells and is given the name of the cell that shows the primary increase.

- Neutrophilic leukocytosis = neutrophilia
- Lymphocytic leukocytosis = lymphocytosis
- Eosinophilic leukocytosis = eosinophilia
- Monocytic leukocytosis = monocytosis
- Basophilic leukocytosis = basophilia

In response to an acute infection, trauma, or inflammation, white blood cells release a substance called colony-stimulating factor (CSF). CSF stimulates the bone marrow to increase white blood cell production. In a person with normally functioning bone marrow, the numbers of white blood cells can double within hours if needed. An increase in the number of circulating leukocytes is rarely due to an increase in all five types of leukocytes. When this occurs, it is most often due to dehydration and hemoconcentration. In some diseases, such as measles, pertussis and sepsis, the increase in white blood cells is so dramatic that the picture resembles leukemia. Leukemoid reaction, leukocytosis of a temporary nature, must be differentiated from leukemia, where the leukocytosis is both permanent and progressive.

Therapy with steroids modifies the leukocytosis response. When corticosteroids are given to healthy persons, the WBC count rises. However, when corticosteroids are given to a person with a severe infection, the infection can spread significantly without producing an expected WBC rise. An important concept to remember is that, leukocytosis as a sign of infection can be masked in a patient taking corticosteroids.

Leukopenia occurs when the **WBC** falls below 4,000. Viral infections, overwhelming bacterial infections, and bone marrow disorders can all cause leukopenia. Patients with severe leukopenia should be protected from anything that interrupts skin integrity, placing them at risk for an infection that they do not have enough white blood cells to fight. For example, leukopenic patients should not have intramuscular injections, rectal temperatures or enemas.

Drugs that may produce leukopenia include:

- Antimetabolites
- Barbiturates
- Antibiotics

- Anticonvulsants
- Antithyroid drugs
- Arsenicals
- Antineoplastics
- Cardiovascular drugs
- Diuretics
- Analgesics and anti-inflammatory drugs
- Heavy metal intoxication

Leukocytes: critical low and high values

- A WBC of less than 500 places the patient at risk for a fatal infection.
- A WBC over 30,000 indicates massive infection or a serious disease such as leukemia.

When a patient is receiving chemotherapy that suppresses bone marrow production of

leukocytes, the point at which the count is lowest is referred to as the nadir.

1.3 Problems affecting white blood cells

Your white blood cell count can be low for a number of reasons. This includes when something is destroying the cells more quickly than the body can replenish them. Or when the bone marrow stops making enough white blood cells to keep you healthy. When your white blood cell count is low, you are at great risk for any illness or infection, which can spiral into a serious health threat.

White blood cells can be affected by various factors and conditions, leading to health issues. Common problems include infections (bacterial and viral), autoimmune disorders (e.g., rheumatoid arthritis), bone marrow disorders (e.g., leukemia), the impact of cancer treatments (chemotherapy and radiation), certain medications causing leukopenia, immune system disorders, allergic reactions, nutritional deficiencies, and chronic stress. These issues can compromise the production and function of white blood cells, weakening the immune system and increasing susceptibility to infections and other health complications. Addressing the underlying causes is essential for effective diagnosis and treatment. Your healthcare provider can do a blood test to see whether your white blood cell count is normal. If your count is too low or too high, you may have a white blood cell disorder.

Conditions that impact the bone marrow, such as leukemia or myelodysplastic syndromes,

can disturb the production and maturation of white blood cells. Treatments like chemotherapy and radiation, while targeting cancer cells, can also suppress bone marrow function, affecting the production of white blood cells and compromising the immune system.

Certain medications may have side effects that influence white blood cell counts or function. Malnutrition, characterized by insufficient nutrients, can weaken the immune system, including the white blood cells. Hematological cancers, like leukemia and lymphoma, directly affect white blood cells and their ability to function normally. Immune deficiencies, whether inherited or acquired, can result in decreased white blood cell numbers or impaired immune responses. Viral infections, such as HIV, can specifically target and reduce the number of certain white blood cells, leading to immunodeficiency. Chronic inflammatory conditions, like rheumatoid arthritis, can disrupt the balance of white blood cell populations and their normal activities.

Exposure to high levels of radiation, either due to accidents or medical treatments, can damage the bone marrow and impact white blood cell production. Prolonged stress has been linked to immune system dysfunction, potentially affecting the behavior of white blood cells. Additionally, exposure to certain toxins or chemicals can have adverse effects on the bone marrow and white blood cell production.

In summary, disruptions in white blood cell function or count can stem from a diverse range of factors, including infections, autoimmune disorders, bone marrow disorders, treatments like chemotherapy, medication side effects, malnutrition, hematological cancers, immune deficiencies, viral infections, inflammatory conditions, radiation exposure, stress, and toxic exposures. Identifying the specific cause is crucial for implementing targeted and effective treatments. If there are concerns about white blood cell issues, consulting with a healthcare professional is recommended for a thorough evaluation and appropriate diagnostic measures. Issues affecting white blood cells can result from various factors, impacting their production, function, or overall count. Factors such as infections, autoimmune disorders, bone marrow disorders (e.g., leukemia), chemotherapy and radiation treatments, medication side effects, malnutrition, hematological cancers, immune deficiencies, viral infections (e.g., HIV), inflammatory disorders, radiation exposure, stress, and toxic exposures can all play a role. These factors may disrupt the normal balance of white blood cells, leading to conditions that compromise the immune system and increase susceptibility to infections or other health issues. If someone suspects problems with their white blood cells, consulting a healthcare

professional for a comprehensive evaluation and appropriate diagnostic tests is essential for accurate identification and treatment of underlying concerns.

A number of diseases and conditions may affect white blood cell levels:

- **Weak immune system.** This is often caused by illnesses such as HIV/AIDS or by cancer treatment. Cancer treatments such as chemotherapy or radiation therapy can destroy white blood cells and leave you at risk for infection.
- **Infection.** A higher-than-normal white blood cell count usually means you have some type of infection. White blood cells are multiplying to destroy the bacteria or virus.
- **Myelodysplastic syndrome.** This condition causes abnormal production of blood cells. This includes white blood cells in the bone marrow.
- **Cancer of the blood.** Cancers including leukemia and lymphoma can cause uncontrolled growth of an abnormal type of blood cell in the bone marrow. This results in a greatly increased risk for infection or serious bleeding.
- **Myeloproliferative disorder.** This disorder refers to various conditions that trigger the excessive production of immature blood cells. This can result in an unhealthy balance of all types of blood cells in the bone marrow and too many or too few white blood cells in the blood.
- **Medicines.** Some medicines can raise or lower the body's white blood cell count.
- Conditions such as extreme physical stress caused by an injury or emotional stress can also trigger high white blood cell levels. So can inflammation, labor or the end of pregnancy, smoking, or even extreme exercise.

2.LITERATURE SURVEY

Introduction

Rosyadi et al. conducted a research that is able to classify WBC from blood cell images taken from blood smear samples using digital microscope. The researchers utilized Otsu threshold method for segmentation and K-Means clustering method for classification. Based on their research it was concluded that upon execution of k- means clustering to classify and count WBC, the most significant geometry feature is its circularity generating an accuracy of 67%.

Gautam et al. proposed a method which utilizes Naïve Bayes classifier and morphological features to classify WBC. The features which the researchers used to train their system were; area, eccentricity, perimeter and circularity. The proposed method was able to generate 80.88% accuracy.

Yu et al. proposed a method which uses CNN to automatically classify WBCs. The researchers utilized the network architectures; ResNet50, Inception V3, VGG 16, VGG 19, and Xception. The proposed method was able to generate an accuracy of 88.5%.

Recently, the study on the field of CNN showed to be increasingly significant in the advancement of image classification. There have been various types of CNN that was used by previous researchers. However, recent models proved to be more efficient on the improvement of image classification accuracy specifically on tasks such as object detection and segment

Sno	Authors	Title	Year	Summary	Journal
1	Esteva,A.et al.	Dermatologist level classification of skin cancer with deep neural network	2017	Demonstrates the potential of deep neural networks in classifying medical images, providing insights for medical image analysis tasks.	Nature
2	Bandyopadhyay, S. K. et al.	Leukocyte Classification Using Convolutional Neural Network with Multiscale Feature Extraction.	2017	Introduces a CNN-based approach for leukocyte classification with multiscale feature extraction, showcasing effectiveness in identifying blood cells.	Computational and Mathematical Methods in Medicine
3	Islam, M. T. et al.	Automatic Classification of White Blood Cells using Deep Convolutional Neural Networks.	2019	Presents an automated system for white blood cell classification using CNNs, demonstrating promising results in subtype identification.	IEEE ICCSP
4	Li, S. et al	Automatic White Blood Cell Classification Using Pre-Trained Deep Learning Models: ResNet and Inception.	2019	Investigates the application of pre-trained deep learning models (ResNet, Inception) for white blood cell classification, highlighting the potential of leveraging pre-trained architectures.	IEEE BIBM

Table 1 : literature survey

3. PROPOSED METHODOLOGY

3.1 Existing system

The existing System uses the tradition machine learning algorithms to classify the different types of blood cells on the basis of the data in the dataset. The current system for analyzing white blood cells relies on manual methods, where medical professionals manually examine blood smears or microscopic images. This process is labor-intensive, time-consuming, and subjective, leading to potential inconsistencies in results. The dependence on human expertise, limited throughput, and the inability to handle large datasets are notable challenges. To address these issues, there is a need for more efficient and automated systems, such as Convolutional Neural Networks (CNNs), to enhance the speed and accuracy of white blood cell analysis.

Algorithms like K-means Clustering and Random Forest Techniques are used in the existing methods. The algorithm processes the images and classifies the white blood cells from the data in the database by using the K-nearest neighbor algorithm.

Advantages of the Existing System:

Familiarity with Traditional Algorithms: The existing system utilizes traditional machine learning algorithms like K-means Clustering and Random Forest Techniques, which may be well-established and understood in the medical field.

Manual Expertise: Human expertise is involved in the current system, allowing for nuanced analysis and interpretation of blood cell samples. Skilled medical professionals can provide valuable insights.

Established Processes: The manual examination of blood smears or microscopic images is a well-established process in many healthcare settings, and professionals may be accustomed to these procedures.

Initial Implementation Costs: The initial implementation of traditional machine learning algorithms may have lower resource and financial requirements compared to more advanced technologies.

3.1.1 Disadvantages of Existing system

The accuracy of the algorithm can be improved using latest algorithms. The image processing rate of the existing System is limited and can be optimized. The dataset is too large and takes too much time to process. Human error is inherent in manual methods, and fatigue or distractions can further increase the likelihood of mistakes during the analysis of numerous samples. The existing system requires significant resources, including skilled personnel, specialized equipment, and dedicated time. This can pose challenges, especially in resource-limited healthcare settings. As medical datasets grow in size and complexity, the manual approach becomes less efficient and may struggle to handle the increased volume of data.

3.2 Proposed System

The Proposed system is a CNN based Deep Learning Algorithm in which the system will be trained from the images which are preprocessed before being taken to the neural network. All the images are made into required scale and the algorithm will train upon the items present in the training part of the database. The trained model will be tested on the testing part of the database. Using all this we get an accuracy of about 96% using CNN model in classifying the White Blood Cells. In this system, the reliance on manual inspection is significantly reduced, if not eliminated.

The proposed CNN-based model is designed to autonomously identify and classify white blood cells by learning intricate features and patterns from a diverse dataset during the training phase. This automated approach aims to overcome the subjectivity and potential inconsistencies associated with manual interpretation. The labor-intensive nature of the existing system is addressed through automation. The CNN is capable of processing a large number of blood cell images in a relatively short time, substantially increasing the throughput of the diagnostic process. This not only expedites the analysis but also allows for a more timely diagnosis and treatment planning.

By minimizing the dependence on human expertise, the proposed system mitigates the risk of errors and inconsistencies in white blood cell classification. The CNN's ability to capture specific spatial dependencies and features enhances its accuracy in distinguishing between different cell types, including rare or complex morphologies. The proposed system is also poised to address resource limitations. While the existing manual system requires significant resources, the automated approach offers scalability, potentially making white blood cell analysis more accessible in various healthcare settings.

Accuracy stands as a pivotal metric in the realm of Convolutional Neural Network (CNN) assessment, particularly in the nuanced landscape of white blood cell (WBC) analysis via microscopic imagery. At its core, accuracy encapsulates the essence of correct classification, embodying the ratio of accurately identified instances to the total pool of observations. In the intricate domain of WBC analysis, accuracy serves as a beacon, guiding the evaluation of CNN models' abilities to discern diverse cell types and detect anomalies indicative of diseases or irregularities within the immune system. A high accuracy score underscores the CNN's proficiency in traversing the intricacies of WBC morphology, effectively learning and generalizing patterns from the training dataset to make precise predictions on novel test samples. This metric serves as a testament to the model's robustness, indicating its capability to navigate through the complexities inherent in microscopic images and extract meaningful features crucial for classification.

However, the omnipotence of accuracy necessitates a nuanced interpretation, especially in scenarios where class imbalances prevail or the costs associated with false positives and false negatives diverge. In such contexts, a high accuracy score may not adequately reflect the CNN model's performance, as it may disproportionately prioritize dominant classes or fail to address critical misclassifications. Consequently, a holistic evaluation framework encompasses a spectrum of complementary metrics, including precision, recall, F1 score, and the confusion matrix. These metrics offer deeper insights into the CNN's performance, delineating its strengths and limitations across various facets of WBC analysis. Furthermore, considerations of computational efficiency, model interpretability, and generalization capabilities augment the evaluation process, fostering a comprehensive understanding of the CNN's efficacy in real-world applications. In essence, while accuracy stands as a cornerstone metric in CNN evaluation for WBC analysis, its true significance unfolds within the broader tapestry of performance assessment, where it interweaves with an ensemble of metrics to provide a comprehensive narrative of the model's effectiveness and utility in clinical practice.

3.2.1 Advantages of Proposed System

Advantages of the System is the accuracy is more. Once the model is trained the system will take very less time for classifying the blood cells. The dataset is of small size with all the required items for training and testing the model. Automation and Efficiency: The automated nature of the proposed system significantly reduces the reliance on manual inspection, leading to a more efficient and faster analysis of white blood cells. This is particularly beneficial in handling large volumes of samples in a timely manner.

Accuracy and Consistency: CNNs are capable of learning intricate features and patterns from diverse datasets, improving the accuracy of white blood cell identification and classification. The system reduces the subjectivity and inconsistencies associated with manual interpretation, leading to more reliable results.

High Throughput: The proposed system can process a large number of blood cell images quickly, increasing the throughput of the diagnostic process. This allows for a more timely analysis and facilitates prompt decision-making in healthcare settings.

Reduced Dependence on Human Expertise: By leveraging machine learning algorithms, the proposed system minimizes the dependence on the expertise of individual pathologists or technicians. This helps mitigate the risk of human error and ensures consistent performance across various cases.

Handling Variability: CNNs are designed to capture specific spatial dependencies and features, making the system more robust in handling variations in cell morphology, staining techniques, and other factors. This enhances its ability to accurately classify different types of white blood cells.

3.3 Purpose of the System

The main purpose of the project is to classify the type of white blood cells based on the lesion image. Convolution neural networks is used for classifying the tumour. The might have already spread to the different parts of the body making it extremely dangerous. To revolutionize the existing manual methods of white blood cell analysis by introducing an automated approach through the integration of Convolutional Neural Networks (CNNs). The system aims to significantly enhance the efficiency, accuracy, and scalability of white blood cell identification and classification. By automating the process, the proposed system reduces the labor-intensive nature of manual inspection, leading to faster and more consistent analyses. It addresses the subjectivity associated with human interpretation, providing a reliable and objective method for identifying different types of white blood cells. Through high-throughput capabilities, the system ensures timely diagnoses, particularly crucial in healthcare settings with large workloads. The reduction in dependence on human expertise minimizes the risk of errors and facilitates consistent performance across various cases. The system's ability to handle variability in cell morphology and staining techniques contributes to its robustness. Additionally, scalability makes the proposed system adaptable to diverse

healthcare environments, making advanced white blood cell analysis more accessible. Overall, the purpose of the system is to leverage technology to advance the field of hematology, offering an innovative and efficient tool for improved diagnostic outcomes.

We aim to build a machine learning model that would help the patient in the detection of the white bloodcell type. Due to the fine-grained differences in the appearance of skin lesions, automated classification is quite challenging through images. To attain highly segregated and potentially general tasks against the finely grained object categorized, images are processed and features are extracted using the neural networks. We use a convolution neural network and train the system over the features using machine learning.

Scope of the System

The scope of the project is to find the type of tumour if the provided image of the white blood cell lesion. This is done by processing the image data set to extract the features and then training the model to give accurate predictions. The proposed system holds a broad scope within the field of hematology, primarily focusing on automating the process of white blood cell analysis through the utilization of Convolutional Neural Networks (CNNs). It aims to bring efficiency to the identification and classification of white blood cells by reducing the manual labor involved in the process. The system's scope extends to efficient data processing, allowing it to handle a substantial volume of blood cell images in a timely manner.

One of the key objectives is to enhance the accuracy and consistency of white blood cell analysis. By leveraging machine learning, the system aims to mitigate the potential for human error and ensure reliable results across various cases. It seeks adaptability to different healthcare environments, ranging from large hospitals to smaller clinics, making automated white blood cell analysis accessible in diverse settings.

The project aims to utilize Convolutional Neural Networks (CNNs) to detect white blood cells (WBCs) from microscopic images, a task crucial for medical diagnostics and research. The scope encompasses several key stages, starting with the acquisition of a diverse dataset of microscopic images containing various types of WBCs. These images undergo meticulous preprocessing to enhance their quality and prepare them for input into the CNN model. The selection of an appropriate CNN architecture is critical, considering factors such as model performance and computational efficiency. Through extensive model training, which involves feeding the preprocessed images into the CNN and adjusting its parameters through

backpropagation, the model learns to accurately detect WBCs. Evaluation of the trained model's performance on a separate test dataset ensures its ability to generalize to unseen data and provides reliable predictions. Further fine-tuning and optimization techniques are applied to enhance the model's capabilities, such as transfer learning and hyperparameter tuning.

Once the model demonstrates satisfactory performance, it is deployed into a production environment, where it can be integrated into medical imaging systems or other relevant applications. Validation in a real-world setting ensures the model's effectiveness and reliability, with feedback from end-users and stakeholders guiding iterative improvements. Comprehensive documentation and reporting of the project's methodology, findings, and conclusions facilitate knowledge dissemination and future research endeavors. Through these concerted efforts, the project endeavors to contribute to the advancement of medical diagnostics and the understanding of white blood cell biology.

3.4 Current Scenario

Yet, diagnosis is still a visual process, which relies on the long-winded procedure of clinical screenings, followed by dermoscopic analysis, and then a biopsy and finally a histopathological examination. This process easily takes time and the need for many medical professionals.

The sheer amount of time and technicalities it takes when diagnosing the patient (let alone beginning their treatment) and the many opportunities for human error, leaves thousands of deaths annually. In recent years, there has been an increasing interest in incorporating automation and advanced technologies into the field of hematology. Some laboratories have started to explore the use of image analysis systems and machine learning techniques to assist or automate aspects of white blood cell identification and classification

However, the widespread adoption of automated systems, especially those based on deep learning techniques like Convolutional Neural Networks (CNNs), may vary across different healthcare institutions and regions. The integration of such advanced technologies is influenced by factors such as infrastructure, resources, regulatory considerations, and the level of awareness and acceptance within the medical community. Despite their effectiveness, CNNs for WBC analysis still face challenges such as the need for large annotated datasets, standardization of imaging protocols, and validation against diverse patient populations. Addressing these challenges presents opportunities for collaboration between researchers,

clinicians, and technology developers to further improve the accuracy and applicability of CNN-based diagnostic tools. CNNs for WBC analysis still face challenges such as the need for large annotated datasets, standardization of imaging protocols, and validation against diverse patient populations.

Addressing these challenges presents opportunities for collaboration between researchers, clinicians, and technology developers to further improve the accuracy and applicability of CNN-based diagnostic tools. Their integration into clinical practice holds promise for improving healthcare outcomes and enhancing our understanding of WBC-related diseases. the project is contributing to the broader scientific community by generating insights into WBC biology and pathology.

By analyzing large volumes of microscopic images and leveraging AI-driven image analysis techniques, researchers can uncover novel patterns and correlations that may inform our understanding of immune system function, inflammatory processes, and disease mechanisms. This project is characterized by a convergence of cutting-edge AI technologies, interdisciplinary collaborations, and a growing recognition of the transformative potential of AI in healthcare. As the project continues to evolve, it holds promise for revolutionizing WBC detection and advancing our understanding of human health. One of the key objectives is to enhance the accuracy and consistency of white blood cell analysis. By leveraging machine learning, the system aims to mitigate the potential for human error and ensure reliable results across various cases. It seeks adaptability to different healthcare environments, ranging from large hospitals to smaller clinics, making automated white blood cell analysis accessible in diverse settings.

Functional requirements:

The Functional Requirements Specification documents the operations and activities that a system must be able to perform.

Functional Requirements should include:

1. Descriptions of data to be entered into the system
2. Descriptions of operations performed by each screen
3. Descriptions of work-flows performed by the system
4. Descriptions of system reports or other outputs

5. CNN based diagnostic can enter the data into the system.
6. Interface system meets applicable regulatory requirements
7. Functional Requirements usually define if/then behaviour and include calculations.

