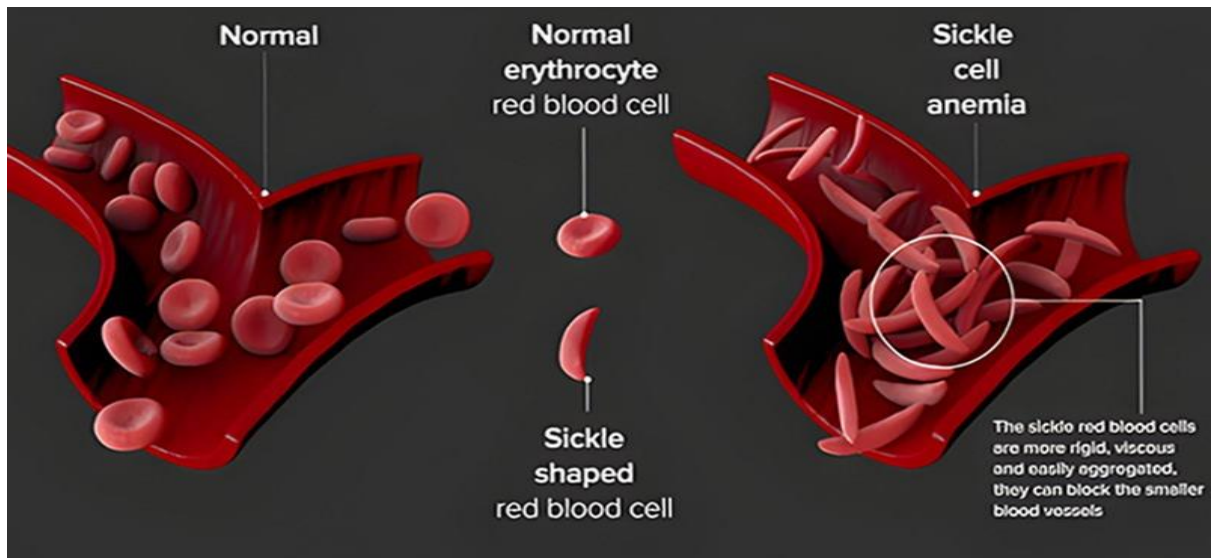


## 1. INTRODUCTION

In the grand orchestra of human biology, even minimal errors in the genetic score can disrupt a life's symphony. One powerful example is Sickle Cell Anemia (SCA) a disease with origins at a single base pair in the genome, yet impacts every organ system and every stage of life. What starts as a modification of a single amino acid in the  $\beta$ -globin chain of hemoglobin—valine-replaces glutamic acid—sets off a chain reaction of cellular dysregulation that distorts the form, function, and survival of red blood cells. These normally pliable, round oxygen-packing vessels transform into rigid, sickle-shaped cells that can't move freely through the blood. But this transformation has consequences far greater than microscopic; they are deeply human. Consequently, blood flow is obstructed, resulting in pain, organ damage, and other severe complications [1].

This disorder results from having a single mutation in the gene that encodes hemoglobin, the protein that distributes oxygen throughout the body. In healthy people, red blood cells can squeeze through narrow blood vessels and release oxygen to tissues. But in SCA, the mutated hemoglobin (known as HbS) tends to clump together when oxygen levels fall. This stiffens the red cells and causes them to assume their characteristic sickle shape. These cells are also much more fragile, and fracture easily, leading to constant sousage of healthy red blood cells, the medical term being chronic hemolytic anemia [1].



**Fig. 1.1: Comparison Between Normal and Sickle-Shaped Red Blood Cells [2]**

In Figure 1.1, it gives us a reference to compare healthy red blood cells vs. sickle cells. Normal red blood cells, shown above, are smooth, round, and flexible, so they can pass easily through even the tiniest blood vessels. In comparison, the sickle cells are shown as stiff crescent shapes with pointed ends. These misshapen cells become sticky, both with each other and with the vessel walls, forming clots that block blood flow. This illustration highlights how the unusual shape disrupts normal circulation, causing the repeated episodes of pain and tissue damage that define the disease.

Sickle cell anemia is a major contributor to the burden of suffering, globally but especially in regions of Sub-Saharan Africa, India, the Middle East and Mediterranean areas. In these areas, the frequency of the sickle gene is extremely high because of the evolutionary benefit that it provides: people with a single copy of the mutated gene (carriers) have greater resistance to malaria. The gene survives through generations because this natural selection(process) has ensured that. [3] People who inherit both copies of the gene, one from each parent, have Sickle Cell Anemia. They suffer cycle after cycle of pain, infections, and fatigue, and can develop long-lasting complications including stroke, stunted growth, and damage to vital organs like the heart and lungs [3].

SCA has many dangerous effects — one of the most dangerous is that it reduces oxygen supplies to key areas of the body. When sickle cells stick together, they occlude capillaries, the smallest blood vessels, and deprive tissues of the oxygen they require. This can lead to sudden attacks of severe pain, known as “pain crises,” and over time, repeated crises damage the organs and increase the threat of life-threatening complications. Children born with SCA are considered the most susceptible to infections and strokes, especially in the absence of early therapeutic intervention [3].

It is vital to the management of the disease. In several nations, newborn screening programs test babies for SCA, enabling treatment to start as quickly as possible. This typically consists of vaccinations, antibiotics to prevent infections, pain relief, and routine medical examinations. In cases of severe aplastic anemia, blood transfusions or bone marrow transplants may be indicated [4].

Knowing how red blood cells behave in SCA is crucial information to improve diagnosis and treatment. Research has also shown that red blood cell shape is the only factor that

matters, as studies continue. The cells' internal dimensions, color, texture, and flexibility are also significant to their disease course. Even how these cells gather into clusters when viewed through a microscope can provide doctors with hints about how severe the condition is.

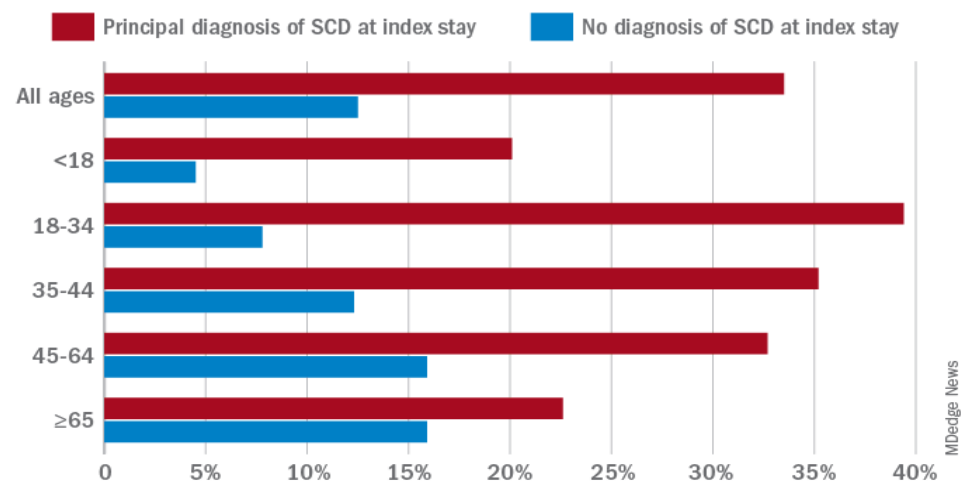
<b>Sickle Cell Disease: A Global Public Health Issue</b>	
<b>COUNTRY</b>	<b>SICKLE CELL BIRTHS/YEAR</b>
Nigeria	91,011
<b>Democratic Republic of the Congo</b>	<b>39,743</b>
Tanzania	11,877
<b>Uganda</b>	<b>10,877</b>
Angola	9,017
<b>Cameroon</b>	<b>7,172</b>
Zambia	6,039
<b>Ghana</b>	<b>5,815</b>
Guinea	5,402
<b>Niger</b>	<b>5,310</b>
Sub-Saharan African Total	242,187
<b>Worldwide Total</b>	<b>305,773</b>
<a href="http://www.MyThreeSicklers.org">www.MyThreeSicklers.org</a>	