The Functional Requirements Specification is designed to be read by a general audience. Readers should understand the system, but no particular technical knowledge should be required to understand the document.

Input

The user will only be required to input the lesion image in the user interface for which he/she wants to know the type of the lesion. He does not require registration for using the service.

Interface

We will develop the interface using a flask framework which is easy for creating a web application for projects using the python programming language.

Output

The user will obtain a lesion label as output for which he gave input.

Non-functional requirements:

Non-functional requirements are often called "quality attributes" of a system. Other terms for non-functional requirements are "qualities", "quality goals", "quality of service requirements", "constraints" and "non-behavioral requirements". Informally these are sometimes called the "utilities", from attributes like stability and portability. Qualities—that is non-functional requirements—can be divided into two main categories. Execution qualities, such as safety, security, and usability, which are observable during operation (at run time). Evolution qualities, such as testability maintainability, extensibility, and scalability, which are embodied in the static structure of the system.

Non-functional requirements, often referred to as "quality attributes" of a system, play a crucial role in shaping its overall performance and user experience. These requirements are sometimes also termed as "qualities", "quality goals", "quality of service requirements", "constraints", or "non-behavioral requirements". They encompass a range of characteristics that contribute to the system's utility and effectiveness beyond its basic functionality.

These qualities can be broadly categorized into two main groups: execution qualities and evolution qualities. Execution qualities are those aspects of the system that are observable during its operation, typically at runtime. Examples of execution qualities include safety, ensuring that the system operates without causing harm; security, safeguarding against unauthorized access or data breaches; and usability, ensuring that the system is intuitive and user-friendly.

On the other hand, evolution qualities are inherent in the static structure of the system and dictate its long-term maintainability and adaptability. Testability refers to the ease with which the system can be tested to ensure its correctness and reliability. Maintainability pertains to the ability to efficiently update and modify the system to accommodate changes or fix issues. Extensibility involves the ease of adding new features or functionalities to the system without significant rework. Scalability relates to the system's capability to handle increasing loads or user demands without sacrificing performance.

Overall, non-functional requirements are critical for ensuring that a system not only functions correctly but also meets the expectations and needs of its users while being robust, secure, and adaptable to future changes. Paying attention to these requirements from the early stages of system design and development is essential for delivering a high-quality product that satisfies both functional and non-functional criteria.

Reliability

The system produces reliable output which is true

Accuracy

The results produced by the system must be accurate.

Performance

The performance of the system is high which produces faster output.

Modifiability

The system can be easily modified if we want to make any new changes.

Software specifications

Operating System	:	Windows 10
IDE	:	Jupyter Notebook(Anaconda)
Coding Language	:	Python
Framework	:	Flask
Technologies used	:	Keras over TensorFlow

Software specifications deal with defining software resources prerequisites and specifications that are installed on the computer to provide optimal functioning of the application. These specifications are generally not included in the software installation package and need to be installed separately before the software is installed.

Windows OS:

Windows OS, a computer operating system(OS) developed by Microsoft Corporation to run personal computers. Featuring the first graphical user interface for I compatible PCs, the Windows OS soon dominated the PC market. Approximately 90 percent of PCs run some version of the Windows OS.

Anaconda:

The open-source Anaconda Distribution id the easiest way to perform Python/R data science and machine learning in Linux, Windows, and Mac OS. With over 11 million users worldwide, it is the industry standard for developing, testing, and training on a single machine, enabling individual data scientists to:

- Anaconda is open-source and free to use, fostering collaboration and innovation within the Python community.
- Quickly download 1500+ Python/R data science packages.
- Manage libraries, dependencies, and environments with anaconda.
- Develop and train machine learning and deep learning models with sci-kit learn, Tensor Flow
- Analyze the data with scalability and performance with Numpy, Pandas, and matplotlib.

3.5 Python:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding; make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Guido van Rossum created the Python Programming Language in the late 1980s. In contrast to other programming languages such as C, C++, and Java, Python strives to provide a simple but powerful syntax. Python is used for software development companies and organizations such as Google, Yahoo, CERN, and NASA. Experienced programmers can accomplish great things with Python but Python's beauty is that it is accessible to beginning programmers and allows them to tackle interesting problems more quickly than many other, more complex languages that have a steeper learning curve. Python's appeal lies in its clear and concise syntax, which facilitates rapid development and iteration.

The language's extensive standard library and rich ecosystem of third-party packages cater to a wide range of use cases, from building web applications with frameworks like Django and Flask to developing machine learning models with libraries like TensorFlow and PyTorch. Moreover, Python's cross-platform compatibility enables developers to write code once and deploy it across different operating systems, making it an attractive choice for multi-platform applications. One of the key objectives is to enhance the accuracy and consistency of white blood cell analysis. By leveraging machine learning, the system aims to mitigate the potential for human error and ensure reliable results across various cases. It seeks adaptability to different healthcare environments, ranging from large hospitals to smaller clinics, making automated white blood cell analysis accessible in diverse settings.

Its open-source nature fosters a vibrant community of developers who contribute to its evolution, ensuring continuous improvements and updates. In recent years, Python's ascendancy in emerging technologies such as data science, artificial intelligence, and automation has solidified its position as a go-to language for tackling complex and

interdisciplinary challenges. Its versatility, coupled with its ease of learning and use, makes Python a powerhouse in the modern computing landscape, driving innovation and empowering developers worldwide. Python's scalability and performance have improved with advancements like Just-In-Time (JIT) compilation and asynchronous programming. While Python may not match the raw speed of languages like C or C++, its versatility, ease of use, and extensive library support make it a compelling choice for a wide range of applications, from small scripts to large-scale enterprises.

3.6 Keras:

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. Keras, a powerful deep learning API, serves as a high-level interface to the TensorFlow machine learning platform, combining flexibility and efficiency in model development. This open-source library was designed to streamline the process of experimenting with deep learning models, emphasizing a rapid transition from conceptualization to obtaining results – a crucial aspect in efficient research and development.

With a foundation in Python, Keras simplifies the implementation of complex neural network architectures. Its user-friendly and intuitive design facilitates the creation of models for various machine learning tasks, particularly those associated with modern deep learning techniques. As the high-level API of TensorFlow, Keras provides a productive environment for solving machine learning challenges. It abstracts away many complexities, offering essential building blocks and abstractions that expedite the development and deployment of machine learning solutions. Its focus on high iteration velocity means that developers and researchers can quickly iterate on their models, experimenting with different architectures and hyperparameters to optimize performance. Keras is tailored to leverage the scalability and cross-platform capabilities of TensorFlow 2.0. This means that machine learning engineers and researchers can harness the power of Tensor Processing Units (TPUs) or scale their computations across large GPU clusters. Furthermore, Keras allows for the export of trained models to different platforms, enabling deployment in diverse environments – from running models in the browser to deploying them on mobile devices. The synergy between Keras and TensorFlow 2.0 not only enhances the development process but also facilitates the execution of machine learning models on advanced hardware setups. This versatility ensures that practitioners can harness the computational power required for training complex models efficiently.

In summary, Keras serves as a versatile and accessible tool for developing and deploying deep learning models. Its integration with TensorFlow 2.0 provides a robust foundation for scalability and cross-platform deployment, empowering researchers and engineers to explore and implement cutting-edge machine learning solutions with ease. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research. Keras is the high-level API of TensorFlow an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity. Keras empowers engineers and researchers to take full advantage of the scalability and cross-platform capabilities of TensorFlow 2.0: you can run Keras on TPU or large clusters of GPUs, and you can export your Keras models to run in the browser or on a mobile device.

3.7 Tensorflow:

TensorFlow is an open-source software library for numerical computation using data-flow graphs. It was originally developed by the Google Brain Team within Google's Machine Intelligence research organization for machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well. It reached version 1.0 in February 2017 and has continued rapid development, with 21,000+ commits thus far, many from outside contributors. This article introduces TensorFlow, its open-source community and ecosystem, and highlights some interesting TensorFlow open-sourced models. TensorFlow incorporates high-level.

TensorFlow, an open-source software library, has emerged as a cornerstone in the field of numerical computation, particularly in the domain of machine learning and deep neural networks research. Initially conceived by the Google Brain Team within Google's Machine Intelligence research organization, TensorFlow quickly gained traction due to its flexibility and scalability, making it applicable across a wide range of domains beyond its original intended use. Since its initial release, TensorFlow has undergone significant evolution, culminating in version 1.0 being reached in February 2017. However, development has not slowed down; rather, it has continued at a rapid pace, with over 21,000 commits contributed by both Google engineers and external contributors alike. This active development community has played a vital role in expanding TensorFlow's capabilities and ensuring its relevance in an ever-changing technological landscape.

One of the key strengths of TensorFlow lies in its ability to represent computations as data-flow graphs, providing a flexible framework for building and deploying various machine learning models. This high-level abstraction simplifies the process of designing complex neural networks and enables researchers and developers to focus on algorithmic innovations rather than low-level implementation details.

Beyond its core functionality, TensorFlow boasts a thriving open-source community and ecosystem. This ecosystem encompasses a wide range of tools, libraries, and resources built on top of TensorFlow, further enhancing its utility and accessibility. From pre-trained models and datasets to specialized frameworks for specific tasks like natural language processing or computer vision, the TensorFlow ecosystem offers a wealth of resources to support researchers, developers, and practitioners in their machine learning endeavors. In addition to its technical capabilities, TensorFlow's open-source nature fosters collaboration and knowledge-sharing within the machine learning community. By providing transparent access to its source code and encouraging contributions from individuals and organizations worldwide, TensorFlow has become a catalyst for innovation and advancement in the field of artificial intelligence.

TensorFlow represents more than just a software library; it embodies a vibrant community-driven ecosystem that continues to push the boundaries of what is possible in machine learning and artificial intelligence. Its widespread adoption and ongoing development serve as a testament to its versatility, scalability, and enduring relevance in the fast-paced world of technology.

APIs like Keras, tightly integrated in TensorFlow 2.x, simplifying the process of defining, training, and evaluating neural network models. Estimators provide an abstraction for training, evaluation, and prediction, aiding in the creation of complex models. TensorBoard, a visualization tool, aids in debugging by displaying the computation graph, monitoring metrics, and providing insights into model performance. Distributed TensorFlow supports the scaling of machine learning workflows across multiple devices or servers, facilitating the handling of large datasets and complex models.

TensorFlow Architecture

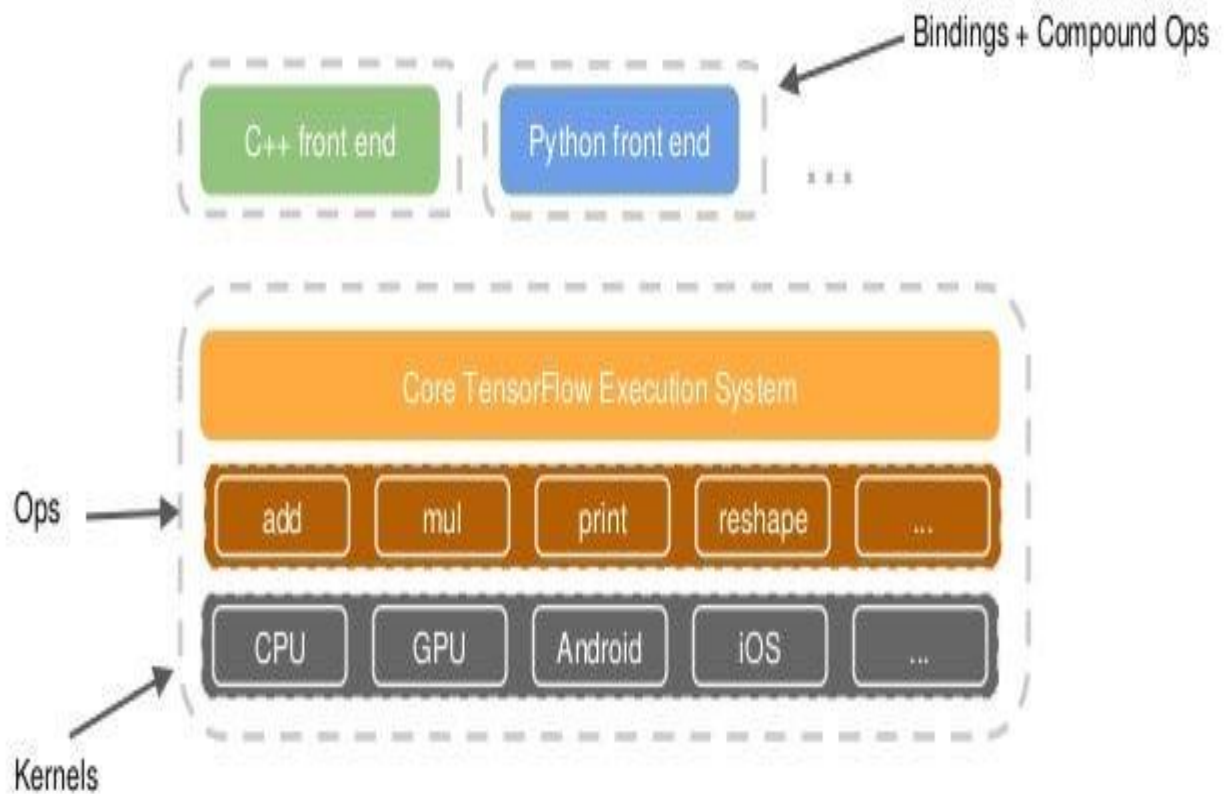


Fig 3:TensorFlow architecture

Hardware specifications

PROCESSOR	: i5RAM , 4GB
HARD DISK	: 5GB (minimum)

4. EXPERIMENTAL RESULTS & DISCUSSION

4.1 System Architecture:

System architecture is a conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structure and behavior of the system. A representation of a system, including a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and human interaction with these components. System architecture is a crucial aspect of software and hardware design, providing a conceptual framework that encompasses the structure, behavior, and various perspectives of a system. It serves as a blueprint for creating and understanding complex systems, offering a high-level view that aids in reasoning about the system's functionality and organization.

An architecture description is a formal representation of the system, designed to convey its intricacies in a structured manner. This description includes details about the arrangement of components, their interactions, and the overall system behavior. The goal is to create a comprehensive document that not only captures the current state of the system but also supports future modifications and enhancements.

The mapping of functionality onto hardware and software components is a critical aspect of system architecture. This involves defining how different tasks and operations are distributed across the underlying hardware infrastructure and the corresponding software modules. It ensures that the system's computational resources are utilized efficiently and effectively.

Equally important is the mapping of the software architecture onto the hardware architecture. This step involves aligning the software components with the hardware resources available, optimizing performance, and ensuring that the overall system meets its functional and non-functional requirements. It addresses issues such as load balancing, resource utilization, and scalability. Human interaction with system components is an integral part of system architecture. It involves understanding how users interact with the system, including user interfaces, input mechanisms, and overall user experience. This aspect considers not only the technical functionalities but also the usability and accessibility of the system, ensuring a positive interaction between users and the deployed architecture.

In essence, system architecture is a holistic approach to designing and describing complex systems. It goes beyond a mere representation of components and interactions, delving into the mappings between software and hardware, as well as the dynamics of human-system interaction. A well-defined system architecture provides a roadmap for development, maintenance, and evolution, contributing to the success and sustainability of a technological solution.

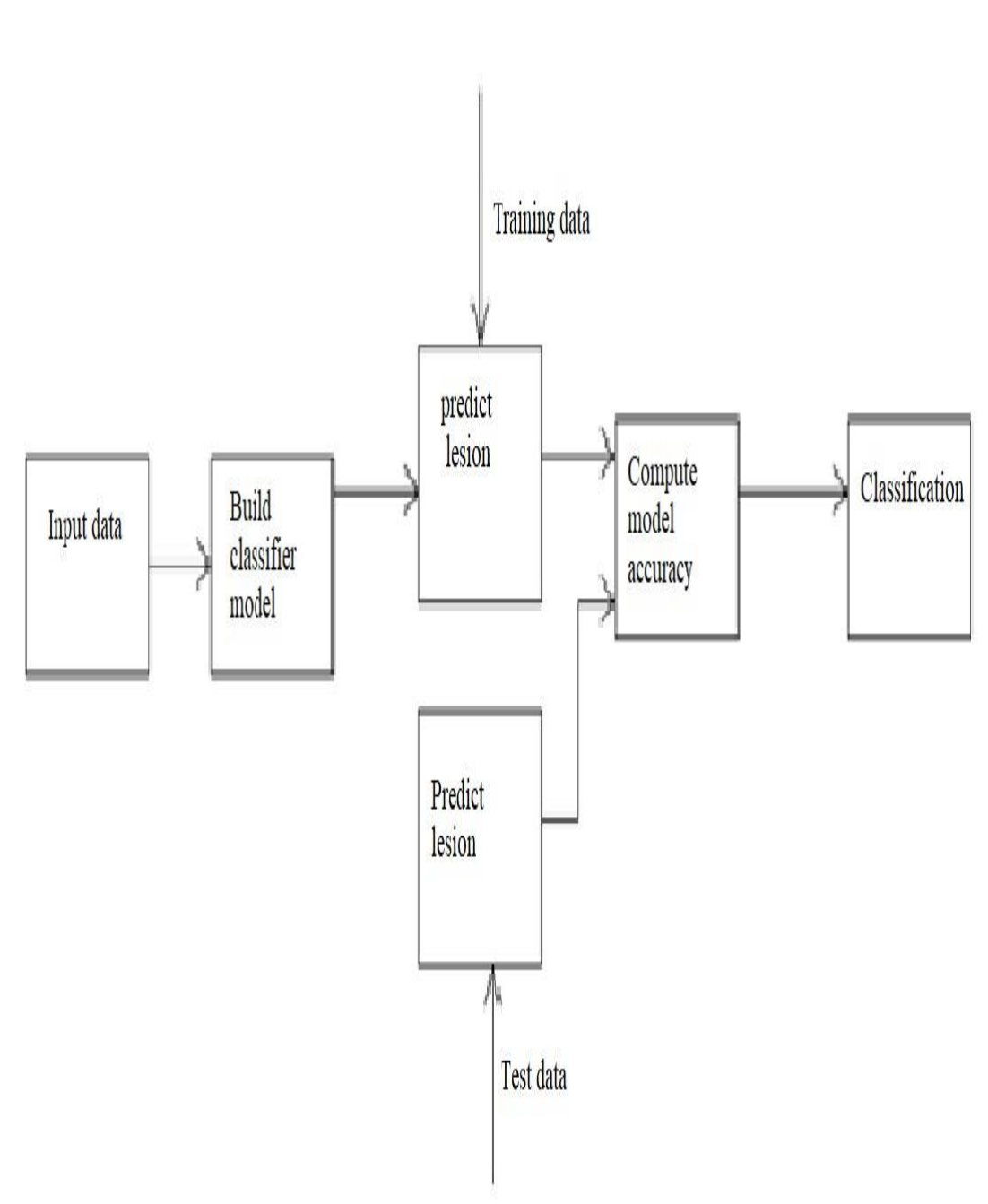


Fig 4: System architecture

4.2 System design:

The Unified Modelling Language (UML) allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic-semantic and pragmatic rules. The Unified Modelling Language is commonly used to visualize and construct software-intensive systems. Because software has become much more complex in recent years, developers are finding it more challenging to build complex applications within short periods. Even when they do, these software applications are often filled with bugs, and it can take programmers weeks to find and fix them.

This is time that has been wasted, since an approach could have been used which would have reduced the number of bugs before the application was completed. In system design, the Unified Modeling Language (UML) serves as a powerful tool for expressing analysis models. Its modeling notation adheres to a set of syntactic-semantic and pragmatic rules, providing a standardized way for software engineers to visualize and construct software-intensive systems. With the increasing complexity of software applications, developers face challenges in building intricate systems within tight timeframes. This complexity often leads to applications filled with bugs, and the process of identifying and fixing these issues can be time-consuming.

While UML itself is not a methodology, it offers a set of diagrams that, when integrated into a methodology, enhance the comprehension of an application under development. Despite its non-mandatory nature for formal work products, UML diagrams contribute significantly to understanding and communication within software development projects. The diagrams serve as a valuable introduction to the language and the underlying principles, providing a common visual language that aids in expressing and sharing design concepts.

Beyond the diagrams, UML encompasses a broader set of capabilities. However, for the context of system design, these diagrams play a pivotal role. By incorporating standard UML diagrams into the methodology's work products, teams can facilitate the seamless integration of UML-proficient individuals into the project, enhancing productivity and collaboration.

A UML system is typically represented using five different views, each offering a unique perspective on the system. These views are articulated through a set of diagrams, contributing to a comprehensive understanding of the system's architecture, behavior, and interactions. The diverse set of diagrams allows stakeholders to explore the system from various angles,

fostering a more holistic approach to system design. UML provides a standardized and versatile approach to system design through its modeling notation. It addresses the challenges posed by complex software development by offering a visual language that enhances communication and understanding. The incorporation of UML diagrams into methodologies contributes to a more accessible and collaborative development process, ultimately leading to more efficient and successful software projects. Unified Modeling Language (UML) serves as a powerful tool for expressing analysis models. Its modeling notation adheres to a set of syntactic-semantic and pragmatic rules, providing a standardized way for software engineers to visualize and construct software-intensive systems. With the increasing complexity of software applications, developers face challenges in building intricate systems within tight timeframes. This complexity often leads to applications filled with bugs, and the process of identifying and fixing these issues can be time-consuming. While UML itself is not a methodology, it offers a set of diagrams that, when integrated into a methodology, enhance the comprehension of an application under development. Since UML is not a methodology, it does not require any formal work products. Yet it does provide several types of diagrams that, when used within a given methodology, increase the ease of understanding an application under development. There is more to UML than these diagrams, but for my purposes here, the diagrams offer a good introduction to the language and the principles behind its use. By placing standard UML diagrams in your methodology's work products, you make it easier for UML-proficient people to join your project and quickly become productive.