**Fig. 1.2 Global prevalence of SCA.[5]**

The above Figure 1.2 shows the birth prevalence of Sickle Cell Disease (SCD) on a map. It demonstrates that Sub-Saharan Africa has the highest concentration of births with SCD and some regions have  $\geq 15$ –20 cases per 1000 births. Countries such as Nigeria, the Democratic Republic of Congo, and India carry the bulk of the disease burden. This color gradient represents differing incidence rates globally, highlighting the necessary implementation of targeted newborn screening as well as public health policy and early intervention programs in high-risk areas.

SCA brings along emotional and social challenges, in addition to physical ones. Patients with the disease have frequent hospitalizations and limited mobility and may have interruptions in education or work. Pain episodes can occur suddenly and last a few hours or days. Over time, this can cause anxiety, depression, and decreased quality of life. This is often an increasing burden on families, as care and medical expenses can be high, particularly in low-income areas where resources [6] are scarce.

### 30-day readmission rates with or without SCD by age, 2016



Note: Based on data from the Nationwide Readmissions Database.  
Source: Agency for Healthcare Research and Quality

**Fig 1.3. Readmission rates at 30 days for SCA by age group (2016). [7]**

Figure 1.3 is a bar graph depicting 30-day hospital readmission rates in those with SCA stratified by age group. Readmission rates were highest in young adults aged 18–30 years, coinciding with a critical transition period during which many patients transfer from pediatric to adult healthcare systems. The admission rate decreases a bit in older cohorts but is still substantial. This statistic highlights the critical need for ongoing disease management, support systems, and access to specialist care — particularly during adolescence and young adulthood when complications often escalate.

In recent years, patient awareness and willingness to seek care for SCA have been a focus of public health organizations. At the community level, screening programs, improved blood testing methods and early education have all contributed to improving identification and treatment of the condition at an early stage. Simultaneously, international parts are working to increase access to curative treatments (typically via gene therapy and/or bone marrow transplant), but these approaches still many patients access ligaments costs and complexity remain.

### Inheritance and Genetics

The below image from the MD Anderson Cancer Center is a beautiful representation of Mendelian inheritance in sickle cell disease. It depicts a couple both carriers of sickle cell

trait (HbA/HbS), who are not affected by the disease but carry one sickle cell gene. If both parents pass on the sickle gene to their child, their child will inherit the genotype HbS/HbS, which means he/she will suffer from sickle cell disease.

The probability breakdown is shown in the image for both parents having the sickle cell trait (HbA/HbS):

## Inheritance of sickle-cell disease

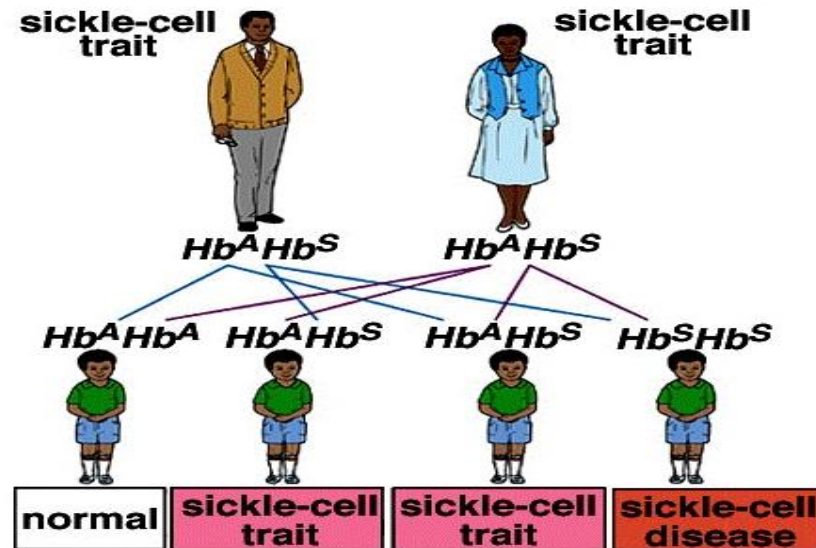


Fig 1.4. Inheritance of Sickle Cell Disease [8]

- 25% – (HbA/HbA) – Child Completely normal, no sickling.
- 50% (HbA/HbS) – child has sickle cell trait (carrier) like the parents
- 25% chance (HbS/HbS) – The child has full out sickle cell anemia.

Figure 1.4 is notable because of genetic counseling, especially in high-prevalence sickle cell trait populations. Knowing can help the so-called affected families prepare and better manage the condition, often even before birth.

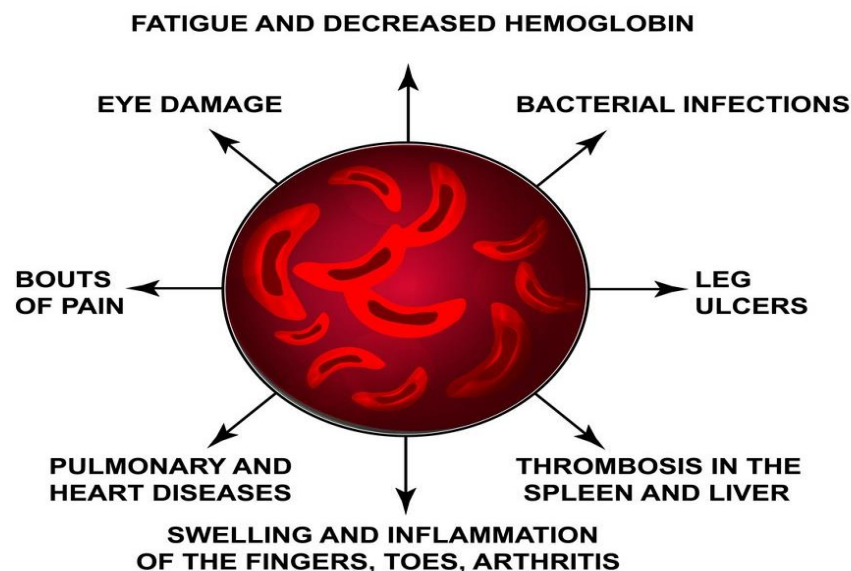
### Symptoms and Genetic Inheritance of SCA

Below Figure 1.5 visually summarizes the variety of complications that sickle cell anemia can cause. This disease can't be explained in terms of one organ system it has consequences for multiple organs and systems based on impaired blood flow and oxygen delivery.”

- **Fatigue and Low Hemoglobin:** Since sickled cells age rapidly and die, the body is left with anemia, and a deficiency of healthy red blood cells. This causes fatigue and chronic tiredness, weakness, and shortness of breath.