A UML system is represented using five different views that describe the system from a distinctly different perspective. Each view is defined by a set of diagrams which is as follows.

4.3 UML DIAGRAMS:

Use Case Diagram

Use Case diagrams identify the functionality provided by the system (use cases), the users who interact with the system (actors), and the association between the users and the functionality. Use Cases are used in the Analysis phase of software development to articulate the high-level requirements of the system. The primary goals of the Use Case diagrams include:

1. Providing a high-level view of what the system does.
2. Identifying the users ("actors") of the system.
3. Determining areas needing human-computer interfaces

Sequence Diagram:

Sequence diagrams document the interactions between classes to achieve a result, such as a use case. Because UML is designed for object-oriented programming, these communications between classes are known as messages. The Sequence diagram lists objects horizontally, and time vertically, and models these messages over time.

Sequence diagrams serve as powerful tools for visualizing and documenting the interactions between various components or classes within a system. At their core, these diagrams capture the flow of messages exchanged between objects over time, providing a clear representation of the dynamic behavior underlying a particular use case or scenario.

In a sequence diagram, objects or components involved in the interaction are depicted horizontally along the top of the diagram. Each object is represented by a lifeline, which extends vertically downwards, indicating the object's existence over the course of the interaction. Time is represented vertically, with the sequence of events progressing from top to bottom.

Messages exchanged between objects are represented by arrows or lines connecting the respective lifelines, indicating the direction of communication. These messages can take various forms, including method calls, signal emissions, or data exchanges, depending on the nature of the interaction being modeled. Sequence diagrams can also include additional elements to provide context and clarity to the interaction. For example, activation bars can be used to depict the period during which an object is actively processing a message, while notes and comments can annotate specific parts of the diagram to explain behavior or highlight important details.

Overall, sequence diagrams offer a concise and intuitive way to understand the runtime behavior of a system, allowing developers, designers, and stakeholders to identify potential bottlenecks, dependencies, or communication patterns. By visually representing the flow of messages between objects, sequence diagrams facilitate communication and collaboration among team members, aiding in the design, analysis, and validation of complex systems and use cases.

Activity Diagram:

The activity diagram displays a special state diagram where most of the states are action states and most of the transitions are triggered by the completion of the actions in the source states. This diagram focuses on flows driven by internal processing. Activity diagrams provide a graphical representation of workflows or processes within a system, emphasizing the flow of activities driven by internal processing. Unlike traditional state diagrams, which primarily focus on the states and transitions of objects, activity diagrams primarily consist of action states and transitions triggered by the completion of these actions.

In an activity diagram, actions represent specific tasks or operations that occur within the system. These actions can range from simple computations or data manipulations to more complex processes involving multiple steps. Each action state is depicted as a rectangle with rounded corners, indicating the execution of a particular activity.

Transitions between action states indicate the flow of control or the sequence of activities within the system. These transitions are typically triggered by the completion of actions in the source state, signifying that the system is ready to move to the next activity. Arrows connecting the action states represent these transitions, with optional labels to denote conditions or events triggering the transition.

In addition to action states and transitions, activity diagrams may also include other elements to enhance clarity and understanding. For example, decision nodes allow the diagram to depict branching or conditional flows, where the next activity depends on the outcome of a decision or evaluation. Fork and join nodes enable the diagram to model parallel activities, where multiple actions can occur concurrently before converging back into a single flow. Overall, activity diagrams offer a visual means of representing complex workflows or processes, making them invaluable tools for system analysis, design, and documentation. By focusing on the internal processing and flow of activities within a system, activity diagrams help stakeholders understand how tasks are performed, identify potential bottlenecks or inefficiencies, and ensure that the system behaves as intended under various conditions. . These transitions are typically triggered by the completion of actions in the source state, signifying that the system is ready to move to the next activity. Arrows connecting the action states represent these transitions, with optional labels to denote conditions or events triggering the transition.

Use Case Diagram:

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. A use case diagram provides a visual representation of the interactions between actors (users or external systems) and the system under consideration. It showcases the various ways in which actors interact with the system to achieve specific goals or functionalities. Here's a description of a generic use case diagram

List of Use cases:

- Input dataset
- Pre-processing data
- Partition dataset
- CNN model
- Predict

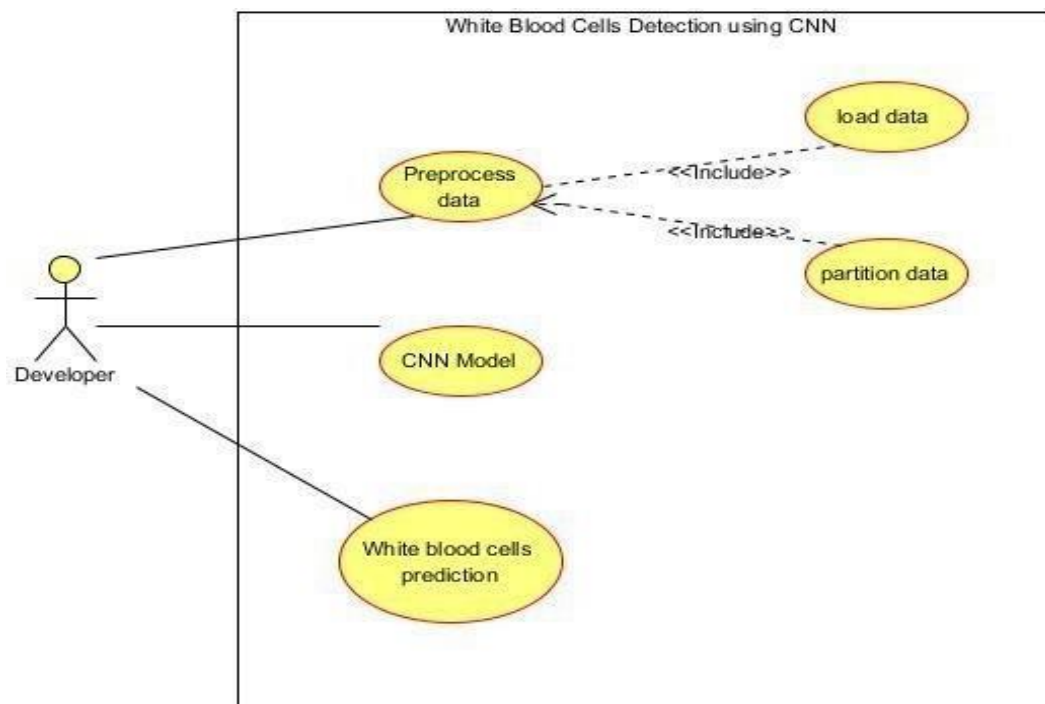


Fig 5: Usecase diagram for developer

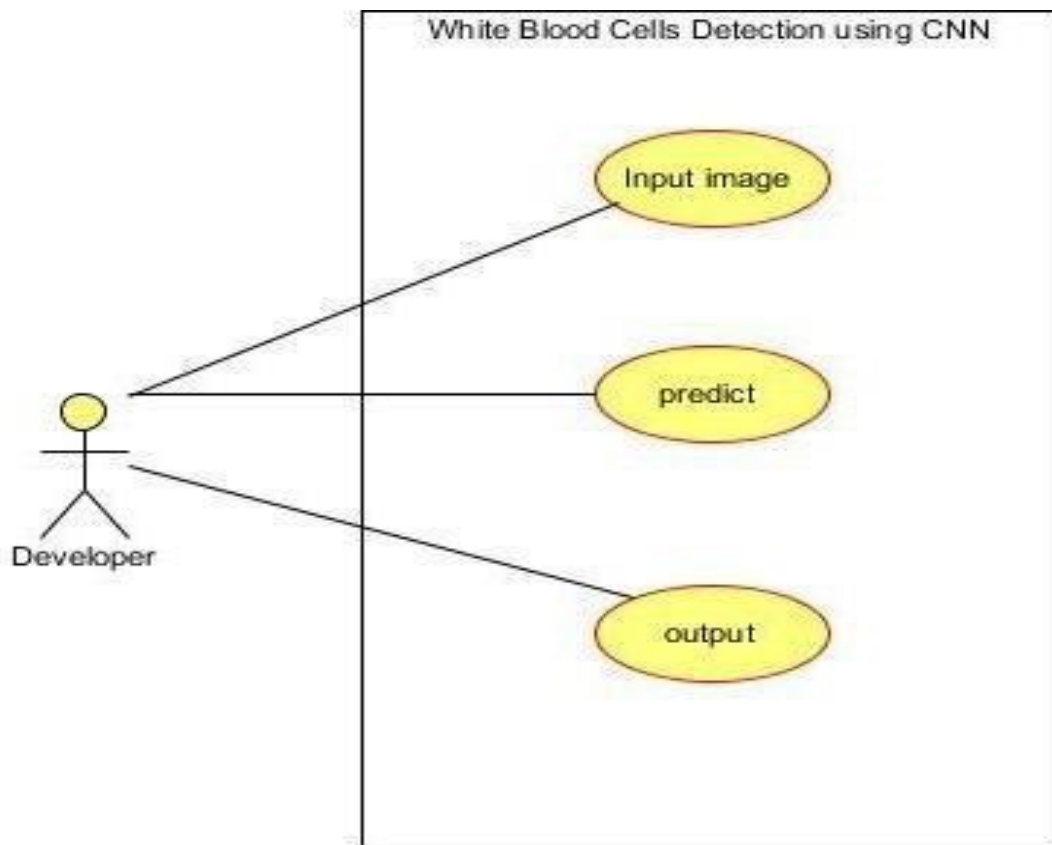


Fig 6 : Usecase diagram for user

Sequence Diagram

A sequence diagram is an interaction diagram that emphasizes the time ordering of messages. A sequence diagram shows a set of objects and the messages sent and received by those objects. The objects are typically named or anonymous instances of classes, but may also represent instances of other things, such as collaborations, components, and node. Transitions between action states indicate the flow of control or the sequence of activities within the system. These transitions are typically triggered by the completion of actions in the source state, signifying that the system is ready to move to the next activity. Arrows connecting the action states represent these transitions, with optional labels to denote conditions or events triggering the transition.

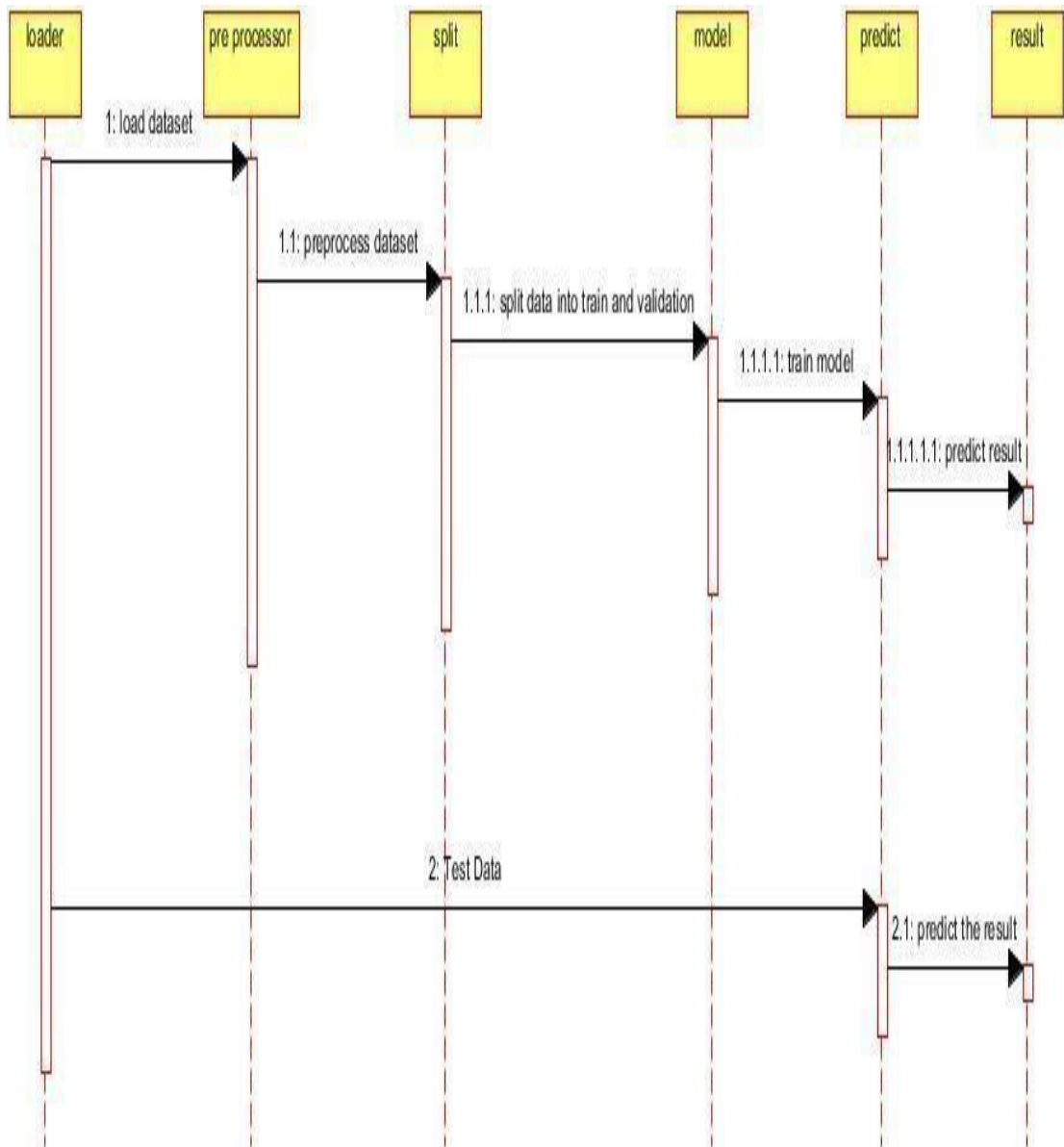


Fig 7 sequence diagram

Activity Diagram:

Activity diagrams are one of the five diagrams in the UML for modeling the dynamic aspects of the systems. An activity diagram is essentially a flow chart, showing the flow of control from activity to activity. We use an activity diagram to model the dynamic aspects of the system. For the most, this involves modeling the sequential (and possibly concurrent) steps in a computational process. With an activity diagram, you can also model the flow of an object as it moves from state to state at different points in the flow of a control. Activity diagrams may stand alone to visualize, specify, construct, and document the dynamics of a society of

objects, or they may be used to model the flow of control. These diagrams are instrumental in modeling various computational processes, offering a clear representation of how tasks are executed within the system. Whether it involves a linear sequence of actions or simultaneous activities occurring in parallel, activity diagrams offer a comprehensive view of the system's dynamic behavior.

Moreover, activity diagrams extend beyond mere process modeling; they also facilitate the depiction of object flow within the system. By illustrating how objects transition between different states throughout the control flow, these diagrams offer insights into the system's behavior and interactions. Activity diagrams can function independently as a means to visualize, specify, construct, and document the dynamics of a system. Alternatively, they may be integrated into larger models to represent the flow of control within the context of other UML diagrams, providing a holistic view of system behavior and structure.

In summary, activity diagrams play a crucial role in system modeling by providing a visual representation of dynamic behavior, enabling stakeholders to understand, analyze, and communicate complex processes effectively

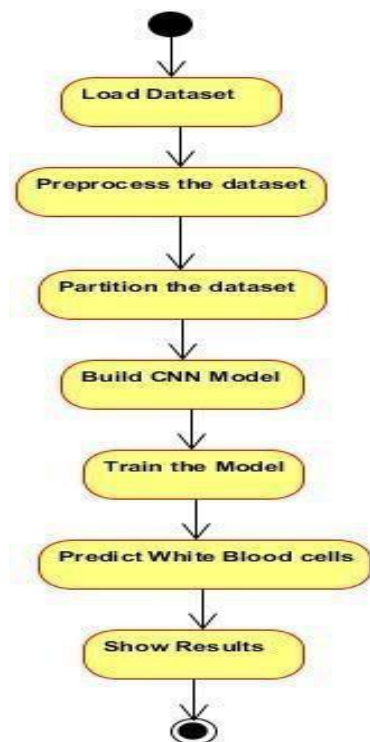


Fig8 Activity diagram

4.4 Neural networks:

A neural network is a computing system made up of several simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs. Neural networks are typically organized in layers. Neural networks are versatile and can be applied to various tasks, such as image recognition, natural language processing, and regression analysis. The architecture of a neural network includes an input layer, one or more hidden layers, and an output layer. The hidden layers allow the network to learn complex representations from the input data.

Training a neural network involves presenting it with labeled examples to adjust the weights and biases, allowing the model to generalize patterns and make accurate predictions on new, unseen data. Deep neural networks, characterized by multiple hidden layers, have shown remarkable success in capturing intricate features from complex datasets. Neural networks, inspired by the human brain's interconnected neurons, constitute a computing system where simple processing elements dynamically respond to external inputs. These networks, often organized into layers, exhibit versatility across a spectrum of applications such as image recognition, natural language processing, and regression analysis. The fundamental architecture involves an input layer, one or more hidden layers, and an output layer. The hidden layers play a crucial role in enabling the network to discern complex representations from the input data, facilitating the extraction of intricate features.

Training a neural network is a process of refining its parameters, including weights and biases, by presenting it with labeled examples. Through this iterative learning process, the model generalizes patterns and enhances its ability to make accurate predictions on new, unseen data. Notably, deep neural networks, characterized by multiple hidden layers, have demonstrated remarkable success in capturing nuanced features from intricate datasets, contributing to advancements in various domains.

Several types of neural networks cater to specific tasks, addressing challenges within their respective domains. Feedforward neural networks, with information flowing in one direction, serve general-purpose applications. Convolutional Neural Networks (CNNs) specialize in image processing tasks by leveraging spatial hierarchies. Recurrent Neural Networks (RNNs) excel in handling sequential data, making them suitable for tasks involving time series or natural language processing. Transformers, another type, have proven effective in processing

sequential data and are prominently used in advanced natural language processing applications. The neural network's structure comprises layers of interconnected nodes, each hosting an activation function. Patterns are introduced to the network through the input layer, undergo processing in one or more hidden layers via weighted connections, and eventually produce an output in the output layer. This layered arrangement allows neural networks to discern complex patterns and relationships within data, contributing to their effectiveness in diverse applications.

The continual evolution of neural network architectures, training methodologies, and the exploration of novel activation functions contribute to the ongoing advancements in machine learning and artificial intelligence. As researchers and practitioners delve deeper into the intricacies of neural networks, the potential for innovation and breakthroughs in various fields remains promising.

Key types of neural networks include feedforward neural networks, convolutional neural networks (CNNs) for image processing, recurrent neural networks (RNNs) for sequence data, and transformers for natural language processing tasks. Each type is tailored to address specific challenges within its domain. Layers are made up of several interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is output.

Convolutional Neural networks:

In traditional feed-forward neural networks, each neuron in the input layer is connected to every output neuron in the next layer – we call this a fully connected(FC) layer. However, on CNN,we don't use FC layers until the very last layers in the network.We can thus define a CNN as a neural network that swaps in a specialized —convolutional layer in place of a fully-connected layer for at least one of the layers in the network.

A nonlinear activation function, such as ReLU, is then applied to the output of these convolutions and the process of convolution => activation continues along with a mixture of other layer types to help reduce the width and height of the input volume and help reduce the width and height of the input volume and help reduce overfitting until we finally reach the end of the network and apply one or two FC layers where we can obtain our final output

classifications. The distinguishing feature of CNNs is the use of convolutional layers. These layers apply convolution operations to input images, employing filters or kernels to extract spatial hierarchies of features. This enables CNNs to capture local patterns, textures, and complex visual structures, making them well-suited for image-related tasks.

Typical CNN architectures consist of convolutional layers, pooling layers for down-sampling, and fully connected layers for making predictions. Convolutional layers perform feature extraction, while pooling layers reduce the spatial dimensions of the data, focusing on essential information. Fully connected layers combine extracted features for final predictions. CNNs have demonstrated superior performance compared to traditional image processing techniques, largely due to their ability to automatically learn relevant features from raw data. Transfer learning, where pre-trained CNN models are fine-tuned for specific tasks, has become a common practice, leveraging the knowledge gained from large datasets.

Beyond image processing, CNNs have found applications in diverse fields, including natural language processing, drug discovery, and medical image analysis. Their adaptability and effectiveness stem from their ability to automatically learn hierarchical representations from input data, making them a powerful tool in the broader landscape of machine learning. Each layer in a CNN applies a different set of filters, typically hundreds or thousands of them, and combines the results, feeding the output into the next layer in the network. During training, a CNN automatically learns the values for these filters. In the context of image classification, our CNN may learn to:

- Detect edges from raw pixel data in the first layer.
- Use these edges to detect shapes in the second layer.
- Use these shapes to detect higher-level features in the highest layers of the network.