- **Bacterial Infections:** Patients with sickle cell anemia are more susceptible to infections, particularly in childhood, because their spleen (which is a critical immune organ) may become damaged as time passes.
- **Leg Ulcers:** When blood flow is restricted and not enough oxygen reaches the tissues, we may experience leg ulcers, open sores, that will take time to heal.
- **Thrombosis in The Spleen and Liver:** When blood vessels get blocked, it may cause tissue death or even damage organs like the spleen and liver.
- **Swelling and inflammation of Fingers, Toes, and Joints:** This painful condition (called dactylitis) often appears first in infants, leading to swelling in small joints.
- **Heart and lung diseases:** repeated blockages can put stress on the heart and lungs, occasionally leading to pulmonary hypertension, heart failure, or acute chest syndrome.
- **Bouts of Pain:** Called sickle cell crises, these episodes of blocked blood vessels can result in severe intractable pain that often leads to hospitalization.
- **Eye Damage:** Blockages in some of the small blood vessels in the eyes can cause vision problems or, over time, complete blindness.

## **SYMPTOMS OF SICKLE CELL ANEMIA**



**Fig 1.5. Symptoms of Sickle Cell Anemia [9]**

This symptom profile underscores the devastation that the disease can wreak if it is not carefully managed. Treatments such as hydroxyurea, blood transfusions, and pain medications relieve symptoms, but a bone marrow or stem cell transplant is the only known cure and is not always possible.

Not only medical professionals but also the general public need to know the symptoms and genetic causes of sickle cell anemia. It gives people the information they need to make informed choices over marriage, reproduction, and early interventions. In general, with greater awareness, routine screenings, and effective management, most who suffer from sickle cell anemia can live longer and healthier lives.

And for basic care—pain management, antibiotics and hydration—to make a life-saving difference, in places with limited access to specialized care [10]. The increasing recognition of the genetic basis of SCA has also made prevention possible. Genetic counseling can educate families about the likelihood of passing the disease on to children, and prenatal testing enables early diagnosis. The above-mentioned steps, combined with community involvement and government-sanctioned health care services, create a sound, sustainable approach to alleviate the impact of Sickle Cell Anemia worldwide.

Based on the previous background knowledge of Sickle Cell Anemia ( SCA ), its genetic base, its effects on the morphology of red blood cells in their patients and their systemic relevance, not only locally to their patients but also globally, this study seeks to apply deep learning techniques to the collaborative solution of problems based on the automatic identification of their anemia as a result of the automatic analysis of images that show the cells in their patients. As a well-known RBC abnormality, sickled cells have identifiable visual features (deformed shape, single/re-fractured cell rigidity, aggregation, etc.) thus, deep learning-based models tend to recognize them with high precision. This is accomplished by segregating individual red blood cells and extracting information on relevant visual features including texture, edge sharpness, internal density, concepts of density, and color variations. Such features are crucial for differentiating between sickled and therefore potentially dangerous cells, and healthy ones, as well as detecting early signs of anemia.

The fast development of deep mastering, especially the use of transformer-based methods, allows for the modeling of detailed spatial and contextual relations within microscopic images to increase classification accuracy and robustness. Dimensionality reduction techniques are also incorporated within this methodology to enhance the performance and manage computational complexity. Interpretability is a main concern — visual heatmaps, saliency maps and feature importance plots are generated to ensure the model is interpretable and clinically relevant. This helps not only build the trust of the medical professionals in the system but also have the expert feedback play a role in iteratively improving the accuracy of the diagnostic system.

The system architecture is independent of the diversity of imaging conditions and healthcare setup that needs to be encountered allowing itself to be sufficiently generalized for deployment in low-resource settings where expert hematological analysis may not be practical. Biomed-ML: Integration of bio-medical expertise with the strengths of machine learning, this has matured into a project that already contributes to more efficient, accurate and transformative SCA diagnostics.



## 2. LITERATURE REVIEW

Pauling L et al. [11] first defined the sickle cell anemia as a "molecular disease", showing by electrophoresis that hemoglobin from the sickle cell patients is not identical to normal hemoglobin. They discovered that sickle cell hemoglobin's charge is different than normal hemoglobin, resulting in polymerization in low-oxygen conditions, and subsequently causing deformation of the erythrocytes. Their work laid the foundation for the genetics of the disease and the connection between protein structure and pathology.

Jennifer SS et al. [12] proposed a transfer deep learning method to Sickle Cell disease (SCD) classification using ResNet-50, AlexNet, MobileNetV2, VGG-16, and VGG-19. Model performance was improved with advanced image augmentation, ablation experiments, and hyperparameter tuning. The highest scores were achieved by a ResNet-50 model with 100% precision and recall for all shapes, and a MobileNetV2 with SVC combination model reached 99.53% accuracy. They managed to improve SCD detection using deep learning, as their research beat all classical ML methods.

Dada EG et al. [13] investigated deep learning methods for sickle cell disease detection from peripheral blood image samples. Their research classified using Plain Convolution Neural Networks (PCNN) with 15 and 48 layers, used Data Augmentation Plain Network (DAPN-48), VGG19, and ResNet-50. Sensitivity and balanced accuracy of the experiments indicated that PCNN-15 and DAPN-48 obtained 99-100% (both) while outperforming PCNN-48. Our study underscores that a combined deep learning-based approach of CNN integrated with data augmentation improves detection accuracy, thereby making deep learning a robust technique for SCD diagnosis.

Alzubaidi L et al. [14] conducted deep CNN for classification of red blood cells to normal, sickle cell anemia and others. An ECOC classifier on top of CNN model was utilized to enhance the accuracy. Their method was proposed to achieve robustness to variations in cell size, color, and shape, reporting a classification accuracy of 92.06%. The researchers hope one day to expand on this work so that they can classify even more types of blood cells.

Amer MA and Ibrahim D. [15] detect sickle cell anemia using the VGG16 model. Their method categorized red blood cells into three classes: round (normal), elongate (sickle

cells), and other blood components. Validation of the model was done using the ERYTHROCYTESIDB dataset which yielded an overall accuracy of 99.4%. The results on sickle cell detection outperform the networks previously reported due to exploring different optimizers and different learning rate schedulers.

Aliyu HA et al. [16] Cross-Education (CE)-based Deep Representation to SCD. A blood smear was used to test AlexNet and SVM for the detection of abnormalities over 14 classes and normal cells. The study demonstrated that AlexNet achieved best accuracy, specificity, and precision in a data set including 182 patients and 11,215 RBC images. This framework to minimize errors in severity evaluation and reduce manual efforts is helpful to diagnose efficiently.

Ekong B et al. [17] performed a Bayesian networks methodology to characterize sickle cell disease in teenagers based on medical parameters namely age, platelet count, median corpuscular hemoglobin concentration and red blood cell count. The probabilistic graphical method implemented by the model could refine the prediction with newly accumulated data, and achieve a 99% accuracy. The researchers categorized teenagers into two groups, one with a milder AS genotype, and the other with a serious health risk, with the SS genotype. Their results show why Bayesian networks may be useful for early and targeted intervention in teenage patients.

Xu M et al. [18] proposed a deep learning approach to sickle cell disease identification using transfer learning. They evaluated GoogLeNet, ResNet18 and ResNet50, and ResNet50 achieved the best accuracy, at 94.90%. Grad-CAM was employed for explainable AI to improve transparency and the interpretability of model predictions. Their approach improves diagnostic efficiency by providing a robust and automated classification of sickle cells, supporting pathologists in detection.

Goswami et al. [19] proposed a model using deep learning for the diagnosis of sickle cell disease, explaining that GoogLeNet, ResNet18, and ResNet50 were used for comparison, and the ResNet50 model was able to classify the data with the highest accuracy of 94.90%. In the context of Grad-CAM in healthcare, Ivanov I, Toleva drew attention to the essential role of explainable AI (XAI), enabling the explanation of predictions and promoting

transparency when using models. This helps increase the accuracy of diagnosis, improving the reliability of AI for medical experts in clinical environments.

Using various algorithms, Shekhar [20] discussed the use of machine learning where sickle cell anemia is predicted and the dosages are classified for the respective patients. Using this, the data was clustered through Fuzzy C-Means clustering which was found to be 99.90% efficient in predicting drug dosages. On a different note, RNNs were not satisfactory on classification tasks and empirical results showed that models like SVM and RFC excelled over RNN.

Ezenwosu O et al. [21] studied depression in sickle cell anemia patients between 7–17 years old, noting a 0.253 correlation with age, and a 0.225 correlation with educational level. There was no effect for sex ( $p = 0.466$ ), SES ( $p = 0.798$ ), or severity of illness ( $p = 0.810$ ). Their research provides support for the implementation of ML-based psychological screening into standardised care to enable early detection.

Yan Q. et al. [22] proposed RBCMatch, a semi-supervised deep learning model based on FixMatch, and obtained 91.2% accuracy (validation) and 87.5% (held-out) accuracy for RBC classification on anemia recovery with 5% labeled data. By analyzing the principal components between RBC morphology with hemoglobin and RBC counts, the model captured the trends within anemia recovery. In summary, this framework aids clinical decision-making by automating RBC classification and faster recovery progression tracking.

Mohammed A. Fadhel et al. [23] recently developed a CNN model that was able to classify RBC into Normal, Abnormal (sickle-shaped), and other blood content with an accuracy of 87.15 %. To achieve real-time classification with lower execution time and power, an acceleration of the model was performed through FPGA (Altera DE2 Cyclone II) implementation. It showed robustness in denoising RBC images and improved the ability in diagnose regardless of hardware platforms used.

Xu M et al. [24] developed a CNN-based approach for the automatic classification of RBCs in SCD, using hierarchical extraction of RBCs, patch-size normalization, and deep CNNs. They constructed an end-to-end model, which allowed for effective classification of

oxygenated and deoxygenated RBCs, showing strong performance without handcrafted features, all while performing analysis of morphological shape.

V. Sharma, A. et al. Proposed K-NN based system for sickle cell anemia and thalassemia detection in microscopic blood smear images [25]. This process includes four steps which are: pre-processing, marker-controlled watershed segmentation, morphological operations, and feature extraction with 80.6% accuracy and 87.6% sensitivity. These techniques quickly process characteristics of erythrocyte abnormalities like sickle cells, dacrocytes or elliptocytes.

Mutar MT et al. [26] used ML models trained on CBC reports to detect blood disorders such as SCD, anemia, leukemia, and lymphoma. Between Decision Tree, XGBoost, and Random Forest classifiers, the latter yielded the best performance. Their method enables early detection, which can decrease mortality and improve treatment. The study shows the diagnostic power of ML using noninvasive blood test results.

Abdul RS et al. [27] implemented ML-based models considering clinical and demographic data for the purpose of more effective early diagnosis of SCA, particularly in high-incidence regions like Saudi Arabia. With the assistance of KNN, XGBoost, and Random Forest, the paper achieved 86.77% accuracy using just five features. Random Forest performed best among the others with 89% precision and 87% recall. The technique enables patient-specific therapy and active intervention in pediatric SCA patients.

Jeevika S et al. [28] compared seven deep learning models to detect SCA from microscopic images, such as DenseNet-201, ResNet-152, Xception, and hybrid models. Model selection was based on image preprocessing and thorough evaluation of metrics. The highest performing model had excellent diagnostic potential. Their automated method is aimed at improving early detection of SCA and clinical efficiency.

Heitzer A.M. et al. [29] investigated fetal hemoglobin (HbF) and their genetic determinants' effect on neurocognition in SCA patients. Using SNP data on three HbF-related QTLs (BCL11A, MYB, HBB cluster), they confirmed that increased HbF is linked with increased IQ in various cohorts. Two BCL11A variants and one MYB SNP significantly mediated the association. The research indicates the potential of HbF in cognitive outcomes and early intervention.

Gabriel et al. [30] have developed an ensemble-based system through the combination of Random Forest and Extra Trees classifiers for Sickle Cell Disease diagnosis leading an F1-score of 90.71% and an SDS-score of 93.33% outperforming previous work. The model focused on feature importance that resulted in less complexity and more interpretability. It generalizes well to the external dataset that validates its applicability to real clinical diagnostic use.

Khan R. et al. [31] also suggested a deep learning-powered InceptionV3 model for non-invasive detection of anemia and prediction of hemoglobin concentration from retinal images. The model, trained on 2,265 South Indian subjects, achieved optimal accuracy and 0.58 g/dL mean absolute error for the prediction of Hb. Anemia retinal vessel pathology like increased tortuosity and reduced density were indicated by GradCAM. The model enables quantitative, non-invasive hematology by retinal imaging.

Subhaga K and TP AD [32] presented the application of Convolutional neural networks (CNNs) for sickle cell anemia detection, highlighting the robustness of CNN in differentiating various cell shapes. IVANOV I, TOLEVA B: A novel image processing method for determining the accuracy of the detection of cells in images reduced the computation time. These techniques highlight the power of deep learning models like VGG16, ResNet and MobileNet in providing higher classification accuracy in medical applications.

Bheem Sen et al. [33] Use a deep CNN with data augmentation--flipping, zooming, and shifting--to sort RBCs related to sickle cell disease. IDB1 dataset microscopic images were sorted into circular, elongated, and others. Pre-trained models, VGG16 and VGG19 as well as ResNet50 and ResNet101 InceptionV3, were evaluated. The classification accuracy was high for all models. It would appear to suggest that a noble service has been accomplished in the realm of SCD diagnosis.

Yeruva S et al. [34] used a multi-layer perceptron (MLP) based classifier to propose an automated methodology for sickle cell anemia detection. This model outperforms the traditional models SVM, KNN, and Decision Trees with highest accuracy in categorizing Normal, Sickle Cells, and Thalassemia. A web application was also constructed

specifically for medical labs to allow faster and more accurate diagnosis with minimal expertise.

Williams R et al. [35] conducted a study comparing the measured resting energy expenditure (REE) with the energy predicted by the Harris-Benedict and WHO formulas in children with sickle cell anaemia.,The standard equations under-predicted REE by 12% or 15%. Two modified equations were developed and tested at lengths verifying meaning: The modified equations closely matched measured values ( $P > 0.6$ ). With these modifications (which could be seen we believe so) one has improved accuracy in assessing energy needs of children who have SCA thoroughly,Editors, TempDataThe modified equations offer precise estimates of energy requirements for pediatric patients with sickle cell anemia.