The last layer of a CNN uses these higher-level features to make predictions regarding the contents of the image. In practice, CNNs give us two key benefits: local invariance and compositionality. The concept of local invariance allows us to classify an image as containing a particular object regardless of where in the image the object appears. We obtain this local invariance through the usage of —pooling layers‖ which identifies regions of our input volume with a high response to a particular filter. The second benefit is compositionality. Each filter composes a local patch of lower-level features into a higher-level representation, similar to how we can compose a set of mathematical functions that build on the output of previous functions. This composition allows our network to learn more rich features deeper

in the network. For example, our network may build edges from pixels, shapes from edges, and then complex objects from shapes – all in an automated fashion that happens naturally during the training process. The concept of building higher-level features from lower-level ones is exactly why CNNs are so powerful in computer vision. To understand the contents of the image, we must apply image classification, which is the task of using computer vision and machine learning algorithms to extract meaning from an image. This action could be as simple as assigning a label to what the image contains, or as advanced as interpreting the contents of an image and returning a human-readable sentence.

Layer types:

There are many types of layers used to build convolutional neural networks, but the ones we are most likely to encounter include:

- Convolutional Layer
- Pooling Layer
- Fully-connected Layer
- Dropout Layer

Sure, let's explore these layers in a more narrative fashion:

As considering the requirement of the project we are using Convolutional(CONV), Pooling(Pool), Fully Connected(FC) as our layers in the model.

Stacking a series of these layers in a specific manner yields a CNN. We often use simple text diagrams to describe a CNN: INPUT => CONV => RELU=> POOL => FC=> SOFTMAX. Here we define a simple CNN that accepts input, applies a convolution layer, then an activation layer, then a fully-connected layer, and finally, a softmax classifier to obtain the output classification probabilities. The softmax activation layer is often omitted from the network diagram as it is assumed it directly follows the final FullyConnected layer. Of these layer types, CONV and FC are the only layers that obtain parameters that are learned during the training process. Activation and dropout layers are not considered true —layers themselves but are often included in network diagrams to make the architecture explicitly clear. Pooling layers(Pool), of equal importance as CONV and FC, are also included in network diagrams as they have a substantial impact on the spatial dimensions of an image as

it moves through a CNN. CONV, POOL, and FC are the most important when defining your actual network architecture. That's not to say that the other layers are not critical, but take a backseat to this critical set of four as they define the actual architecture itself.

Convolutional Layer The Convolutional Layer is fundamental to CNNs and performs convolution operations on input data. It uses filters or kernels to scan the input, capturing spatial hierarchies of features. These learned filters enable the network to identify patterns and representations within the data.

Pooling Layer: The Pooling Layer is responsible for down-sampling and reducing the spatial dimensions of the data. Max pooling and average pooling are common techniques used to retain the most significant information while discarding less relevant details. Pooling helps to make the network more robust to variations in scale and orientation.

Fully-connected Layer: The Fully-connected Layer, also known as a dense layer, connects every neuron in one layer to every neuron in the next layer. In CNNs, fully-connected layers are typically used towards the end of the network to combine extracted features and make final predictions.

Dropout Layer: These transitions are typically triggered by the completion of actions in the source state, signifying that the system is ready to move to the next activity. Arrows connecting the action states represent these transitions, with optional labels to denote conditions or events triggering the transition.

Typical CNN architectures consist of convolutional layers, pooling layers for down-sampling, and fully connected layers for making predictions. Convolutional layers perform feature extraction, while pooling layers reduce the spatial dimensions of the data, focusing on essential information. Fully connected layers combine extracted features for final predictions. These layer types work in concert to form the architecture of a CNN. Convolutional layers extract local features, pooling layers down-sample the data, fully-connected layers combine extracted features for classification, and dropout layers mitigate overfitting. Additional layers may include normalization layers, activation layers (e.g., ReLU - Rectified Linear Unit), and others. The arrangement and combination of these layers in a CNN create a hierarchical structure that enables the model to understand and interpret complex patterns within the input data, making them particularly effective in tasks like image recognition and computer vision.

. These transitions are typically triggered by the completion of actions in the source state, signifying that the system is ready to move to the next activity. Arrows connecting the action states represent these transitions, with optional labels to denote conditions or events triggering the transition.

4.5 Convolutional Layer:

The CONVOLUTIONAL layer is the core building block of a Convolutional Neural Network. The CONV layer parameters consist of a set of K learnable filters (i.e., —kernels), where each filter has a width and a height, and are nearly always square. These filters are small (in terms of their spatial dimensions) but extend throughout the full depth of the volume. The convolutional layer is a fundamental component of convolutional neural networks (CNNs) used in machine learning, particularly for tasks like image recognition. Think of this layer as a specialized filter that scans through the input data to identify and extract features. These features could be simple, like edges or textures, or more complex patterns that are crucial for recognizing objects in images.

The convolutional layer serves as a cornerstone in Convolutional Neural Networks (CNNs), pivotal for tasks such as image recognition. It comprises a set of learnable filters, commonly referred to as kernels, each characterized by width and height dimensions, typically square-shaped. These filters are relatively small spatially but span the entire depth of the input volume. Operating akin to specialized filters, the convolutional layer traverses the input data, discerning and extracting features essential for subsequent processing. These features can range from elementary elements like edges or textures to intricate patterns crucial for discerning objects in images. In contrast to conventional neural network layers, the convolutional layer doesn't establish connections between every input and output neuron. Instead, it employs small, trainable filters to capture specific patterns within the input. Through the process of convolution, wherein the filter iteratively computes dot products with the input at different positions, feature maps are generated, spotlighting the presence of diverse patterns. The efficacy of the convolutional layer lies in its ability to detect hierarchical features within data. For instance, in the realm of image analysis, initial layers might specialize in detecting rudimentary features like edges, while deeper layers progressively discern more intricate combinations of these edges, eventually culminating in the recognition of complete objects. Unlike traditional neural network layers, the convolutional layer doesn't connect every input neuron to every output neuron. Instead, it uses small, learnable filters to capture specific patterns within the input. These filters move across the input data, performing a mathematical operation known as convolution. This operation involves taking the dot product of the filter and the input at different positions, producing feature maps that highlight the presence of various patterns.

The convolutional layer is effective in detecting hierarchical features in data. For example, in image processing, early layers might learn simple features like edges, while deeper layers learn more complex combinations of these edges, eventually recognizing entire objects.

This layer's ability to automatically learn and extract relevant features from the input data makes it crucial for tasks involving spatial relationships, such as image and video analysis. The convolutional layer's architecture allows it to capture local patterns efficiently and contribute to the overall success of convolutional neural networks in various machine learning applications.

For inputs to the CNN, the depth is the number of channels in the image (i.e., a depth of three when working with RGB images, one for each channel). For volumes in the network, the depth will be the number of filters applied in the previous layer.

To make this concept clearer, let's consider the forward-pass of a CNN, where we convolve each of the K filters across the width and height of the input volume. More simply, we can think of each of our K kernels sliding across the input region, computing an element-wise multiplication, summing, and then storing the output value in a 2-dimensional activation map, such as

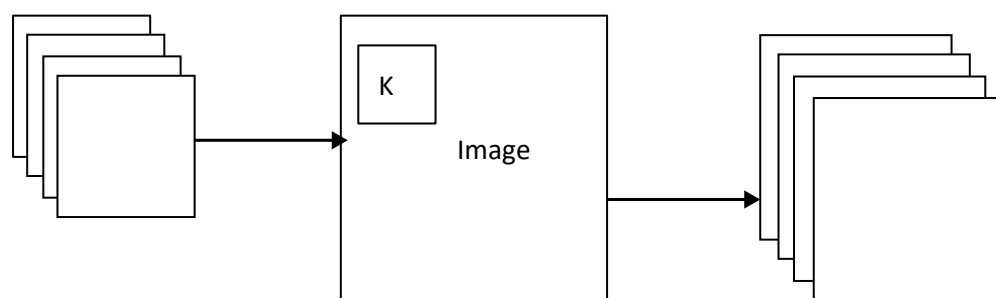


Fig 9 convolution operation

Step 1: K kernels waiting to be applied to the image.

Step 2: Each kernel is convolved with the input volume.

Step 3: The output of each Convolution operation produces a 2d output.

After applying all k filters to the input volume, we now have K , 2-dimensional activation maps. We then stack our K activation maps along the depth dimension of our array to form the final output volume.

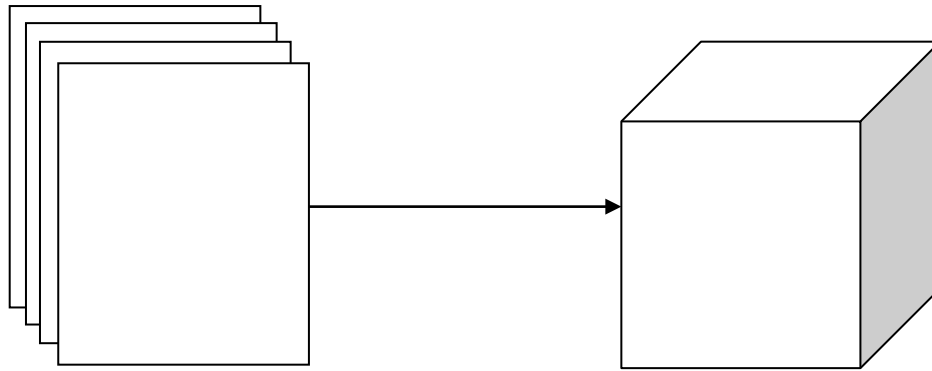


Fig 9.1 : Convolutional second stage operation

After obtaining the K activation maps, they are stacked together to form the input volume to the next layer in the network. Every entry in the output volume is thus an output of a neuron that looks at only a small region of the input. In this manner, the network learns filters that activate when they see a specific type of a feature at a given spatial location in the input volume. In lower layers of the network, filters may activate when they see edge-like or corner-like regions. Then, in the deeper layers of the network, filters may activate in the presence of high-level features, such as parts of the face, the hood of a car, etc. This activation concept goes back to our neural network analogy: these neurons are becoming excited and activating when they see a particular pattern in an input image.

The concept of convolving small filters with a large input volume has special meaning in convolutional neural networks—specifically, the local connectivity, and the receptive field of a neuron. When working with images, it's often impractical to connect neurons in the current volume to all neurons in the previous volume—there are simply too many connections and too many weights, making it impossible to train deep networks on images with large spatial dimensions. Instead, when utilizing CNNs, we choose to connect each neuron to only a local region of the input volume—we call the size of this local region the receptive field of the neuron. To make the point clear, consider a dataset where the input volume has an input size of $32 \times 32 \times 3$. Each image that has a width of 32 pixels, a height of 32 pixels, and depth of 3. If our receptive field is of size 3×3 , then each neuron in the CONV layer will connect to a 3×3 local region of the image for a total of $3 \times 3 \times 3 = 27$ weights (remember, the depth of the filters is three because they extend through the full depth of the input image, in this case, three channels). Now,

let's assume that the spatial dimensions of our input volume have been reduced to a smaller size, but our depth is now larger due to utilizing more filters deeper in the network, such that volume size is now $16 \times 16 \times 94$. Again, if we assume a receptive field of size 3×3 , then every neuron in the CONV layer will have a total of $3 \times 3 \times 94 = 846$ connections to the input volume, simply put, the receptive field F is the size of the filter, yielding an $F \times F$ kernel that is convolved with the input volume.

At this point we have explained the connectivity of neurons in the input volume, but not the arrangement or size of the output volume. Three parameters control the size of an output volume: the kernel, stride, and zero-padding size, each of which we'll review below.

4.6 KERNEL

A filter or kernel is an integral component of the layered architecture. It is a smaller-sized matrix in comparison to the dimensions of the image, that contains real-valued entries. The kernels are then convolved with the input volume to obtain so-called activation maps. Activation maps indicate activated regions, i.e. regions where features specific to the kernel have been detected in the input.

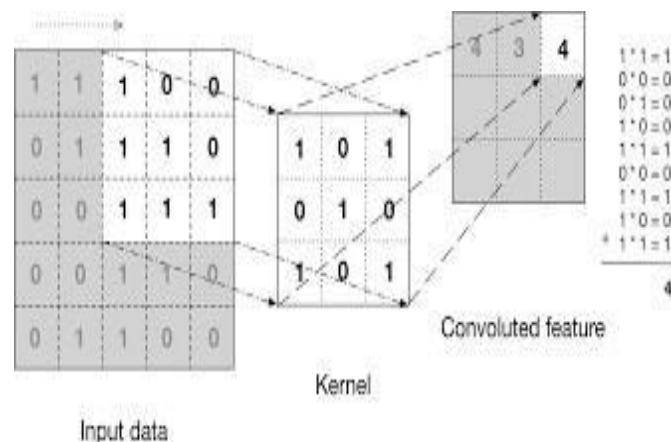


Fig 10 Kernel operation

Kernel for the different things like image identity, edge detection, sharpening the images with To make the point clear, consider a dataset where the input volume as an input size of $32 \times 32 \times 3$. Each image that has a width of 32 pixels, a height of 32 pixels, and depth of 3. If our reception field is of size 3×3 , then each neuron in the CONV layer will connect to a 3×3 local region of the image for a total of $3 \times 3 \times 3 = 27$ weights (remember, the depth of the filters is three because they extend through the full depth of the input image, in this case, three channels).

Kernels play a crucial role in enabling CNNs to automatically learn hierarchical representations from raw input data. They help the network identify edges, textures, and more complex structures within the images. Different kernels are designed to capture various aspects of the input, allowing the network to learn diverse features as.

The size of the kernel, often a small square matrix (e.g., 3x3 or 5x5), and the number of kernels used in a layer are hyperparameters that impact the learning capacity and feature representation of the network. Effective kernel design is key to the success of CNNs in tasks like image recognition and computer vision.






Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Fig 11 Kernel types

We described a convolution operation as —sliding a small matrix across a large matrix, stopping at each coordinate, computing an element-wise multiplication and sum, then storing the output. Stride in the context of Convolutional Neural Networks (CNNs) refers to the step size with which the convolutional kernel moves across the input data during the convolution operation. It determines the spatial resolution of the resulting feature map. A larger stride value results in a down-sampled feature map, reducing the spatial dimensions, while a smaller stride produces a higher resolution map.

The stride parameter influences the output size and computational efficiency of the convolutional operation. A larger stride leads to a more aggressive down-sampling, reducing the computational cost and memory requirements. However, it may also result in information loss, as fewer positions are considered during the convolution.

In contrast, a smaller stride captures more detailed information from the input but increases computational complexity. The choice of stride is often influenced by the desired trade-off between computational efficiency and the preservation of spatial details in the feature map.

Striding is not only applied in convolutional layers but can also be used in pooling layers, where it influences the down-sampling of the feature map. The pooling layer typically follows the convolutional layer, and the combination of convolutional layers with different strides and pooling layers contributes to the hierarchical feature extraction process in CNNs.

The stride parameter is a crucial element in designing CNN architectures, allowing practitioners to tailor the network to the requirements of specific tasks and balance computational efficiency with the need for detailed feature representation. This description is similar to a sliding window that slides from left-to-right and top-to-bottom across an image.

ZERO-PADDING

It needs to pad the borders of an image to retain the original image size when applying a convolution – the same is true for filters inside of a CNN. Using zero padding, we can pad our input along the borders such that our output volume size matches our input volume size. The amount of padding we apply is controlled by the parameter `padding`. This technique is especially critical when we start looking at deep CNN architectures that apply multiple CONV filters on top of each other. To visualize zero padding, where we applied a 3x3 kernel to a 5x5 input image with a stride of stride=1.

Without zero padding, the spatial dimensions of the input volume would decrease too quickly, and we wouldn't be able to train deep networks (as the input volumes would be too tiny to learn any useful patterns from). Zero-padding is a technique used in Convolutional Neural Networks (CNNs) to add extra border pixels of value zero around the input data (such as an image) before applying convolutional operations. This padding helps to preserve spatial information and mitigate issues related to the reduction of feature map dimensions.

The addition of zero-padding allows the convolutional kernel to extend beyond the original boundaries of the input, ensuring that features near the edges are adequately considered during the convolution operation. Without zero-padding, the convolutional operation can lead to a reduction in spatial dimensions, potentially causing the loss of information at the borders. Zero-padding is specified by the number of rows and columns of zeros added to the top, bottom, left, and right of the input data. It is a hyperparameter that can be adjusted based on the desired impact on the spatial dimensions of the feature map.

This technique is especially relevant when designing CNN architectures, as it enables the network to maintain spatial resolution and capture features more effectively, particularly in the early layers where detailed information is crucial for understanding the input patterns.

Putting all these parameters together, we can compute the size of an output volume as a function of the input volume size (W, assuming the input images are square, which they nearly always are), the receptive field size F, the stride S, and the amount of zeropadding P, To construct a valid CONV layer, we need to ensure the following equation is an integer:

$$((W-F+2P)-S)+1$$

If it is not an integer, then the strides are set incorrectly, and the neurons cannot be tiled such that they symmetrically fit across the input volume.

Pooling Layers:

There are two methods to reduce the size of an input volume-CONV layers with a stride > 1 (which we've already seen) and POOL layers. It is common to insert POOL layers in-between consecutive CONV layers in CNN architectures:

INPUT -> CONV => RELU=> POOL=> CONV => RELU => POOL>FC

The primary function of the POOL layer is to progressively reduce the spatial size (ie, width and height) of the input volume. Doing this allows us to reduce the number of parameters and computation in the network- pooling also helps us control overfitting. POOL Layers operate on each of the depth slices of an input independently using either the max or average function. Max pooling is typically done in the middle of the CNN architecture to reduce the spatial size, whereas average pooling is normally used as the final layer of the network where we wish to avoid using FC layers entirely. The most common type of POOL layer is max pooling,

although this trend is changing with the introduction of more exotic micro-architectures.

Typically we'll use a pool size of 2×2 , although deeper CNNs that use larger input images (>200 pixels) may use a 3×3 pool size early in the network architecture. We also commonly set the stride to either $S=1$ or $S=2$. Figure 11.10 (heavily inspired by Karpathy et al. 2011) follows an example of applying max pooling with a 2×2 pool size and a stride of $S=1$. Notice for every 2×2 block, we keep only the largest value, take a single step (like a sliding window), and apply the operation again thus producing an output volume size of 3×3 .

We can further decrease the size of our output volume by increasing the stride - here we apply $S=2$ to the same input. For every 2×2 block in the input, we keep only the largest value. Then take a step of two pixels, and apply the operation again. This pooling allows us to reduce the width and height by a factor of two, effectively discarding 75% of activations from the previous layer.

In summary, POOL layers accept an input volume of size $W_{input} \times H_{input} \times D_{input}$. They then require two parameters:

The receptive field size F (also called the "pool size"), Stride S .

Applying the POOL operation yields an output volume of size $W_{output} \times H_{output} \times D_{output}$, where:

$$W_{output} = ((W_{input} - F) / S) + 1$$

$$H_{output} = ((H_{input} - F) / S) + 1$$

$$D_{output} = D_{input}$$

The primary function of the POOL layer is to progressively reduce the spatial size (ie, width and height) of the input volume. Doing this allows us to reduce the number of parameters and computation in the network- pooling also helps us control overfitting.

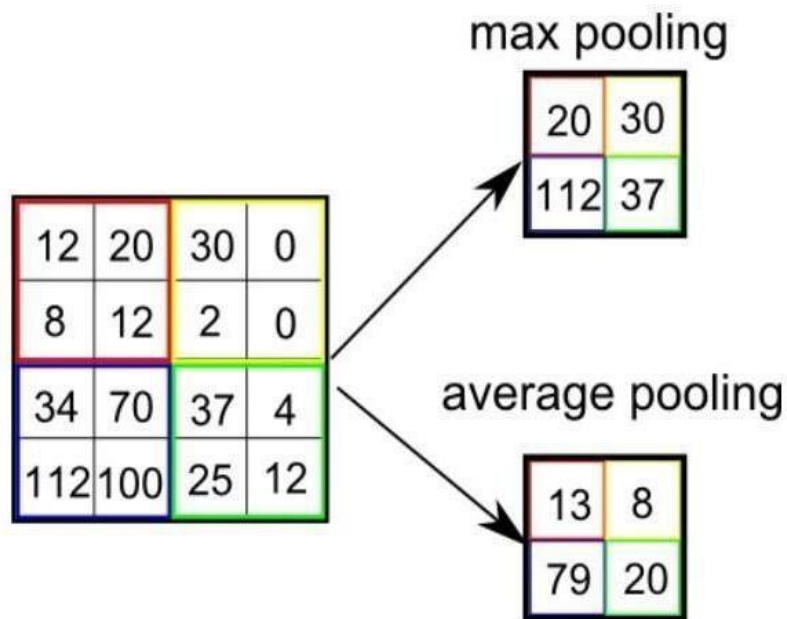


Fig 12 Pool types

Fully-connected Layers

Neurons in FC layers are fully-connected to all activations in the previous layer, as is the standard for feedforward neural networks. FC layers are always placed at the end of the network (i.e, we don't apply a CONV layer, then an FC layer, I followed by another CONV) layer.

It's common to use one or two FC layers prior to applying the softmax classifier, as the following (simplified) architecture demonstrates: INPUT-> CONV-> RELU => POOL=> CONV-> RELU=> POOL-> FC->FC.