**Table 2.1: Summary of Sickle Cell Anemia (SCA) Detection Techniques**

Ref No.	Dataset Description	Models / Techniques	Performance Metrics	Limitations Identified
[36]	218 eye images from the public Eyes-defy-anaemia dataset, captured via a low-cost smartphone-mounted macro lens setup to reduce ambient light interference.	Sclera segmentation + vessel extraction + classification using SVM, KNN, Random Forest, AdaBoost (SVM with polynomial kernel performed best).	Sclera Segmentation: Precision 88.53%, Recall 82.53%, F1 Score 84.10% Anemia Classification: F2 Score 86.4% (sclera colour), 83.8% (vessel colour)	1) Small dataset (218 images).2) Single capture setup with limited variability.3) Only basic colour metrics used.4) Needs clinical validation.
[37]	300 patient samples (187 female, 113 male; ages 3–72) from Nilgiris tribal population via NGO NAWA.	Enhanced Whale Optimization for feature selection + Clustering-based Boosted C5.0.	Accuracy: Normal 99.63%, Sickle Cell 98.52%, Thalassemia 96.34% Sensitivity: Normal 97.64%, Sickle Cell 96.32%, Thalassemia 99.48% Specificity: Normal 96.18%, Sickle Cell 98.36%,	1) Only 300 samples.2) Focused on a specific region.3) Not tested on deep learning models.

			Thalassemia 97.28%Error Rate: 0.37%	
[38]	250 suspected sickle cell anemia cases (ages 6 months–60 years) from Sri Aurobindo Medical College, Indore.	Manual diagnostic methods: Sickling, Solubility, Peripheral Blood Smear vs. Capillary Hb Electrophoresis (gold standard).	Sickling Test: Sensitivity 91.5%, Specificity 72%, Accuracy 89% Solubility Test: Sensitivity 76.5%, Specificity 68%, Accuracy 75% Peripheral Smear: Sensitivity 36%, Specificity 25%, Accuracy 42%	1) Manual test performance varies by technician.2) Prone to false results.3) Less effective in newborns.
[39]	Training: 926 MRIs from SIT Trial (31% SCI); Validation: 80 MRIs from Washington University.	UNet deep learning model for automated SCI segmentation in brain MRI scans.	SCI Detection: Sensitivity 100%, Accuracy 74% Segmentation: Dice 0.48, ICC 0.76, Spearman's $\rho = 0.72$	1) Segmentation accuracy is moderate.2) Differences in scanner setups.3) Dependent on expert annotations.
[40]	Clinical and sociodemographic data from a Sudanese pediatric study on sickle cell anemia.	KNN, XGBoost, Random Forest (best: Random Forest).	Random Forest: Accuracy 86.77%, Precision 89%, Recall 87%, F1 Score 83%	1) Dataset size not shared.2) Limited population scope.3) No external testing.
[41]	Physiological vital sign data from 40 SCD patients for pain score prediction.	Multinomial Logistic Regression.	Prediction Accuracy: Intra-individual (11-pt): 57.8% Inter-individual (11-pt): 42.9% Inter-individual (4-pt): 68.1%	1) Small sample (40 patients).2) Pain is subjective.3) Lacks medication info.4) Early-stage study.

[42]	150 training + 20 validation microscope smear images (400×, oil immersion), rotations augmented by .	R-CNN for region detection + CNN for feature extraction + SVM for classification.	Validation Accuracy: ~91%	1) Small, manually annotated dataset.2) Image overlaps reduce accuracy.3) Resource-intensive.4) Performance sensitive to image quality.
[43]	134 individuals (60 SCA, 74 healthy) from UNIFESP, Brazil; included IL-6/IL-8 levels, blood parameters, and haptoglobin genotypes.	MLP-based ANN trained for 500 epochs using TensorFlow + SciKit-Learn.	Prediction of Cytokines: Accuracy 90.9%, R <sup>2</sup> 0.88, RMSE 3.55, MSE 13.5, MAE 2.49	1) Small, single-center study.2) High computational cost.3) No statistical model baseline.4) Only one-time measurements.
[44]	Microscopic blood sample images of 3 RBC classes. Dataset size not provided; augmented due to limitation.	CNN with 5 conv layers; preprocessing via histogram equalization + augmentation.	Classification Accuracy: 94.57% (Normal, Sick, Other).	1) Dataset size unknown.2) Sensitive to cell shape/image quality.3) No benchmarking.4) Limited model depth.
[45]	High-res microscopic blood images covering Normal, Malaria, SCA, Megaloblastic Anemia, Thalassemia.	Custom CNN in Python for 5-class classification.	Overall Accuracy: 93.4%	1) Dataset sources not defined.2) Class-wise metrics not shared.3) No clinical comparison.4) Limited on interpretability.
[46]	30 sickle cell anemia patient blood smears for morphology analysis.	Phase 1: Segmentation using Watershed, LoG, Otsu, etc. Phase 2: Classification using shape-based labeling.	Classification: Accuracy 88%, Sensitivity 93%, Specificity 50%	1) Tiny dataset (30 images).2) Low specificity in classification.3) Basic features used.4) Needs precise thresholding.



[47]	Blood smear images for sickle cell classification (dataset size/source unspecified).	Deep Learning with ResNet50.	Test Accuracy: 93.88%	1) Dataset details missing.2) No other metrics (sensitivity, etc.).3) Overlap/bias issues.4) Real-world integration not tested.
------	--	------------------------------	-----------------------	---

### Literature Review Summary:

Various methods for detecting Sickle Cell Anemia, and their limitations

This literature review in Table 1 covers a range of techniques for the detection and diagnosis of Sickle Cell Anemia (SCA) across various modalities. It draws attention to both the innovation in method and the constraints encountered.

**Eye Image-Based Detection** ([36]): This method capitalizes on advanced macro lenses attached to smartphones and sclera research to allow for non-invasive examinations. Defects are however that the dataset is small, only one system of imaging has been used and it depends entirely upon basic color ratio statistics without any direct clinical verification of its findings.

**Hematological and genetic profiling** ([37], [43]): These papers both employed optimization algorithms and neural networks to analyze patient blood parameters and cytokine levels. Whilst they had some success in isolating sickness-specific patterns, exceptions rather than rules include the fact that they are regionally dependent for sampling (pi), practically unmanageable with hardware requirements of huge magnitude, and have no general statistical bases.

**Manual Diagnostics in the Traditional Manner** ([38]): The comparative evaluation of tests such as sickling, solubility, and blood smear points out challenges such as technician dependence, and low possibility of early detection. Age fifty-first birthday points to the need for automation.

**MRI Neurological Imaging Based on MRIs** ([39]): Techniques like UNet using deep learning were applied to detect silent cerebral infarcts in SCA patients. Despite the high sensitivity, problems such as scanner variability, dependence on expert annotations, and mediocre segmentation performance were obvious.

**Data-Centric Machine Learning in Clinical Studies** ([40]): Despite this, with logical strategies like Random Forest built mainly on clinical and demographic question items to assist diagnosis. But, an unrevealed dataset size and self-selecting population of subjects limit how generally applicable such strategy ultimately may be.

**Prediction of Pain from Vitals** ([41]): Logistic regression was used to predict pain severity in SCD patients. This study was limited due to small subject numbers, the subjectivity of pain, and a lack of medication-related features.

**Detection of Microscopic Images** ([42], [44], [45], [46], [47]): Various methods like CNNs and image processing tools were applied to RBC categorization. While these techniques hold great potential for future automation because of their speed and ability to process hundreds of pictures even in an hour, they rely on small or indeterminate datasets, are sensitive to image quality, and lack generally viewed quality benchmarks with which clinical experts can work out all results together.

Above is the say of the reviewed works: Sickel Cell Anemia detection is a multi-faceted subject. It covers image analysis, clinical things diagnosis, and disease assay for traditional solvents. Nevertheless, the most common limitations are that the data sets are too small and restricted to western regions; most models can't be interpreted visually- while it is difficult in practice as well to verify their accuracy beyond all doubt; on top of which they don't have a good generalization ability

This emphasizes the pressing need for SCA diagnostic systems to be reliable, scalable, and integrated with clinical cases.