Here we apply two fully-connected layers before our (implied) softmax classifier which will compute our final output probabilities for each class. The architecture you described is a common configuration for convolutional neural networks (CNNs). It typically consists of convolutional layers, rectified linear unit (ReLU) activation functions, pooling layers, fully connected (FC) layers, and a softmax classifier.

Fully connected layers connect every neuron from the previous layer to every neuron in the current layer. They act as a classifier, combining features extracted by earlier layers for making predictions. These layers usually employ non-linear activation functions like ReLU. The softmax classifier, typically used as the final layer in classification tasks, converts raw scores into probabilities, representing the likelihood of each class.

Overall, this architecture follows a typical CNN design for classification tasks, leveraging convolutional layers for feature extraction, pooling for dimensionality reduction, fully connected layers for feature integration, and a softmax classifier for class probabilities. It has proven effective in various computer vision tasks.

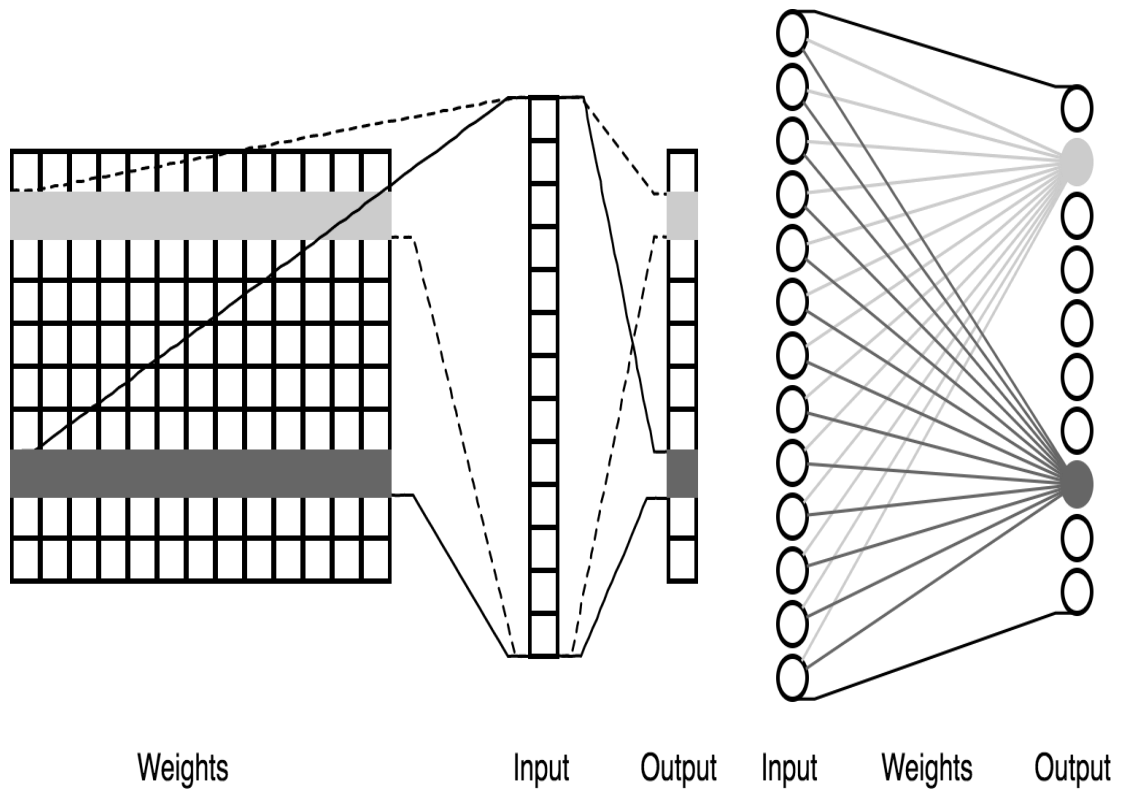


Fig 13 Fully connected layer

Softmax

The softmax function squashes the outputs of each unit to be between 0 and 1, just like a sigmoid function. But it also divides each output such that the total sum of the outputs is equal to 1. The output of the softmax function is equivalent to a categorical probability distribution. Mathematically the softmax function is shown below, where z is a vector of the inputs to the output layer.

CNN MODEL ARCHITECTURE

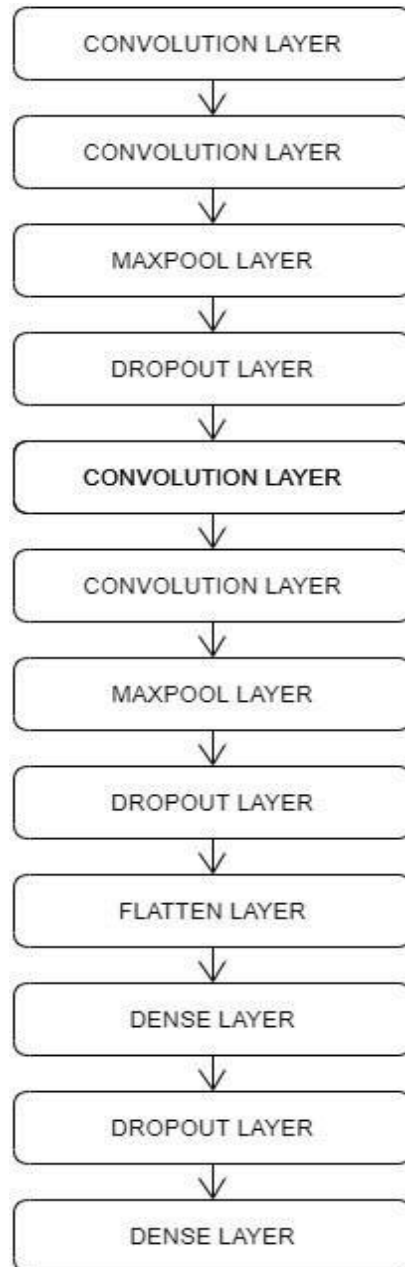


Fig 14 Model architect

A dataset is a collection of data. Most commonly a dataset corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable, and each row corresponds to a given member of the dataset in question. The data set lists values for each of the variables such as the height or weight of an object for each member in the dataset. A data set is organized into some type of data structure. In a database, for example, a data set might contain a collection of business data (names, salaries, contact information, sales figures, and so forth). The database itself can be considered a data set, as can bodies of data within it related to a particular type of information, such as sales data for a particular corporate department.

Datasets play a crucial role in various fields, including data science, machine learning, statistics, and business intelligence. They serve as the foundation for analysis, modeling, and decision-making processes, providing valuable insights into underlying patterns, trends, and relationships within the data.

In the realm of data science, datasets serve as raw material for exploration, manipulation, and transformation into actionable information. Analysts and data scientists leverage datasets to identify correlations, anomalies, and outliers, enabling organizations to make informed decisions and gain a competitive edge. Machine learning algorithms heavily rely on datasets for training, validation, and testing purposes. High-quality datasets facilitate the development and evaluation of predictive models, enabling algorithms to generalize well to unseen data and perform effectively in real-world scenarios.

Moreover, datasets empower researchers and practitioners to conduct empirical studies, validate hypotheses, and contribute to the advancement of knowledge in various domains. By sharing datasets publicly, researchers foster collaboration, reproducibility, and transparency in scientific inquiry, allowing others to build upon existing work and explore new research avenues. In the context of business intelligence and analytics, datasets serve as the backbone of reporting, dashboards, and performance metrics. Organizations leverage datasets to monitor key performance indicators (KPIs), track business trends, and make data-driven decisions to optimize operations and drive growth.

Furthermore, the proliferation of open datasets and data marketplaces democratizes access to valuable information, enabling individuals and organizations of all sizes to harness the power of data for innovation and societal impact. Whether it's analyzing social trends, predicting

market behavior, or understanding public health dynamics, datasets empower stakeholders to derive meaningful insights and drive positive change.

In essence, datasets represent more than just collections of data; they are invaluable assets that fuel discovery, innovation, and progress across diverse disciplines. As the volume and complexity of data continue to grow, the importance of high-quality datasets and robust data management practices becomes increasingly evident, laying the foundation for a data-driven future.

The term data set originated with IBM, where its meaning was similar to that of file. In an IBM mainframe operating system, a data set is a named collection of data that contains individual data units organized (formatted) in a specific, IBM-prescribed way and accessed by a specific access method based on the data set organization. Types of data set organizations include sequential, relative sequential, indexed sequential, and partitioned. Access methods include the Virtual Sequential Access Method (VSAM) and the Indexed Sequential Access Method (ISAM).

DATASET DESCRIPTION

This dataset contains 12,500 augmented images of various types of white bloodcells (JPEG) with accompanying cell type labels (CSV). There are approximately 3,000 images for each of 4 different cell types grouped into 4 different folders (according to cell type). The cell types are Eosinophil, Lymphocyte, Monocyte, and Neutrophil. White blood cell (WBCs) counting is an important indicator of health and is important for many diagnostic tests. Currently, doctors utilize expensive automated counters like flow cytometers, or manually count blood cells on a microscope slide.

Therefore, providing an automated way to detect and count WBCs would be advantageous. Detecting the WBCs is the first step for achieving this goal. A dataset is a collection of data. Most commonly a dataset corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable, and each row corresponds to a given member of the dataset in question. The data set lists values for each of the variables such as the height or weight of an object for each member in the dataset.



4.7 DATA PREPROCESSING

Data pre-processing is a data mining technique that is used to transform the raw data into a useful and efficient format. It is the step before applying Machine Learning Algorithms. It transforms the original data into a suitable shape to be used by a particular algorithm. Data pre-processing includes different tasks as data cleaning, feature selection, and data transformation. This data is usually not necessary or helpful when it comes to analyzing data because it may hinder the process or provide inaccurate results. There are several methods for cleaning data depending on how it is stored along with the answers being sought missing codes, and identifying duplicate data points.

Currently, doctors utilize expensive automated counters like flow cytometers, or manually count blood cells on a microscope slide. Therefore, providing an automated way to detect and count WBCs would be advantageous. Detecting the WBCs is the first step for achieving this goal. A dataset is a collection of data. Most commonly a dataset corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable, and each row corresponds to a given member of the dataset in question. The data set lists values for each of the variables such as the height or weight of an object for each member in the dataset.

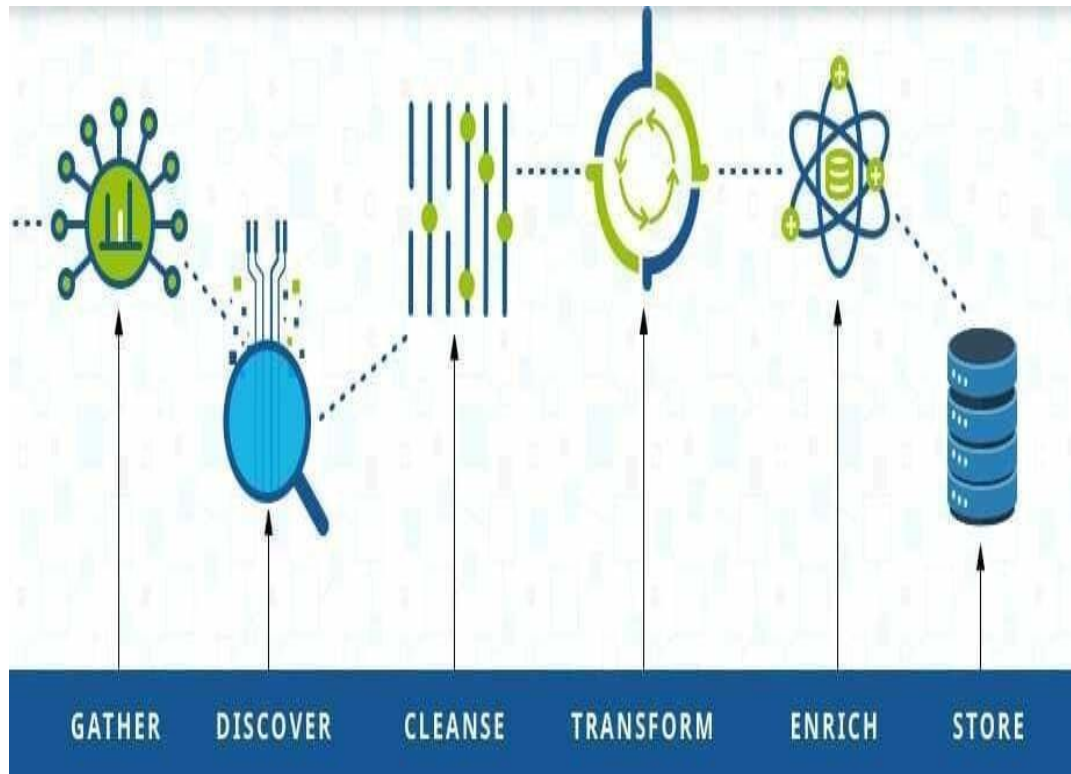


Fig 15 Data Pre-processing

Data Cleaning:

Data cleaning is the process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted. This data is usually not necessary or helpful when it comes to analyzing data because it may hinder the process or provide inaccurate results.

There are several methods for cleaning data depending on how it is stored along with the answers being sought. Data cleaning is not simply about erasing information to make space for new data, but rather finding a way to maximize a data set's accuracy without necessarily deleting information. For one, data cleaning includes more actions than removing data, such as fixing spelling and syntax errors, standardizing data sets, and correcting mistakes such as empty fields, missing codes, and identifying duplicate data points. Data cleaning is considered a foundational element of the data science basics, as it plays an important role in the analytical process and uncovering reliable answers. Data cleaning is an essential step in the data analysis pipeline, ensuring that datasets are accurate, reliable, and ready for further analysis. Beyond simply removing or modifying problematic data, data cleaning involves a range of actions aimed at enhancing the quality and integrity of the dataset.

One aspect of data cleaning involves addressing issues related to data accuracy and completeness. This may include identifying and correcting errors such as misspellings, syntax errors, or inconsistencies in data entry. Standardizing data formats and units across the dataset can also improve accuracy and facilitate comparisons between different data points.

Another key aspect of data cleaning is dealing with missing or incomplete data. This may involve imputing missing values using statistical techniques or domain knowledge to preserve the integrity of the dataset while minimizing bias. Additionally, data cleaning may involve identifying and handling outliers or anomalies that could skew the analysis results.

Data cleaning also addresses issues related to data duplication and redundancy. Identifying and removing duplicate records or entries ensures that each data point is unique and contributes meaningfully to the analysis. This helps avoid double-counting and ensures the accuracy of statistical calculations and insights derived from the data. Moreover, data cleaning is not a one-time process but rather an iterative and ongoing effort throughout the data lifecycle. As new data is collected or updated, it is important to continuously monitor and maintain data quality to prevent the accumulation of errors or inconsistencies over time.

Overall, data cleaning is a foundational aspect of data science and analytics, laying the groundwork for accurate and reliable insights. By systematically addressing issues related to data accuracy, completeness, consistency, and duplication, data cleaning enables analysts to uncover meaningful patterns and trends, leading to more informed decision-making and actionable insights.

There are several methods for cleaning data depending on how it is stored along with the answers being sought. Data cleaning is not simply about erasing information to make space for new data, but rather finding a way to maximize a data set's accuracy without necessarily deleting information. For one, data cleaning includes more actions than removing data, such as fixing spelling and syntax errors, standardizing data sets, and correcting mistakes such as empty fields, missing codes, and identifying duplicate data points. Data cleaning is considered a foundational element of the data science basics, as it plays an important role in the analytical process and uncovering reliable answers.

Data Cleansing

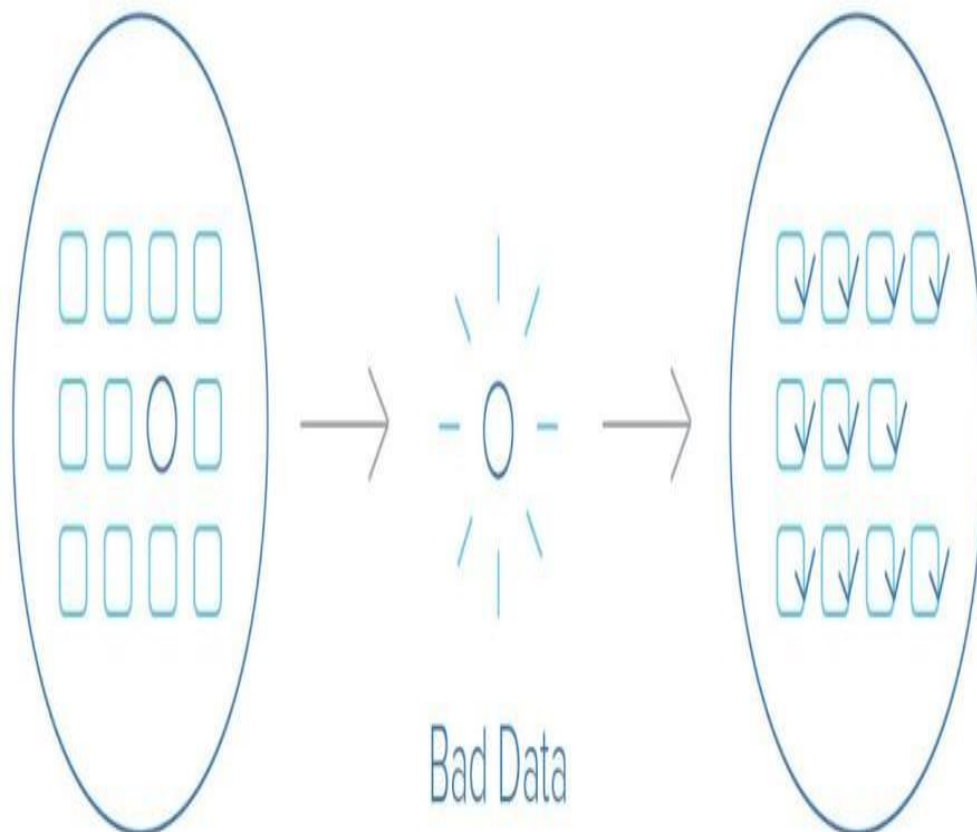


Fig. 16 Data Cleaning

Data cleaning is not simply about erasing information to make space for new data, but rather finding a way to maximize a data set's accuracy without necessarily deleting information. For one, data cleaning includes more actions than removing data, such as fixing spelling and syntax errors, standardizing data sets, and correcting mistakes such as empty fields, missing codes, and identifying duplicate data points. Data cleaning is considered a foundational element of the data science basics, as it plays an important role in the analytical process and uncovering reliable answers.

S.No	Task	Description
1	Problem Statement	Detect and classify white blood cells (WBCs) from microscopic images using Convolutional Neural Networks.
2	Dataset Collection	Gather a dataset of microscopic images containing WBCs.
3	Data Preprocessing	Normalize images, resize them to a standard size, and perform data augmentation techniques if necessary.
4	Model Architecture Design	Design a CNN architecture suitable for image classification tasks, considering depth, filters, etc.
5	Model Training	Train the CNN model on the preprocessed dataset using appropriate optimization techniques.
6	Hyperparameter Tuning	Experiment with different hyperparameters such as learning rate, batch size, etc., to optimize model performance.
7	Model Evaluation	Evaluate the trained model on a separate validation set using metrics like accuracy, precision, recall, and F1-score.
8	Error Analysis	Analyze misclassifications and errors to understand model weaknesses and potential improvements.
9	Fine-tuning and Optimization	Fine-tune the model based on error analysis results and optimize for better performance.
10	Deployment	Deploy the trained model for real-world applications, considering factors like latency and resource constraints.
11	User Interface Design and Integration	Design a user-friendly interface for interacting with the deployed model, if applicable.
12	Testing and Validation	Test the deployed model with real-world data and validate its performance in the target environment.
13	Documentation	Document the entire process, including dataset details, model architecture, training process, and deployment instructions.
14	Maintenance and Updates	Regularly update the model and system to adapt to new data and improve performance over time.

Table 2: Data Cleaning

Most importantly, the goal of data cleaning is to create data sets that are standardized and uniform to allow business intelligence and data analytics tools to easily access and find the right data for each query.

Regardless of the type of analysis or data visualizations you need, data cleaning is a vital step to ensure that the answers you generate are accurate. When collecting data from several streams and with manual input from users, information can carry mistakes, be incorrectly inputted, or have gaps.

Data cleaning helps ensure that information always matches the correct fields while making it easier for business intelligence tools to interact with data sets to find information more efficiently. One of the most common data cleaning examples is its application in data warehouses.

Data Visualization:

Data visualization is a graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data. In the world of Big Data, data visualization tools and technologies are essential to analyze massive amounts of information and make data-driven decisions. Our eyes are drawn to colors and patterns. We can quickly identify red from blue, square from the circle. Our culture is visual, including everything from art and advertisements to TV and movies. Data visualization is another form of visual art that grabs our interest and keeps our eyes on the message.

Our innate tendency to be drawn towards colors and patterns makes data visualization an effective communication tool. Just as we instinctively distinguish between colors like red and blue or shapes like squares and circles, we can swiftly comprehend visual representations of data. In a culture saturated with visual stimuli, from art and advertisements to television and cinema, data visualization emerges as yet another medium that captures our attention and conveys messages with impact. Data visualization is essentially storytelling with a purpose. A well-crafted chart or graph can convey insights and narratives that might otherwise be obscured within vast datasets or complex spreadsheets. By presenting information visually, data visualization enables us to grasp trends and outliers at a glance, fostering rapid comprehension and facilitating decision-making processes.