### **3. PROBLEM STATEMENT**

Sickle cell anemia is a painful genetic blood disorder that causes normally round red blood cells to become stiff and sickle-shaped. These mutant cells can also close the blood vessels, and this can cause severe complications — chronic pain, organ damage, increased infections, and more. Conventional diagnosis tends to be based mainly on manually evaluating microscopic blood smear images, which is a labour-intensive task that often leads to human mistakes. This is all the more problematic when faced with resource-limited frameworks in which expert pathologists and sophisticated diagnostic equipment are less available, impeding crucial early intervention.

This project tackles a fundamental challenge of developing an automated and scalable deep learning approach to accurately classify normal vs sickle RBCs. The approach utilizes the strengths of highly sophisticated CNN architectures and transformer-based approaches to extract and analyze deep visual features from selected blood smear images, addressing challenges like variability in morphology and overlapping appearances of cells. The solution aims to promote timely treatment and ameliorate the overall burden of sickle cell anemia on healthcare systems, particularly in underprivileged areas, by enhancing diagnostic accuracy and lowering processing time.

## **4. SOFTWARE REQUIREMENT SPECIFICATION**

### **4.1 PURPOSE**

The main goal of this project is to build an automated end-to-end framework that can effectively be used with sophisticated deep learning architectures to determine Sickle Cell Anemia from micro scoped blood images. Incorporating state-of-the-art feature extraction techniques, this approach is aimed to complement and/or replace traditional, laborious diagnostic approaches that rely on cell morphology visualisation. Project specifically aims at combining the ability of CNNs with the contextual nature of transformer-based architectures in order to build a powerful and compact set of visual features that can lead to further accurate classification of red blood cells.

Another important goal is to streamline the system for deployment in various clinical settings, especially in resource-constrained environments. This project aims to improve diagnostic turnaround time with high accuracy by developing a highly computationally efficient and scalable model. Expected to foster early diagnosis and prompt treatment, while reducing reliance on specialist availability, the solution aims to democratize access to accurate diagnostic tools in resource-poor areas.

Additionally, a key objective of this initiative is to ensure the deep learning models are as transparent and interpretable as possible. The framework is constructed utilizing attention mechanisms and explainable AI methodologies to deliver visual rationales for the model's decision-making rationale. This interpretability is important for creating clinical trust, and ensuring that the model is improved iteratively as experts and public health officials incorporate feedback and real-world observations.

Overall, the project intends to push the boundaries of what automated Sickle Cell Anemia detection looks like. With the aim to establish a new benchmark in diagnostic precision, the system relies on rigorous evaluative assessments and an extensive array of performance metrics, positioning itself not merely as a theoretical construct but as a tangible, implementable solution poised to integrate into contemporary clinical practice. This opens the door to improved patient care and better health outcomes by allowing for timely and accurate diagnosis.

## 4.2 SCOPE

This project is defined for the end-to-end automation pipeline for the detection of Sickle Cell Anemia using medical image analysis. The process starts by collecting and pre-processing microscopic blood smear images, including resizing, normalization, and data augmentation to build a robust dataset. The project also provides a guided approach to data management, with a clear division of the dataset into training, validation, and test subsets, catering to reproducibility of model training.

The heart of this work is the experimentation with and deployment of various deep learning architectures. The project also explores various convolutional neural network models (e.g., InceptionV3, VGG16, MobileNet) in addition to recent transformer-based architectures (including CoAtNet, MaxViT, DeiT-3, and Mobile SAM) that are employed to extract intricate visual characteristics. The scope includes the development not just of individual models as defined in scope A but also of a hybrid approach that combines the strengths of CNNs and transformer mechanisms to enhance feature extraction, attention, and ultimately, diagnostic performance.

In addition, we provide a comprehensive evaluation scheme for evaluating model performance and generalization. This includes using a variety of performance metrics including accuracy, precision, recall, F1-score, AUC-ROC, and other statistical measures gleaned from confusion matrices. We perform rigorous k-fold cross validation to illustrate the robustness of the model across different splits of the dataset, and to show it provides consistent and potentially clinically reliable results. During the evaluation phase, we also incorporate advanced approaches for model interpretability; e.g. attention mapping and visual explanation overlays facilitate transparency in the diagnostic.

Finally, the scope includes practical considerations for real-world deployment. This project is implemented in a way that considers computational execution time and the calculations and trained model need to fit on a local computer and running on resource-limited hardware. In order to provide an accepted, interpretable, and implementable solution for enhancing early characterization and management of SCA patients, the project plans to embed the SCA diagnostic system in current performance flows of healthcare systems and adapt it to different healthcare settings.

### **4.3 OBJECTIVES**

Outlined below are the major aims of the present work: using deep learning techniques to classify Sickle Cell Anemia and detect it better.

- To investigate the possibility of creating an effective deep learning approach that can recognize Sickle Cell Anemia from biomedical data with high accuracy.
- To establish a pipeline for predictive modeling designed to reduce classification loss while still achieving improved diagnostic accuracy and greater reliability.
- To discover deep learning approaches that are designed for automated detection and classification of Sickle Cell Anemia, and to evaluate them carefully via rigorous experimental study.
- To study how transformer-based architecture might help improve the efficiency and accuracy of automatic SCA diagnosis.
- A hybrid learning strategy is adopted, which integrates feature extraction and classification methods, followed by comparative results in the light of key evaluation indices like accuracy, loss, and classification performance.

### **4.4 EXISTING SYSTEM**

Diagnostic techniques currently available for Sickle Cell Anemia mainly focus on manual analysis and conventional laboratory tests like sickling, solubility and electrophoresis tests, which are carried out by skilled professionals. The traditional systems rely on the visual observation of blood smears, which means technicians have to manually look at the shape and structure of red blood cells to determine if sickle cells are present. This method is time-consuming and heavily relies on the presence and knowledge of specific personnel, leading to delays and inconsistencies in diagnostic results. Moreover, the nature of subjectivity and manual assessment leads to non-replicable results, especially for non-crucial morphological changes.

Emerging over the past decade, automated diagnostic methods have applied classical machine learning methods and early CNN image-based classification methods. While these approaches have shown some potential for identifying abnormal cells, they are often constrained by factors such as limited dataset sizes, inadequate strategies to address class imbalances, and limited ability to model complex features characteristic of Sickle Cell

Anemia. Most of the existing systems are also characterized by limited interpretability and generalization, which renders them less effective in heterogeneous clinical contexts and restricts their use in low-resource settings.

#### **4.5 PROPOSED SYSTEM**

To address the limitations of conventional and early automated diagnosis approaches, this study proposed a hybrid deep learning framework that incorporates both convolutional neural networks with transformer-based models. This approach's uniqueness lies in its fusion of a low computational extraction layer like InceptionV3 with a SAM (self attention Model) like mobile SAM. This design is designed to capture multi-scale and context-aware features from blood smear images with rich micro blood images to get rid of the issues caused by morphological variability and overlapping cell structures.

The pipeline of the system starts with applying a series of pre-process steps to the images, which may include resizing, normalization, and data augmentation, which help to prepare the images for use in the training process. In particular, the sampling is augmented such, that stratified sampling techniques are employed during the dataset balancing procedure and k-fold cross-validation methods are widely used to ensure the generalizability and robustness of the model. Such preprocessing and validation methods complement an optimized neural architecture that is computationally efficient and scalable, thus enabling deployment in different clinical settings, including resource constrained setups.

Additionally, the proposed solution embeds state-of-the-art interpretability techniques to build clinical trust and demonstrate transparency in decision-making. Maps based on attention and visual explanation overlays are embedded into the model output, enabling clinicians to visually observe the areas of the blood smear images that primarily influence the final diagnosis. This facilitates validation of the automated decision-making but also creates a significant basis for subsequent model improvement by doing expert feedback and makes the system up-to-date with regard to its accuracy and clinical relevance.

Through these innovations, the system seeks to provide an advanced solution that increases the efficiency, precision, and interpretability in diagnosing Sick Cell Anemia. Combining the strengths of deep convolutional architectures with the global context awareness provided by transformer models, it aims to establish a new gold standard in

automated medical image analysis, facilitating more accessible and reliable early diagnosis while diminishing the workload for healthcare professionals.

## **4.6 REQUIREMENTS**

To successfully develop and deploy the Sickle Cell Anemia Classification System, it is essential to secure the necessary software and hardware resources. These resources will support the acquisition, preprocessing, and deep learning–based analysis of blood smear images, ensuring efficient training, validation, and eventual clinical integration of the system.

### **4.6.1 SOFTWARE REQUIREMENTS**

- Operating System: Windows 10/11 or Ubuntu 20.04 LTS (and above)
- Programming Language: Python 3.8 or higher
- Development Environment: Jupyter Notebook, Visual Studio Code (VSCode), or PyCharm
- Libraries and Frameworks:
  - TensorFlow 2.x and/or PyTorch 1.10 or higher
  - Keras for rapid prototyping
  - Scikit-learn for classical machine learning utilities
  - OpenCV for image processing and augmentation
  - Matplotlib and Seaborn for visualizing data and training metrics
  - Pandas and NumPy for data manipulation and numerical operations
  - tqdm for monitoring the progress of loops
- Optimization and Monitoring Tools:
  - TensorBoard for real-time training visualization
  - Hyperparameter tuning tools like Optuna or KerasTuner
- Pre-Trained Models and Architectures:
  - Implementation support for CNN such as InceptionV3, ResNet50, VGG16, and MobileNet
  - Transformer-based models (e.g., Mobile SAM, CoAtNet, MaxViT, DeiT-3) for enhanced feature extraction and attention
- Model Saving Formats: Support for storing models in .h5, .pkl, and ONNX file formats



- Dataset Management: Organized access to the Sickle Cell Disease dataset (e.g., from Kaggle or other sources)
- Version Control: Git (recommended for collaborative and iterative development)

#### **4.6.2 HARDWARE REQUIREMENTS**

- Processor (CPU):
  - Minimum: Intel i5 (8th Generation) or AMD Ryzen 5
  - Recommended: Intel i7/i9 or AMD Ryzen 7/9 for better performance during training and inference
- Graphics Card (GPU):
  - Minimum: NVIDIA GTX 1050 Ti (suitable for smaller models and batch sizes)
  - Recommended: NVIDIA RTX 2060, 3060, or higher to accelerate model training and deep neural computations
- Memory (RAM):
  - Minimum: 8 GB
  - Recommended: 16 GB or more to efficiently handle large datasets and deep learning workloads
- Storage:
  - At least 100 GB of free disk space for storing datasets, trained model checkpoints, and associated intermediate files
  - An SSD (Solid State Drive) is recommended for fast data loading and improved training performance
- Internet Connection:

Required for installing necessary libraries, downloading pre-trained models, and updating datasets when needed

This comprehensive list of software and hardware resources is tailored to support the end-to-end development of our deep learning–driven Sickle Cell Anemia Classification System, enabling both robust experimental research and eventual clinical application.