The significance of data visualization becomes particularly evident when faced with the challenge of interpreting extensive spreadsheets or tables of data. While staring at rows and columns of raw data can be overwhelming and obscure trends, a visual representation can elucidate patterns and relationships, enabling us to derive meaningful insights efficiently.

One aspect of data cleaning involves addressing issues related to data accuracy and completeness. This may include identifying and correcting errors such as misspellings, syntax errors, or inconsistencies in data entry. Standardizing data formats and units across the dataset can also improve accuracy and facilitate comparisons between different data points.

Another key aspect of data cleaning is dealing with missing or incomplete data. This may involve imputing missing values using statistical techniques or domain knowledge to preserve the integrity of the dataset while minimizing bias. Additionally, data cleaning may involve identifying and handling outliers or anomalies that could skew the analysis results.

Data cleaning also addresses issues related to data duplication and redundancy. Identifying and removing duplicate records or entries ensures that each data point is unique and contributes meaningfully to the analysis. This helps avoid double-counting and ensures the accuracy of statistical calculations and insights derived from the data. Moreover, data cleaning is not a one-time process but rather an iterative and ongoing effort throughout the data lifecycle. As new data is collected or updated, it is important to continuously monitor and maintain data quality to prevent the accumulation of errors or inconsistencies over time.

When we see a chart, we quickly see trends and outliers. If we can see something, we internalize it quickly. It's storytelling with a purpose. If you've ever stared at a massive spreadsheet of data and couldn't see a trend, you know how much more effective .

Visualization can be. Our eyes are drawn to colors and patterns. We can quickly identify red from blue, square from the circle. Our culture is visual, including everything from art and advertisements to TV and movies. Data visualization is another form of visual art that grabs our interest and keeps our eyes on the message. When we see a chart, we quickly see trends and outliers. If we can see something, we internalize it quickly. It's storytelling with a purpose. If you've ever stared at a massive spreadsheet of data and couldn't see a trend, you know how much more effective a visualization can be.

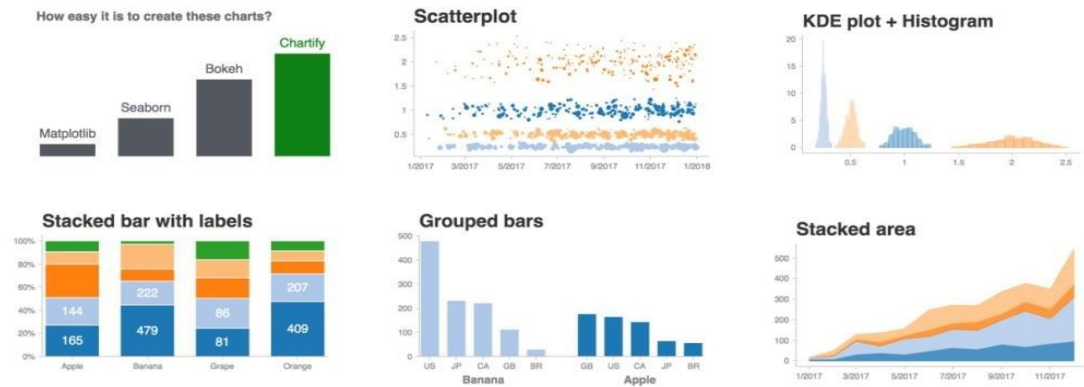


Fig 17 Data Visualization

Data Transformation:

Data transformation is the process of converting data from one format to another, typically from the format of a source system into the required format of a destination system. Data transformation is a component of most data integration and data management tasks, such as data wrangling and data warehousing. One step in the ELT/ETL process, data transformation may be described as either simple or complex, depending on the kinds of changes that must occur to the data before it is delivered to its target destination. The data transformation process can be automated, handled manually, or completed using a combination of the two.

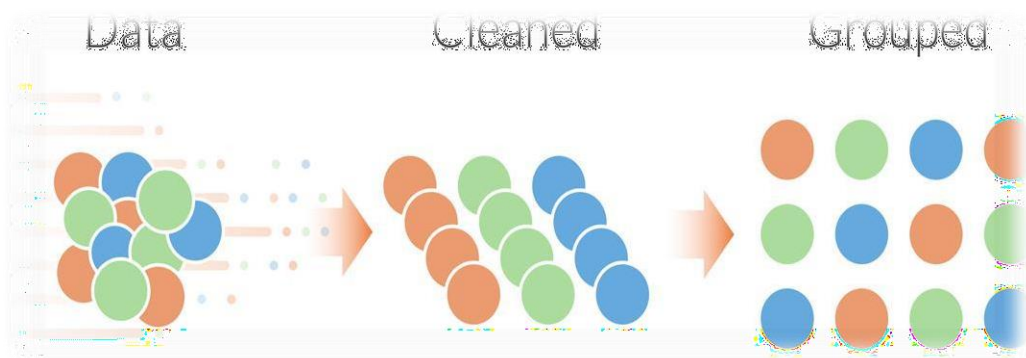


Fig. 18 Data Transformation

Today, the reality of big data means that data transformation is more important for businesses than ever before. An ever-increasing number of programs, applications, and devices continually produce massive volumes of data. And with so much disparate data streaming in

from a variety of sources, data compatibility is always at risk. That's where the data transformation process comes in: it allows companies and organizations to convert data from any source into a format that can be integrated, stored, analyzed, and ultimately mined for actionable business intelligence.

Data splitting

It is standard in ML to split data into training and test sets. The reason for this is very straightforward: if you try and evaluate your system on data you have trained it on, you are doing something unrealistic. The whole point of a machine learning system is to be able to work with unseen data: if you know you are going to see all possible values in your training data, you might as well just use some form of lookup. Separating data into training and testing sets is an important part of evaluating data mining models.

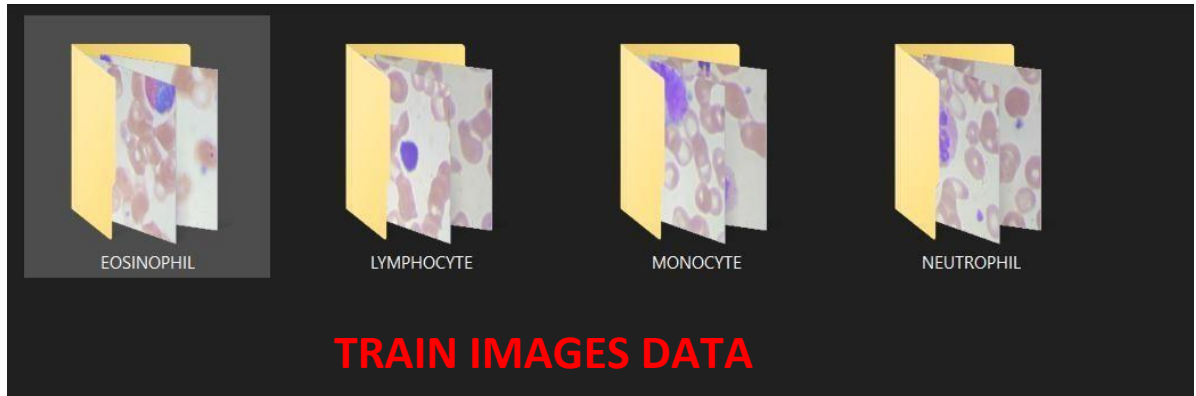
Typically, when you separate a data set into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing. Analysis Services randomly samples the data to help ensure that the testing and training sets are similar. By using similar data for training and testing, you can minimize the effects of data discrepancies and better understand the characteristics of the model. After a model has been processed by using the training set, you test the model by making predictions against the test set. Because the data in the testing set already contains known values for the attribute that you want to predict, it is easy to determine whether the model's guesses are correct.

Training Set:

The observations in the training set to form the experience that the algorithm uses to learn. In supervised learning problems, each observation consists of an observed output variable and one or more observed input variables. Essentially, the training set provides the algorithm with the necessary information to learn patterns, relationships, and dependencies within the data. By exposing the algorithm to a diverse range of input-output pairs, it can iteratively adjust its internal parameters or model structure to minimize errors and improve its predictive accuracy.

The quality and representativeness of the training set are crucial factors influencing the performance of the trained model. A well-curated training set should encompass a broad spectrum of scenarios and variations that the model might encounter during deployment. Additionally, it's essential to ensure that the training set adequately covers the target domain and includes sufficient instances of each class or category for robust learning.

In summary, the training set forms the cornerstone of supervised learning, providing the algorithm with the empirical data necessary to generalize patterns and make accurate predictions or classifications. Its construction and composition significantly impact the efficacy and generalization capability of the trained model.



Test Set:

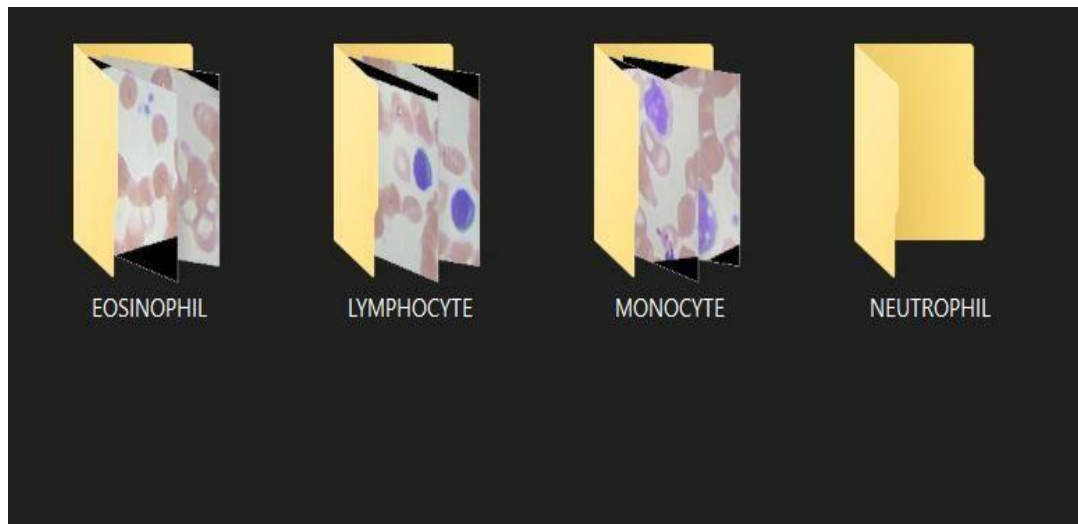
The test set is a set of observations used to evaluate the performance of the model using some performance metrics. No observations from the training set must be included in the test set. If the test set does contain examples from the training set, it will be difficult to assess whether the algorithm has learned to generalize from the training set or has simply memorized it. This segregation is vital to ensure an unbiased evaluation of the model's performance.

Unlike the training set, which is employed to train the model, the test set is exclusively reserved for evaluating how well the model generalizes to new, unseen data. By exposing the model to unseen instances during testing, researchers and practitioners can gauge its ability to make accurate predictions or classifications on data it hasn't encountered before.

The test set serves as a benchmark for assessing various performance metrics, such as accuracy, precision, recall, F1 score, and others, depending on the specific task and objectives. These metrics provide insights into the model's effectiveness in capturing patterns and making predictions, thereby informing decisions regarding model refinement, parameter tuning, or feature selection.

In summary, the test set plays a crucial role in assessing the performance and generalization capacity of machine learning models. Its independence from the training set ensures unbiased evaluation, enabling researchers and practitioners to make informed decisions about model deployment and refinement.

The test set is a set of observations used to evaluate the performance of the model using some performance metrics. No observations from the training set must be included in the test set. If the test set does contain examples from the training set, it will be difficult to assess whether the algorithm has learned to generalize from the training set or has simply memorized it. This



System implementation includes the usage of tools for the development of the project. This project uses Machine Learning technology with Python as programming language and Anaconda distribution and implementation of coding is done in the Jupyter notebook

Python:

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs, and object-oriented approach, aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3.

Anaconda:

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system conda.

The Anaconda distribution includes data-science packages suitable for Windows, Linux, and MacOS. Anaconda distribution comes with 1,500 packages selected from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI).

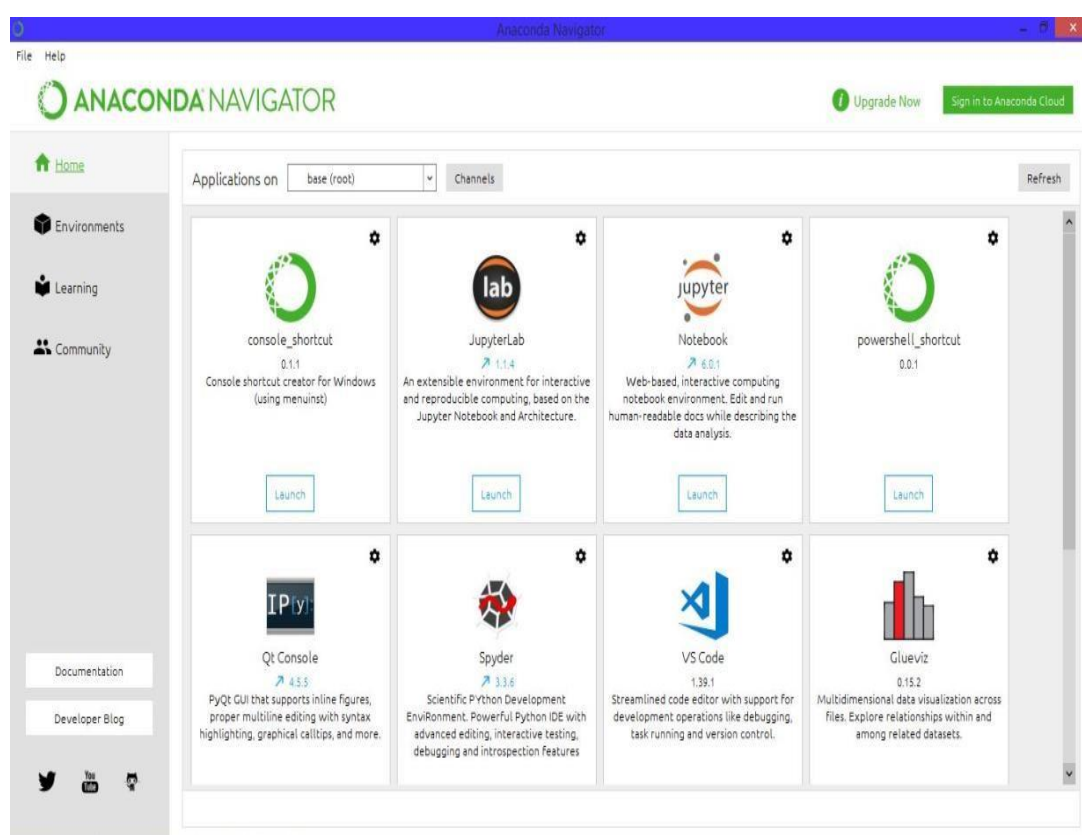


Fig. 19 Anaconda Prompt

Open source packages can be individually installed from the Anaconda repository [8], Anaconda Cloud (anaconda.org), or your own private repository or mirror, using the conda install command. Anaconda Inc compiles and builds the packages available in the Anaconda repository itself, and provides binaries for Windows 32/64 bit, Linux 64 bit and MacOS 64-bit. Anything available on PyPI may be installed into a conda environment using pip, and conda will keep track of what it has installed itself and what pip has installed. Custom packages can be made using the conda build command, and can be shared with others by

uploading them to Anaconda Cloud, PyPI or other repositories. The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, it is possible to create new environments that include any version of Python packaged with conda.

Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

4.8 Jupyter Notebook

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents—which are a type of computational notebook.[8] The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension. Jupyter Notebook can connect to many kernels to allow programming in many languages. By default Jupyter Notebook ships with the IPython kernel. As of the 2.3 release (October 2014), there are currently 49 Jupyter-compatible kernels for many programming languages, including Python, R, Julia and Haskell. The Notebook interface was added to IPython in the 0.12 release (December 2011), renamed to Jupyter notebook in 2015 (IPython 4.0 – Jupyter 1.0). Jupyter Notebook is similar to the notebook interface of other programs such as Maple, Mathematica, and SageMath, a computational interface style that originated with Mathematical in the 1980s.[13] According to The Atlantic, Jupyter interest overtook the popularity of the Mathematical notebook interface in early 2018.

Jupyter Notebook has emerged as a powerful tool for data analysis, scientific research, and collaborative coding projects due to its versatile features and user-friendly interface. With its ability to blend code, visualizations, and explanatory text in a single document, Jupyter Notebook facilitates seamless exploration and communication of ideas within the scientific community. The open-source nature of Jupyter Notebook fosters a vibrant ecosystem of

extensions and plugins, allowing users to customize their environments and enhance productivity. Moreover, its support for multiple programming languages through various kernels enables interdisciplinary collaboration and experimentation across diverse domains.

One of the key advantages of Jupyter Notebook is its reproducibility, as each notebook captures the entire computational process, including code execution, data manipulation, and visualization. This transparency promotes transparency and facilitates the sharing of research findings, enabling others to reproduce and validate results with ease. Furthermore, Jupyter Notebook's integration with version control systems such as Git enables seamless collaboration among team members, facilitating code review, version tracking, and project management. This collaborative workflow promotes knowledge sharing and accelerates the pace of scientific discovery. The adoption of Jupyter Notebook extends beyond academia, with industries ranging from finance and healthcare to technology and engineering leveraging its capabilities for data analysis, machine learning, and model development. Its versatility and accessibility make it a preferred choice for professionals seeking to derive insights from complex datasets and prototype solutions rapidly. In conclusion, Jupyter Notebook stands as a cornerstone of modern computational research, empowering scientists, engineers, and analysts to explore data, develop models, and communicate findings effectively. As the demand for reproducible and transparent research practices continues to grow, Jupyter Notebook remains at the forefront, driving innovation and collaboration in the digital age. Jupyter Notebook offers unparalleled reproducibility by capturing the entire computational process, including code execution, data manipulation, and visualization. This transparency fosters trust and simplifies the sharing of research findings, allowing others to replicate and validate results effortlessly. Its integration with version control systems like Git facilitates seamless collaboration, enabling teams to conduct code reviews, track versions, and manage projects efficiently. This collaborative workflow promotes knowledge dissemination and expedites scientific breakthroughs. Beyond academia, Jupyter Notebook finds extensive use in diverse industries such as finance, healthcare, technology, and engineering. Its versatility makes it a preferred tool for professionals engaged in data analysis, machine learning, and rapid prototyping. With Jupyter Notebook, users can derive insights from complex datasets and develop solutions swiftly, driving innovation across various domains. In conclusion, Jupyter Notebook plays a pivotal role in modern computational research, empowering scientists, engineers, and analysts to explore data, build models, and communicate results effectively.

5. CONCLUSION

Conclusion:

White blood cells are an important component in the human blood. All white blood cells have nuclei, which distinguishes them from the other blood cells and also between white blood cell types and subtypes themselves. In this Project, white blood cell types and their structure are reviewed. An automatic white blood cell classification system, including image acquisition, pre-processing, segmentation and feature extraction is presented. White blood cells are integral components of the human blood, playing a vital role in the immune system's defense against infections and foreign substances. Their distinct characteristic of having nuclei sets them apart from other blood cells, and further differentiation occurs among various types and subtypes of white blood cells.

In this project, a comprehensive review of white blood cell types and their structures is provided, shedding light on the diversity and functions within this crucial component of the circulatory system. Understanding the nuanced characteristics of white blood cells is fundamental in comprehending their role in maintaining the body's health and responding to potential threats.

The automated white blood cell classification system developed in this project represents a significant advancement in medical diagnostics, offering a holistic approach to analyzing blood samples and identifying various white blood cell types with precision. By integrating image acquisition, pre-processing, segmentation, and feature extraction techniques, the system streamlines the analysis process while ensuring robust and accurate results. White blood cells, with their distinctive nuclei and diverse structures, play a pivotal role in the body's immune response, safeguarding against infections and maintaining overall health. The ability to classify and quantify white blood cells automatically not only accelerates diagnostic procedures but also enhances the reliability of clinical assessments, enabling healthcare professionals to make informed decisions promptly. Furthermore, the implementation of automated white blood cell classification holds immense potential in revolutionizing healthcare delivery, particularly in resource-constrained settings where access to skilled laboratory personnel may be limited. By leveraging advanced technologies such as convolutional neural networks, this system empowers healthcare providers with the tools necessary to expedite diagnosis and treatment, ultimately improving patient outcomes and reducing healthcare disparities.

Moving forward, continued research and development in the field of automated blood cell analysis promise to further refine and optimize diagnostic algorithms, enhancing their accuracy and efficiency. Additionally, efforts to integrate such systems into routine clinical practice will require collaboration between researchers, healthcare providers, and regulatory bodies to ensure seamless adoption and adherence to quality standards. White blood cells are integral components of the human blood, playing a vital role in the immune system's defense against infections and foreign substances. Their distinct characteristic of having nuclei sets them apart from other blood cells, and further differentiation occurs among various types and subtypes of white blood cells.

In conclusion, the automated white blood cell classification system presented in this project represents a significant step towards enhancing the efficiency and accuracy of blood smear analysis. By harnessing the power of technology, we can revolutionize the way we diagnose and treat diseases, ultimately advancing the field of medicine and improving patient care on a global scale. The automated white blood cell classification system developed in this project represents a significant advancement in medical diagnostics, offering a holistic approach to analyzing blood samples and identifying various white blood cell types with precision. By integrating image acquisition, pre-processing, segmentation, and feature extraction techniques, the system streamlines the analysis process while ensuring robust and accurate results. White blood cells, with their distinctive nuclei and diverse structures, play a pivotal role in the body's immune response, safeguarding against infections and maintaining overall health. The ability to classify and quantify white blood cells automatically not only accelerates diagnostic procedures but also enhances the reliability of clinical assessments, enabling healthcare professionals to make informed decisions promptly.

5.1 FUTURE SCOPE:

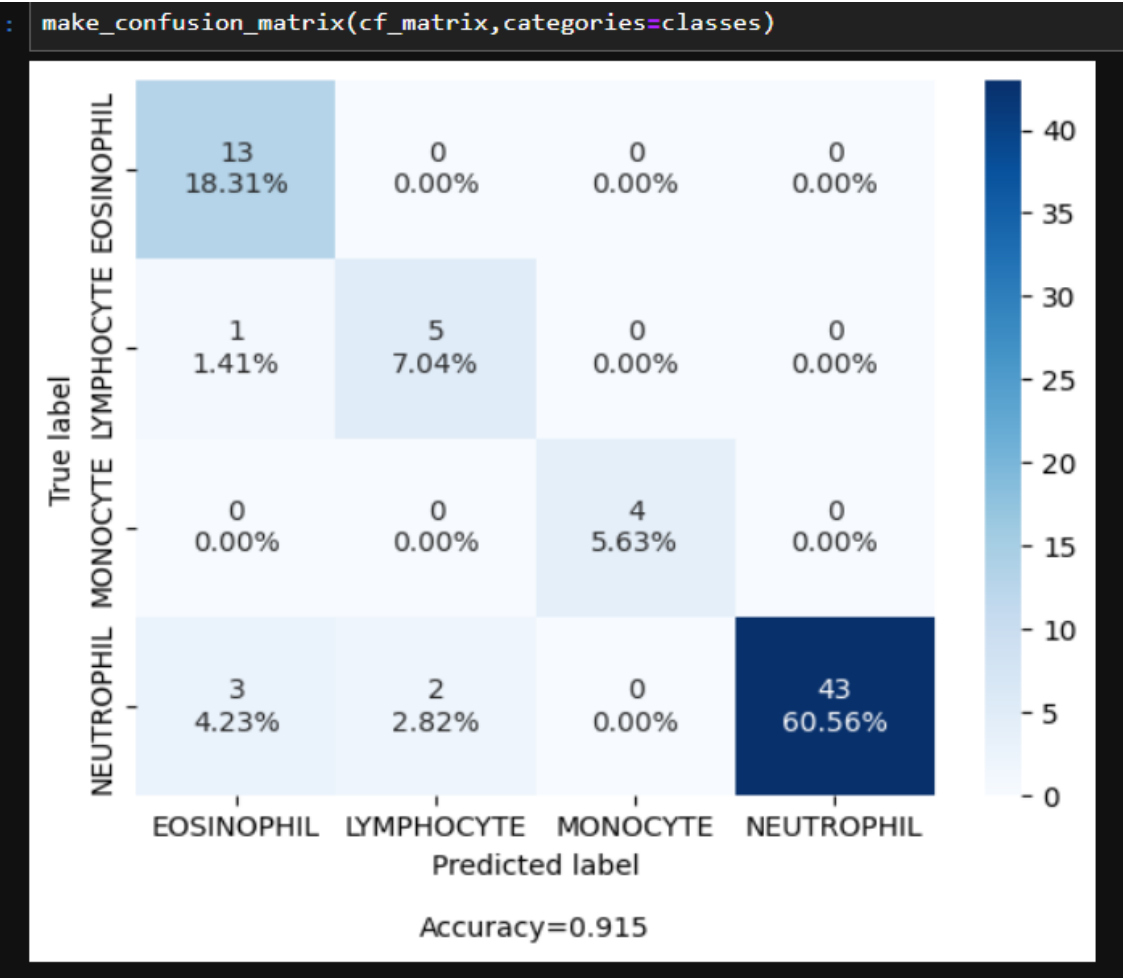
1. **Deep Learning Integration:** By utilizing deep learning methods like convolutional neural networks (CNNs), the system can become more adept at automatically extracting discriminative characteristics from unprocessed visual input.
2. **Interactive User Interface:** Create a system that is easy to use and intuitive to provide smooth communication with medical professionals.

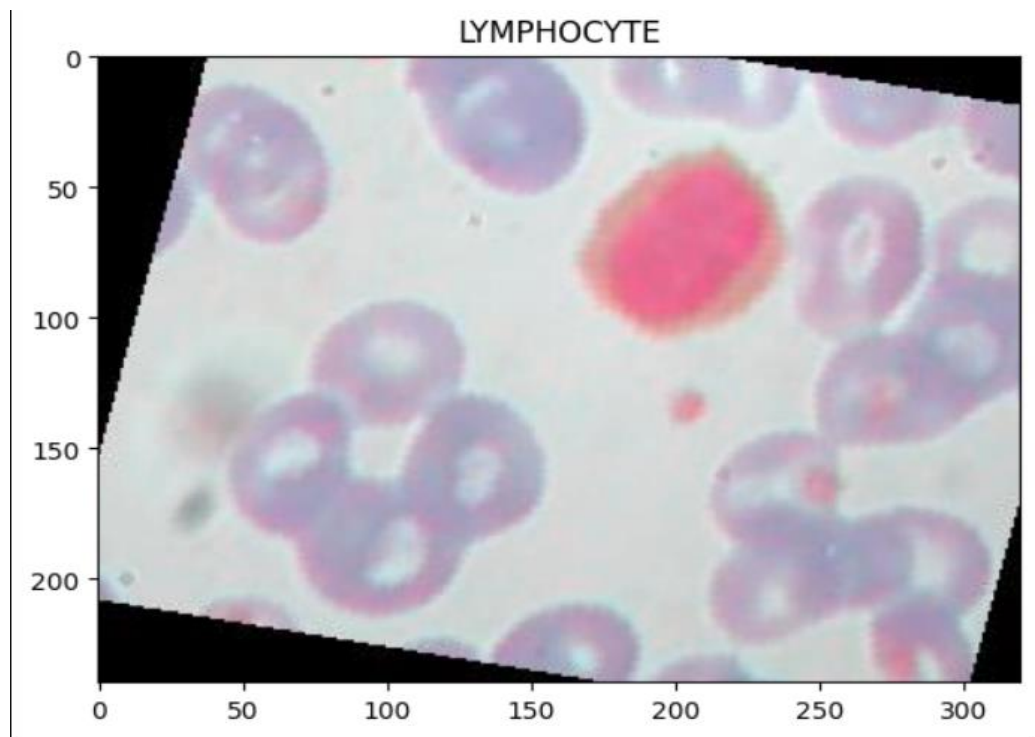
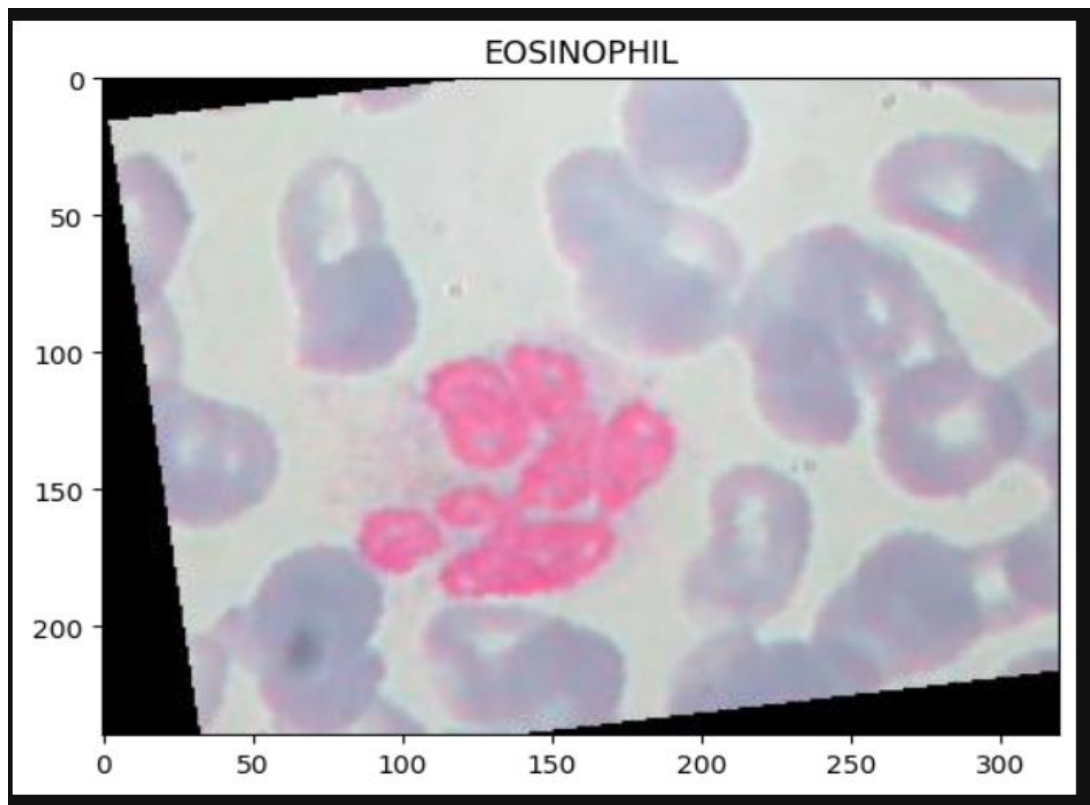
3.To improve usability and adoption in clinical practice, include features like feedback systems, outcome interpretation, and image visualization.

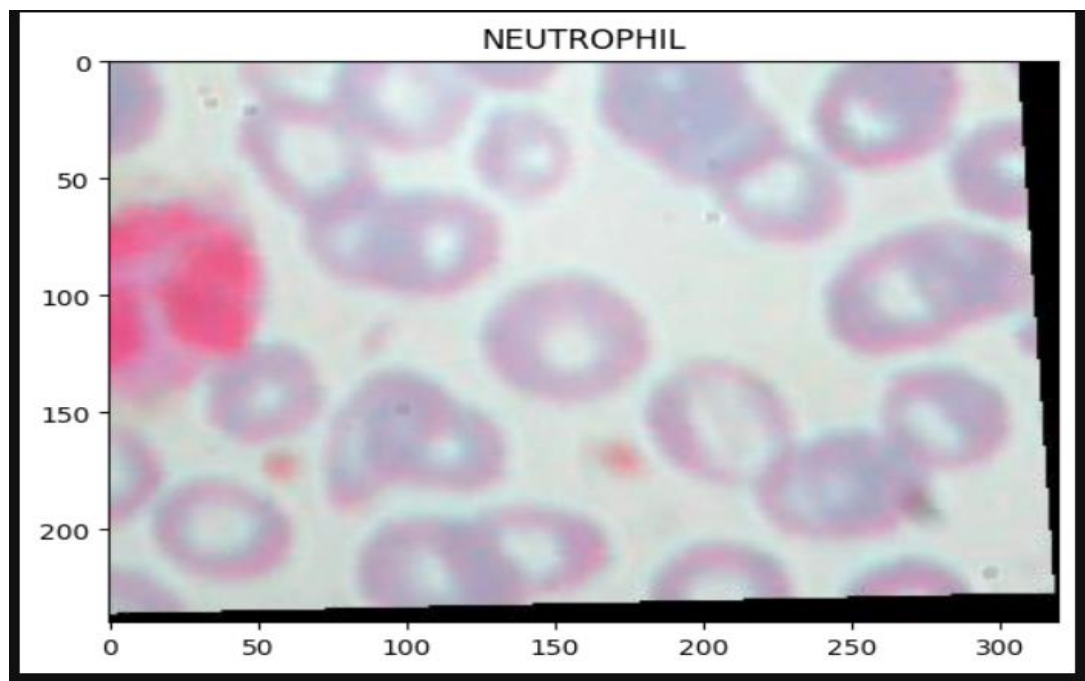
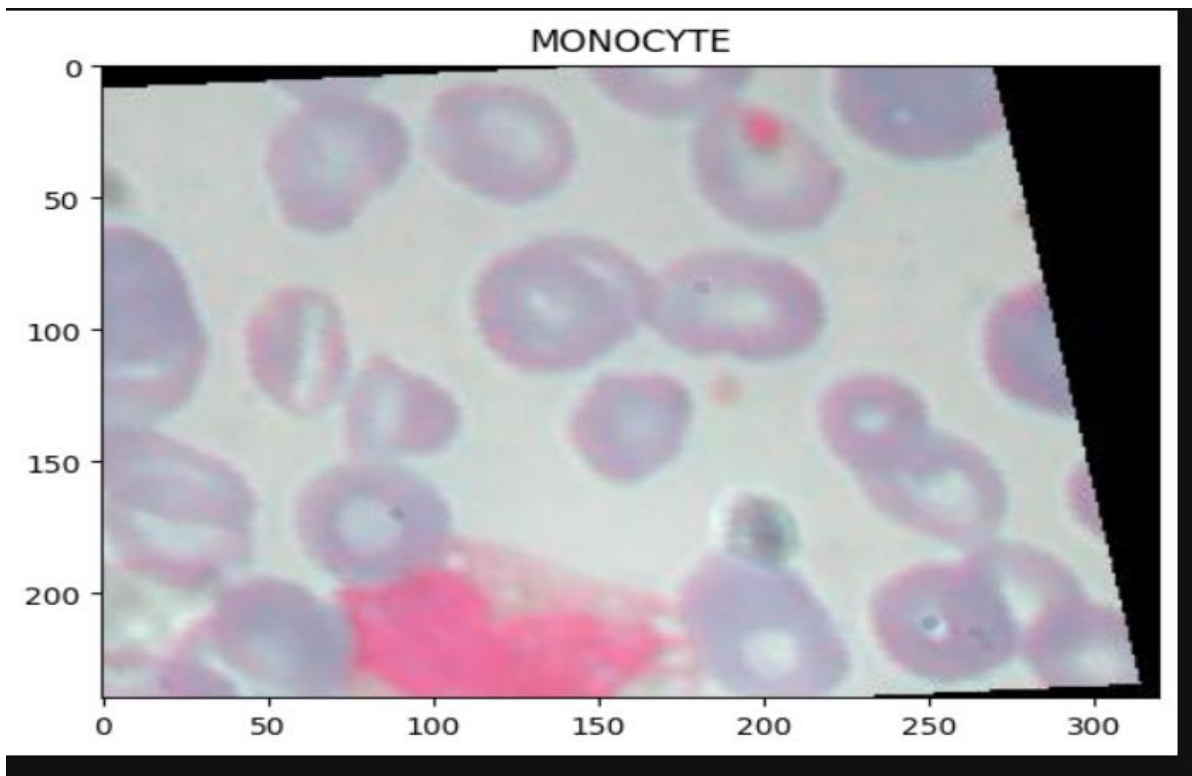
4.Studies on Clinical Validation: To evaluate the system's effectiveness in real-world circumstances, carry out comprehensive validation tests using a variety of patient groups and clinical contexts. Work together with medical facilities and labs to assess the system's clinical usefulness, accuracy, and dependability for identifying a range of illnesses and ailments.

5.Automated Abnormality Detection: Enable the system to identify deviations or anomalies in the morphology of white blood cells that may be signs of underlying medical illnesses, including autoimmune diseases, infections, or hematologic cancers. Create algorithms that highlight data.

Output:







5.2 Summary

Detection of white blood cells using Convolutional Neural Networks (CNNs) represents a cutting-edge approach in medical image analysis, particularly in the field of hematology. CNNs, a type of deep learning model, excel at automatically learning discriminative features from raw image data, making them well-suited for tasks such as cell detection and classification.

The process typically involves several key steps:

- 1.Data Acquisition: High-quality images of blood samples containing white blood cells are collected using microscopy or imaging equipment. These images serve as the input data for training the CNN.
- 2.Data Preprocessing: The images may undergo preprocessing steps such as normalization, resizing, and noise reduction to enhance their quality and ensure consistency across the dataset.
- 3.Annotation: Expert annotations are provided to label the white blood cells in the images. These annotations serve as ground truth labels for training the CNN.
- 4.Model Training: The CNN is trained on a dataset comprising annotated images of white blood cells. During training, the CNN learns to automatically identify features indicative of white blood cells, leveraging the hierarchical representations learned through multiple layers of convolution and pooling.
- 5.Model Evaluation: The trained CNN is evaluated on a separate test dataset to assess its performance in detecting white blood cells. Metrics such as accuracy, precision, recall, and F1 score are computed to quantify the model's performance.
- 6.Deployment: Once the CNN demonstrates satisfactory performance, it can be deployed for automated detection of white blood cells in new, unseen images. The deployed model can streamline the analysis process, aiding in the diagnosis and monitoring of various medical conditions.

CNN-based white blood cell detection systems offer several advantages, including:

High Accuracy: CNNs can achieve high accuracy in detecting white blood cells, often surpassing traditional image processing techniques.

Automation: The automated nature of CNNs reduces the need for manual intervention, saving time and labor in medical diagnostics.

Robustness: CNNs can generalize well to unseen data, making them suitable for handling diverse datasets and real-world clinical scenarios.

Overall, CNN-based white blood cell detection systems represent a promising advancement in medical imaging technology, with the potential to improve the efficiency and accuracy of blood cell analysis in clinical settings.

The application of CNNs in white blood cell analysis holds great promise for improving diagnostic accuracy, efficiency, and scalability in clinical settings. With continued research and development, CNN-based approaches are likely to play an increasingly important role in advancing our understanding and diagnosis of various diseases and conditions related to white blood cells.

REFERENCES

- [1] Khamael AL-Dulaimi, Jasmine Banks, Vinod Chandran, Inmaculada Tomeo-Reyes and Kien Nguyen, —Classification of White Blood Cell Types from Microscope Images: Techniques and Challenges, <https://www.researchgate.net/publication/327931984>
- [2] Carleton HM, Drury RAB, Wallington EA. Carleton's histological technique: Oxford University Press, USA; 1980.
- [3] Clark VL, Kruse JA. Clinical methods: the history, physical, and laboratory examinations. Jama. 1990; 264(21):2808-9.
- [4] 1. Ciresan, D. C., Giusti, A., Gambardella, L. M., & Schmidhuber, J. (2013). "Mitosis detection in breast cancer histology images with deep neural networks." Medical Image Computing and Computer-Assisted Intervention, 16(1), 411-418.
- [5] 2. Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., & Summers, R. M. (2016). "ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3462-3471.
- [6] 3. Xu, J., Luo, X., Wang, G., Gilmore, H., & Madabhushi, A. (2016). "A Deep Convolutional Neural Network for segmenting and classifying epithelial and stromal regions in histopathological images." Neurocomputing, 191, 214-223.
- [7] 4. Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., ... & Sánchez, C. I. (2017). "A survey on deep learning in medical image analysis." Medical Image Analysis, 42, 60-88.
- [8] 5. Sirinukunwattana, K., Raza, S. E. A., Tsang, Y. W., Snead, D. R., Cree, I. A., & Rajpoot, N. M. (2016). "Locality Sensitive Deep Learning for Detection and Classification of Nuclei in Routine Colon Cancer Histology Images." IEEE Transactions on Medical Imaging, 35(5), 1196-1206.

- [9] 6. Bentaieb, A., Hamarneh, G. (2015). "Boundary-driven patch-based segmentation using deep learning for whole slide histopathology images." In Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2015.
- [10]7. Cruz-Roa, A., Basavanhally, A., González F., et al. (2014). "Automatic detection of invasive ductal carcinoma in whole slide images with Convolutional Neural Networks." In SPIE Medical Imaging, International Society for Optics and Photonics.

APPENDIX - A

The appendix for a paper on white blood cell detection and identification using Convolutional Neural Networks (CNNs) serves to provide supplementary information that complements the main content of the paper. It includes details on data preprocessing, model architecture, hyperparameters, training procedures, performance metrics, implementation specifics, computational resources, additional results, and references.

Data preprocessing details encompass the steps taken to prepare white blood cell images for CNN input, such as normalization, resizing, and augmentation techniques. A visual representation of the CNN architecture, along with explanations of each layer's function, helps readers understand the model's design. Hyperparameters, including learning rate, batch size, and optimizer choice, are listed to offer insights into model training settings.

Training details shed light on the specifics of the CNN training process, including the training/validation split ratio, any applied training strategies (e.g., early stopping), and encountered challenges. Performance metrics go beyond those mentioned in the main paper, potentially including confusion matrices, precision-recall curves, and ROC curves.

Implementation details may include snippets of code or pseudocode for critical aspects of CNN implementation, aiding readers in understanding the technical implementation aspects. Computational resources utilized during training and evaluation, such as hardware specifications and software frameworks, are detailed for transparency.

Additional experimental results or analyses that couldn't be accommodated in the main paper due to space constraints find a place in the appendix. Proper formatting according to the target journal or conference guidelines ensures clarity and consistency. Finally, including supplementary figures, such as example images of white blood cells pre- and post-preprocessing, enriches the reader's understanding of the study's methodology and results.

```

import numpy as np
from flask import Flask, request, jsonify, render_template, redirect, url_for
import flask
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras.applications.vgg19 import VGG19
from keras.applications import VGG19
from keras.models import load_model
from keras.preprocessing.image import img_to_array, load_img
import glob
import os
import random
#import pygal
from werkzeug.utils import secure_filename
import plotly
import plotly.graph_objs as go
import numpy as np
import plotly.express as px
import json
#import plotly.plotly as py
from plotly.graph_objs import *

UPLOAD_FOLDER = 'static/uploaded/'
DETAILS_FOL='static/eoss/'
eos=lymph=mono=neutro=0
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['DETAILS_FOL']=DETAILS_FOL

model = tf.keras.applications.VGG19(include_top=False,
weights='imagenet',input_shape=(150,150,3))
loadmodel=load_model(r'model\savemodelvgg19.h5')

```

```

def create_plot(classes,val):

    N = 40
    fig=go.Bar(
        x=classes, # assign x as the dataframe column 'x'
        y=val,
        marker_color='rgb(55, 83, 109)',

    )
    data = [
        go.Bar(
            x=classes, # assign x as the dataframe column 'x'
            y=val,
            marker_color='rgb(55, 83, 109)',

        )
    ]
    layout=Layout({
        'plot_bgcolor': 'rgba(0, 0, 0, 0)',
        'paper_bgcolor': 'rgba(0, 0, 0, 0)',
    })
    fig=Figure(data=data,layout=layout)
    graphJSON = json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)

    return graphJSON


@app.route('/')
def home():
    return render_template('register.html')


@app.route('/registered',methods=["POST","GET"])
def registered():
    global username
    global age

```

```

global gender
username=request.form['name']
age=request.form['age']
gender=request.form['gender']
#print(name," ",age," ",gender)
return render_template('index.html')

```

```

@app.route('/upload_cellimage',methods=["POST","GET"])

```

```

def upload_cellimage():

```

```

    uploaded_files = flask.request.files.getlist("file")
    #eos=lymph=mono=neutro=0
    global eos,lymph,mono,neutro
    classes=['EOSINOPHIL','LYMPHOCYTE','MONOCYTE','NEUTROPHIL']
    my_colors = ['r','g','b','c']

    #print(type_cell)
    for i in uploaded_files:
        #print(i)
        i.save(os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(i.filename)))
        name=i.filename
        #print(name)
        file = os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(i.filename))
        #print(file)
        image=load_img(file,target_size=(150,150))
        image=img_to_array(image)
        image = np.expand_dims(image, axis=0)
        image/=255.
        bt_prediction = model.predict(image)
        singlepred = loadmodel.predict_classes(bt_prediction)
        #print(classes[int(singlepred)])
        if(classes[int(singlepred)]=='EOSINOPHIL'):

```

```

        eos=eos+1

elif(classes[int(singlepred)]=='LYMPHOCYTE'):
    lymph=lymph+1

elif(classes[int(singlepred)]=='MONOCYTE'):
    mono=mono+1

else:
    neutro=neutro+1

val=[eos,lymph,mono,neutro]
print(val)

"""bars=plt.bar(classes,val,color=my_colors,width=0.3)
plt.xlabel('Types of cell')
plt.ylabel('Count')
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x()+0.10, yval + .005, yval)
num=random.randint(0,10000)
plt.savefig('static/graph/{ }'.format(name))"""

text="YOUR RESULT IS SHOWN BELOW! DON'T FORGET TO SHARE YOUR
RESULT WITH YOUR DOCTOR FOR FURTHER EXAMINATION. "

bar = create_plot(classes,val)

count=eos+lymph+mono+neutro
print("COUNT",count)
result="
eos_per=eos/count*100
lymph_per=lymph/count*100
mono_per=mono/count*100
neutro_per=neutro/count*100
if(40<neutro_per<80 and 20<lymph_per<40 and 2<mono_per<8 and 1<eos_per<4 ):

```

```
    result='Normal White Blood Cell Distribution.It is still advised to share your result with  
your doctor for further dignosis.'
```

```
    else:
```

```
        result='White Blood Cell Distribution is not within normal range.It is advised to contact  
your doctor at the earliest for further dignosis. '
```

```
    folder=UPLOAD_FOLDER+'/*'
```

```
    img=glob.glob(folder)
```

```
    for i in img:
```

```
        os.remove(i)
```

```
    return render_template('index.html',  
plot=bar,title=text,username=username,age=age,gender=gender,result=result)
```

```
if __name__ == "__main__":
```

```
    app.config['TEMPLATES_AUTO_RELOAD'] = True
```

```
    app.run(debug=False)
```

```
)
```

```
data = [
```

```
    go.Bar(
```

```
        x=classes, # assign x as the dataframe column 'x'
```

```
        y=val,
```

```
        marker_color='rgb(55, 83, 109)',
```

```
    )
```

```
]
```

```
layout=Layout({
```

```
    'plot_bgcolor': 'rgba(0, 0, 0, 0)',
```

```
    'paper_bgcolor': 'rgba(0, 0, 0, 0)',
```

```
})
```

```
fig=Figure(data=data,layout=layout)
```

```
graphJSON = json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)
```

```

return graphJSON

@app.route('/')
def home():
    return render_template('register.html')

@app.route('/registered',methods=["POST","GET"])
def registered():
    global username
    global age
    global gender
    username=request.form['name']
    age=request.form['age']
    gender=request.form['gender']
    #print(name," ",age," ",gender)
    return render_template('index.html')

@app.route('/upload_cellimage',methods=["POST","GET"])

def upload_cellimage():

    uploaded_files = flask.request.files.getlist("file")
    #eos=lymph=mono=neutro=0
    global eos,lymph,mono,neutro
    classes=['EOSINOPHIL','LYMPHOCYTE','MONOCYTE','NEUTROPHIL']
    my_colors = ['r','g','b','c']

    #print(type_cell)
    for i in uploaded_files:
        #print(i)
        i.save(os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(i.filename)))
        name=i.filename

```

```

#print(name)
file = os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(i.filename))
#print(file)
image=load_img(file,target_size=(150,150))
image=img_to_array(image)
image = np.expand_dims(image, axis=0)
image/=255.
bt_prediction = model.predict(image)
singlepred = loadmodel.predict_classes(bt_prediction)
#print(classes[int(singlepred)])
if(classes[int(singlepred)]=='EOSINOPHIL'):
    eos=eos+1

elif(classes[int(singlepred)]=='LYMPHOCYTE'):
    lymph=lymph+1

elif(classes[int(singlepred)]=='MONOCYTE'):
    mono=mono+1

else:
    neutro=neutro+1

val=[eos,lymph,mono,neutro]
print(val)

"""bars=plt.bar(classes,val,color=my_colors,width=0.3)
plt.xlabel('Types of cell')
plt.ylabel('Count')
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x()+0.10, yval + .005, yval)
num=random.randint(0,10000)
plt.savefig('static/graph/{ }'.format(name))"""

text="YOUR RESULT IS SHOWN BELOW! DON'T FORGET TO SHARE YOUR

```


RESULT WITH YOUR DOCTOR FOR FURTHER EXAMINATION. "

```
bar = create_plot(classes,val)

count=eos+lymph+mono+neutro
print("COUNT",count)
result=""
eos_per=eos/count*100
lymph_per=lymph/count*100
mono_per=mono/count*100
neutro_per=neutro/count*100
if(40<neutro_per<80 and 20<lymph_per<40 and 2<mono_per<8 and 1<eos_per<4 ):
    result='Normal White Blood Cell Distribution.It is still advised to share your result with
your doctor for further dignosis.'
else:
    result='White Blood Cell Distribution is not within normal range.It is advised to contact
your doctor at the earliest for further dignosis. '
folder=UPLOAD_FOLDER+'/*'
img=glob.glob(folder)
for i in img:
    os.remove(i)

return render_template('index.html',
plot=bar,title=text,username=username,age=age,gender=gender,result=result)

if __name__ == "__main__":
    app.config['TEMPLATES_AUTO_RELOAD'] = True
    app.run(debug=False)

)
data = [
    go.Bar(
        x=classes, # assign x as the dataframe column 'x'
        y=val,
```

```

        marker_color='rgb(55, 83, 109)',

    )
]
layout=Layout({
    'plot_bgcolor': 'rgba(0, 0, 0, 0)',
    'paper_bgcolor': 'rgba(0, 0, 0, 0)',
})
fig=Figure(data=data,layout=layout)
graphJSON = json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)

return graphJSON

@app.route('/')
def home():
    return render_template('register.html')

@app.route('/registered',methods=["POST","GET"])
def registered():
    global username
    global age
    global gender
    username=request.form['name']
    age=request.form['age']
    gender=request.form['gender']
    #print(name," ",age," ",gender)
    return render_template('index.html')

@app.route('/upload_cellimage',methods=["POST","GET"])

def upload_cellimage():

```

```

uploaded_files = flask.request.files.getlist("file")
#eos=lymph=mono=neutro=0
global eos,lymph,mono,neutro
classes=['EOSINOPHIL','LYMPHOCYTE','MONOCYTE','NEUTROPHIL']
my_colors = ['r','g','b','c']

#print(type_cell)
for i in uploaded_files:
    #print(i)
    i.save(os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(i.filename)))
    name=i.filename
    #print(name)
    file = os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(i.filename))
    #print(file)
    image=load_img(file,target_size=(150,150))
    image=img_to_array(image)
    image = np.expand_dims(image, axis=0)
    image/=255.
    bt_prediction = model.predict(image)
    singlepred = loadmodel.predict_classes(bt_prediction)
    #print(classes[int(singlepred)])
    if(classes[int(singlepred)]=='EOSINOPHIL'):
        eos=eos+1

    elif(classes[int(singlepred)]=='LYMPHOCYTE'):
        lymph=lymph+1

    elif(classes[int(singlepred)]=='MONOCYTE'):
        mono=mono+1

    else:
        neutro=neutro+1

val=[eos,lymph,mono,neutro]
print(val)

```

```

"""bars=plt.bar(classes,val,color=my_colors,width=0.3)
plt.xlabel('Types of cell')
plt.ylabel('Count')
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x()+0.10, yval + .005, yval)
num=random.randint(0,10000)
plt.savefig('static/graph/{ }'.format(name))"""

text="YOUR RESULT IS SHOWN BELOW! DON'T FORGET TO SHARE YOUR
RESULT WITH YOUR DOCTOR FOR FURTHER EXAMINATION. "

bar = create_plot(classes,val)

count=eos+lymph+mono+neutro
print("COUNT",count)
result=""
eos_per=eos/count*100
lymph_per=lymph/count*100
mono_per=mono/count*100
neutro_per=neutro/count*100
if(40<neutro_per<80 and 20<lymph_per<40 and 2<mono_per<8 and 1<eos_per<4 ):
    result='Normal White Blood Cell Distribution.It is still advised to share your result with
your doctor for further dignosis.'
else:
    result='White Blood Cell Distribution is not within normal range.It is advised to contact
your doctor at the earliest for further dignosis. '
folder=UPLOAD_FOLDER+'/*'
img=glob.glob(folder)
for i in img:
    os.remove(i)

returnrender_template('index.html',
plot=bar,title=text,username=username,age=age,gender=gender,result=result)

```

```

if __name__ == "__main__":
    app.config['TEMPLATES_AUTO_RELOAD'] = True
    app.run(debug=False)

    )
data = [
    go.Bar(
        x=classes, # assign x as the dataframe column 'x'
        y=val,
        marker_color='rgb(55, 83, 109)',

    )
]
layout=Layout({
    'plot_bgcolor': 'rgba(0, 0, 0, 0)',
    'paper_bgcolor': 'rgba(0, 0, 0, 0)',
    })
fig=Figure(data=data,layout=layout)
graphJSON = json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)

return graphJSON

@app.route('/')
def home():
    return render_template('register.html')

@app.route('/registered',methods=["POST","GET"])
def registered():
    global username
    global age
    global gender
    username=request.form['name']

```

```

age=request.form['age']
gender=request.form['gender']
#print(name," ",age," ",gender)
return render_template('index.html')

```

```

@app.route('/upload_cellimage',methods=["POST","GET"])

```

```

def upload_cellimage():

```

```

    uploaded_files = flask.request.files.getlist("file")
    #eos=lymph=mono=neutro=0
    global eos,lymph,mono,neutro
    classes=['EOSINOPHIL','LYMPHOCYTE','MONOCYTE','NEUTROPHIL']
    my_colors = ['r','g','b','c']

    #print(type_cell)
    for i in uploaded_files:
        #print(i)
        i.save(os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(i.filename)))
        name=i.filename
        #print(name)
        file = os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(i.filename))
        #print(file)
        image=load_img(file,target_size=(150,150))
        image=img_to_array(image)
        image = np.expand_dims(image, axis=0)
        image/=255.
        bt_prediction = model.predict(image)
        singlepred = loadmodel.predict_classes(bt_prediction)
        #print(classes[int(singlepred)])
        if(classes[int(singlepred)]=='EOSINOPHIL'):
            eos=eos+1

```

```

elif(classes[int(singlepred)]=='LYMPHOCYTE'):
    lymph=lymph+1

elif(classes[int(singlepred)]=='MONOCYTE'):
    mono=mono+1

else:
    neutro=neutro+1

val=[eos,lymph,mono,neutro]
print(val)

"""bars=plt.bar(classes,val,color=my_colors,width=0.3)
plt.xlabel('Types of cell')
plt.ylabel('Count')
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x()+0.10, yval + .005, yval)
num=random.randint(0,10000)
plt.savefig('static/graph/{ }'.format(name))"""

text="YOUR RESULT IS SHOWN BELOW! DON'T FORGET TO SHARE YOUR
RESULT WITH YOUR DOCTOR FOR FURTHER EXAMINATION. "

bar = create_plot(classes,val)

count=eos+lymph+mono+neutro
print("COUNT",count)
result=""
eos_per=eos/count*100
lymph_per=lymph/count*100
mono_per=mono/count*100
neutro_per=neutro/count*100
if(40<neutro_per<80 and 20<lymph_per<40 and 2<mono_per<8 and 1<eos_per<4 ):
    result='Normal White Blood Cell Distribution.It is still advised to share your result with
your doctor for further diagnosis.'
```

```

else:
    result='White Blood Cell Distribution is not within normal range.It is advised to contact
your doctor at the earliest for further dignosis. '
    folder=UPLOAD_FOLDER+'/*'
    img=glob.glob(folder)
    for i in img:
        os.remove(i)

    returnrender_template('index.html',
plot=bar,title=text,username=username,age=age,gender=gender,result=result)

if __name__ == "__main__":
    app.config['TEMPLATES_AUTO_RELOAD'] = True
    app.run(debug=False)

```


PUBLICATION (PAPER)

IJCRT.ORG

ISSN : 2320-2882



INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Ref No : IJCRT/Vol 12/ Issue 4 / 562

To,
B.SRIRAM

Subject: Publication of paper at International Journal of Creative Research Thoughts.

Dear Author,

With Greetings we are informing you that your paper has been successfully published in the International Journal of Creative Research Thoughts - IJCRT (ISSN: 2320-2882). Thank you very much for your patience and cooperation during the submission of paper to final publication Process. It gives me immense pleasure to send the certificate of publication in our Journal. Following are the details regarding the published paper.

About IJCRT : Scholarly open access journals, Peer-reviewed, and Refereed Journals, Impact factor 7.97 (Calculate by google scholar and Semantic Scholar | AI-Powered Research Tool) , Multidisciplinary, Monthly, Indexing in all major database & Metadata, Citation Generator, Digital Object Identifier(DOI) | UGC Approved Journal No: 49023 (18)

Registration ID : IJCRT_256040

Paper ID : IJCRT2404562

Title of Paper : White Blood Cells Detection by Using Convolutional Neural Network (CNN)

Impact Factor : 7.97 (Calculate by Google Scholar) | License by Creative Common 3.0

Publication Date: 10-April-2024

DOI :

Published in : Volume 12 | Issue 4 | April 2024

Page No : e873-e877

Published URL : http://www.ijcrt.org/viewfull.php?&p_id=IJCRT2404562

Authors : B.SRIRAM, P.MADHAVAN, S.AKHIL KUMAR, S.SABEER KHAN, K.AMUTHA

Notification : UGC Approved Journal No: 49023 (18)

Thank you very much for publishing your article in IJCRT.

Editor In Chief

International Journal of Creative Research Thoughts - IJCRT
(ISSN: 2320-2882)



An International Scholarly, Open Access, Multi-disciplinary, Monthly, Indexing in all major database & Metadata, Citation Generator

Website: www.ijcrt.org | Email: editor@ijcrt.org



INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | ISSN: 2320 - 2882

An International Open Access, Peer-reviewed, Refereed Journal

The Board of
International Journal of Creative Research Thoughts
Is hereby awarding this certificate to

K.AMUTHA

In recognition of the publication of the paper entitled
White Blood Cells Detection by Using Convolutional Neural Network (CNN)

Published In IJCRT (www.ijert.org) & 7.97 Impact Factor by Google Scholar

Volume 12 Issue 4 April 2024 , Date of Publication: 10-April-2024

UGC Approved Journal No: 49025 (18)

PAPER ID : IJCRT2404562

Registration ID : 256040

Scholarly open access journals, Peer-reviewed, and Refereed Journals, Impact factor 7.97 (Calculate by google scholar and Semantic Scholar | AI-Powered Research Tool) , Multidisciplinary, Monthly Journal

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | IJCRT

An International Scholarly, Open Access, Multi-disciplinary, Indexed Journal

Website: www.ijcrt.org | Email id: editor@ijcrt.org | ESTD: 2013



EDITOR IN CHIEF



INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | ISSN: 2320 - 2882

An International Open Access, Peer-reviewed, Refereed Journal

The Board of
International Journal of Creative Research Thoughts
Is hereby awarding this certificate to

B.SRIRAM

In recognition of the publication of the paper entitled
White Blood Cells Detection by Using Convolutional Neural Network (CNN)

Published In IJCRT (www.ijert.org) & 7.97 Impact Factor by Google Scholar

Volume 12 Issue 4 April 2024 , Date of Publication: 10-April-2024

UGC Approved Journal No: 49025 (18)

PAPER ID : IJCRT2404562

Registration ID : 256040

Scholarly open access journals, Peer-reviewed, and Refereed Journals, Impact factor 7.97 (Calculate by google scholar and Semantic Scholar | AI-Powered Research Tool) , Multidisciplinary, Monthly Journal

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | IJCRT

An International Scholarly, Open Access, Multi-disciplinary, Indexed Journal

Website: www.ijcrt.org | Email id: editor@ijcrt.org | ESTD: 2013



EDITOR IN CHIEF



INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | ISSN: 2320 - 2882

An International Open Access, Peer-reviewed, Refereed Journal

The Board of

International Journal of Creative Research Thoughts

Is hereby awarding this certificate to

P.MADHAVAN

In recognition of the publication of the paper entitled

White Blood Cells Detection by Using Convolutional Neural Network (CNN)

Published In IJCRT (www.ijert.org) & 7.97 Impact Factor by Google Scholar

Volume 12 Issue 4 April 2024 , Date of Publication: 10-April-2024

UGC Approved Journal No: 49025 (18)

PAPER ID : IJCRT2404562

Registration ID : 256040

Scholarly open access journals, Peer-reviewed, and Refereed Journals, Impact factor 7.97 (Calculate by google scholar and Semantic Scholar | AI-Powered Research Tool) , Multidisciplinary, Monthly Journal

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | IJCRT

An International Scholarly, Open Access, Multi-disciplinary, Indexed Journal

Website: www.ijert.org | Email id: editor@ijert.org | ESTD: 2013




EDITOR IN CHIEF



INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | ISSN: 2320 - 2882

An International Open Access, Peer-reviewed, Refereed Journal

The Board of

International Journal of Creative Research Thoughts

Is hereby awarding this certificate to

S.AKHIL KUMAR

In recognition of the publication of the paper entitled

White Blood Cells Detection by Using Convolutional Neural Network (CNN)

Published In IJCRT (www.ijert.org) & 7.97 Impact Factor by Google Scholar

Volume 12 Issue 4 April 2024 , Date of Publication: 10-April-2024

UGC Approved Journal No: 49025 (18)

PAPER ID : IJCRT2404562

Registration ID : 256040

Scholarly open access journals, Peer-reviewed, and Refereed Journals, Impact factor 7.97 (Calculate by google scholar and Semantic Scholar | AI-Powered Research Tool) , Multidisciplinary, Monthly Journal

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | IJCRT

An International Scholarly, Open Access, Multi-disciplinary, Indexed Journal

Website: www.ijert.org | Email id: editor@ijert.org | ESTD: 2013




EDITOR IN CHIEF



INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | ISSN: 2320 - 2882

An International Open Access, Peer-reviewed, Refereed Journal

The Board of
International Journal of Creative Research Thoughts
Is hereby awarding this certificate to

S.SABEER KHAN

In recognition of the publication of the paper entitled
White Blood Cells Detection by Using Convolutional Neural Network (CNN)

Published in IJCRT (www.ijert.org) & 7.97 Impact Factor by Google Scholar

Volume 12 Issue 4 April 2024 , Date of Publication: 10-April-2024

UGC Approved Journal No: 49025 (18)

PAPER ID : IJCRT2404562

Registration ID : 256040

Scholarly open access journals, Peer-reviewed, and Refereed Journals, Impact factor 7.97 (Calculate by google scholar and Semantic Scholar | AI-Powered Research Tool) , Multidisciplinary, Monthly Journal

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS | IJCRT
An International Scholarly, Open Access, Multi-disciplinary, Indexed Journal

Website: www.ijert.org | Email id: editor@ijert.org | ESTD: 2013




EDITOR IN CHIEF