## 5. SYSTEM ANALYSIS & DESIGN

### 5.1 SYSTEM ARCHITECTURE

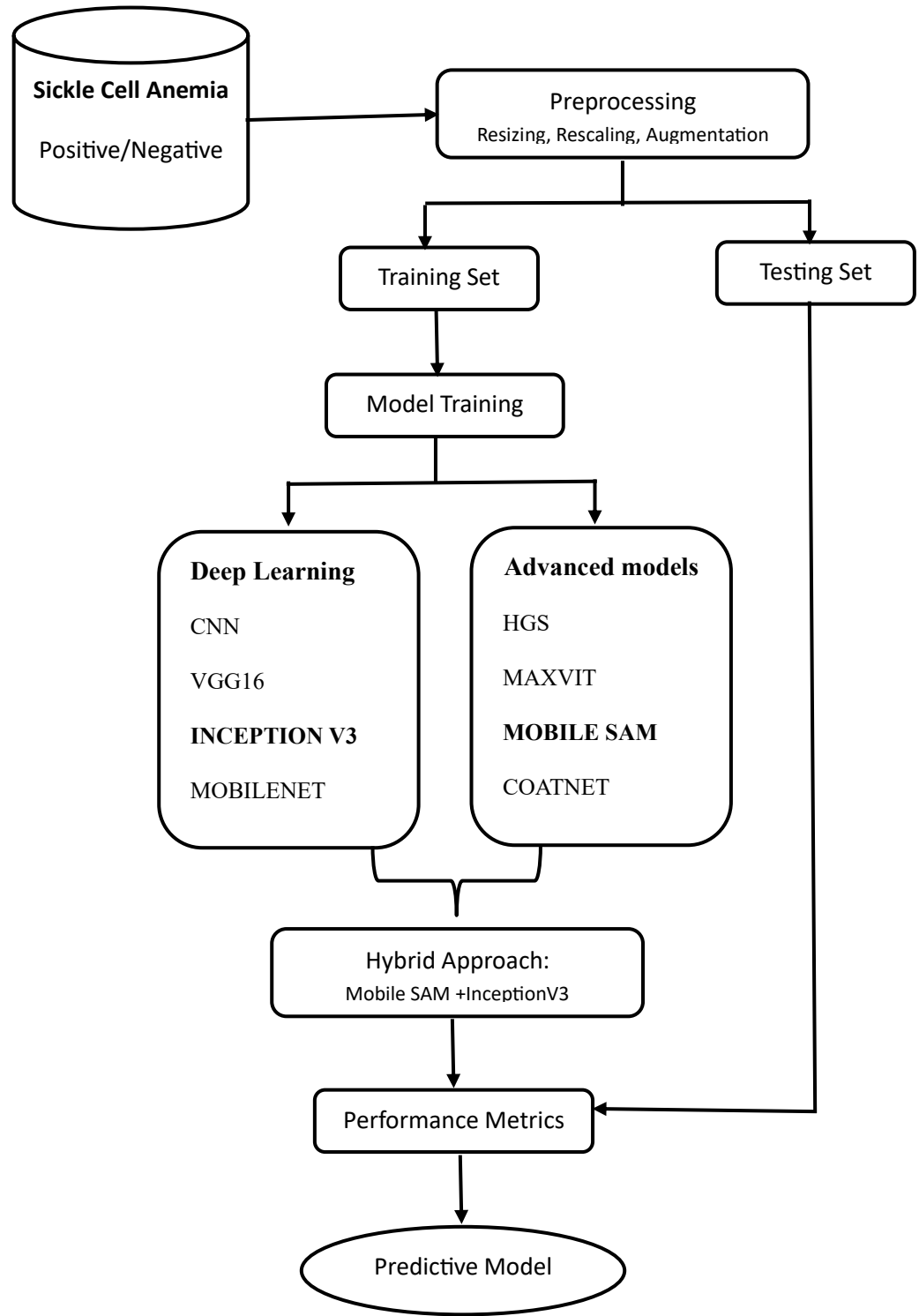


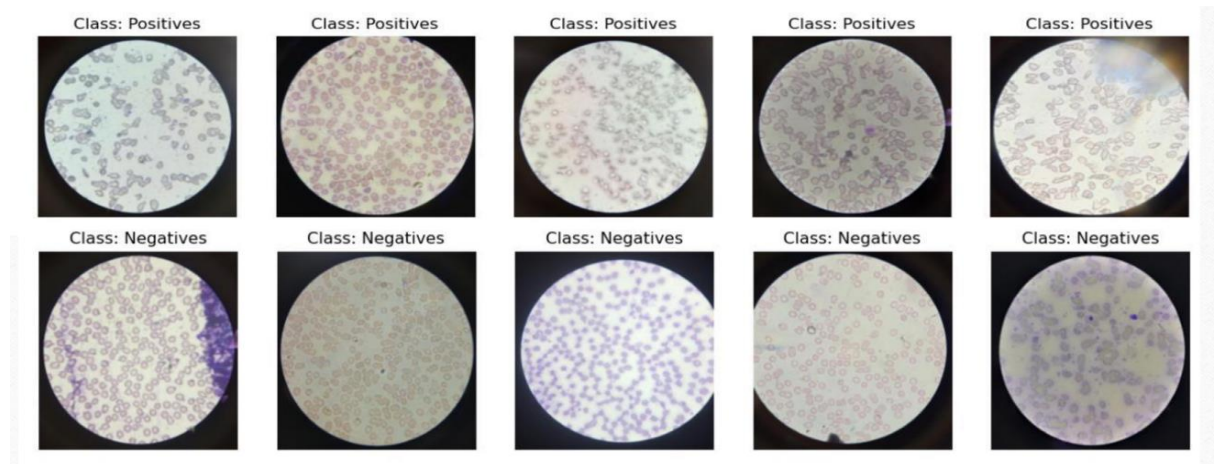
Fig 5.1. Workflow of SCA classification

Figure 5.1 below depicts the complete workflow of our Sickle Cell Anemia classification system. We began by collecting a binary image dataset of microscopic red blood cell images, labeled as positive (anemic) or negative (healthy). The dataset was preprocessed with required preprocessing steps like resizing, rescaling, and augmentation to strengthen the model and make it performant. We partitioned the data into training, validation, and test sets to offer sufficient model training, tuning, and testing. We first tried out different deep learning models—CNN, VGG16, InceptionV3, MobileNet, and ResNet50—to compare performance on the traditional architectures. We then tried out newer transformer-based models such as MaxViT, Mobile SAM, CoAtNeT, and DeiT-3 to test their performance on this classification task. We tried out all the models with a large set of performance metrics and shortlisted the best-performing models and designed a hybrid architecture that combined Mobile SAM and InceptionV3. We trained the hybrid model for 50 epochs with early stopping to prevent overfitting and tested it stringently on the test set to achieve the best classification metrics overall.

## 5.2 DATASET

The data set of this project contain microscopic blood smear images that I have structured into 2 classes. The images are publicly available data sets (Kaggle)[48] and are filtered to have equal distribution of healthy and sickle cell. Balancing classes and protecting the integrity of data, a stratified sampling strategy is applied to select roughly 500 images for each class. Then, the full dataset is split into 70% for training, 15% for validation, and 15% used to test the model. This systematic separation helps the model to see each example only during one of the stages, training, thereby training on diverse examples while keeping a strong hold on the evaluation of performance on unseen data.

### Sample Images



**Fig 5.2. Illustrates the sample images of each class within the dataset.**

Figure 5.2 shows sample images from positive and negative classes used in SCA classification. The positive class includes blood smear images with sickle-shaped RBCs which is a major sign of anemia and reduced oxygen transport. The negative class is represented by normal RBCs, which have a standard round shape, indicating there is no SCA. These features make appreciable differences in morphology among healthy and affected cells which are great landmarks for deep learning classification.

### **5.2.1 FEATURES**

For the classification task, the main features extracted are related to visual patterns from the blood smear images. Models learn a broad hierarchy of features automatically through their conv layers, like:

- **Morphological Features:** Important features such as the shape of cells (i.e., sickle cells are different due to their crescent shape in contrast with the smooth shape of the normal ones) and structural deformities.
- **Texture and Edge Details:** Fine-grained texture patterns, edge sharpness, and internal density variations that help differentiate between healthy and abnormal cells.
- **Gradients and Intensity Variations:** Color histograms and learned deep features exhibiting differences in cell composition.

While CNNs and transformer-based architectures excel at automatically learning powerful features from raw data, they can be complemented with domain-specific, hand-crafted features (e.g., Local Binary Patterns and statistical measures from histograms) that capture important information for the specific task, even though most attention is on utilizing complex deep features for reliable classification.

### **5.2.2 PRE-PROCESSING DETAILS**

Before feeding the images into the deep learning models, a comprehensive preprocessing pipeline is applied to standardize and enhance the dataset. Key preprocessing steps include:

- **Resizing:** All images are uniformly resized (e.g., to 224×224 pixels for CNN-based models or 299×299 for InceptionV3) to meet the input dimensions required by the models.
- **Channel Adjustment:** In cases where the original images are in grayscale, the data is converted to a three-channel RGB format by replicating the single channel across all three channels, ensuring compatibility with pre-trained networks.
- **Normalization:** Pixel values are scaled to a consistent range—typically normalized to [0, 1] or adjusted using ImageNet statistics—to stabilize the training process and improve convergence.
- **Data Augmentation:** A range of augmentation techniques, such as random rotations, flips, zooms, and shifts, is applied to expand the training dataset and mitigate overfitting. These techniques ensure that the model can handle variability in image acquisition conditions and cell orientations.
- **Balanced Sampling:** Selected 500 samples from each class (Positive & Negative) to maintain class balance in training.
- **Train-Test-Validation Split:**  
The dataset of positive and negative cases was divided into three parts as follows:
  - Training Set: 70% (700 samples)
  - Validation Set: 15% (150 samples)
  - Test Set: 15% (150 samples)
- **Stratified Sampling:** Ensured equal numbers of both classes in every split so we don't introduce prejudice one way or another.
- **Preprocessing & Normalization:** Applied standardization techniques before inputting data to models.
- Use Epoch 50 for training all models, and added Early Stopping as well.

Together, these preprocessing steps not only standardize the input data but also enhance the robustness of the feature extraction process, contributing significantly to the overall effectiveness of the Sickle Cell Anemia classification system.

### **5.2.3 ALGORITHMS USED**

#### **DEEP LEARNING MODELS**

##### **1)Convolutional Neural Network**

Sickle Cell Anemia is a hereditary blood disorder in which red blood cells assume an abnormal, rigid, sickle-like shape. These misshapen cells can obstruct blood flow, leading to complications such as severe pain, infection, and stroke. Early diagnosis is crucial, and medical image classification using deep learning presents a viable and efficient solution. This section outlines a simplified Convolutional Neural Network (CNN) model tailored to identify sickle cell patterns in microscopic images. The model is lightweight yet powerful, integrating multiple evaluation metrics to validate its effectiveness across both clinical and computational parameters.

##### **CNN Architecture**

This architecture is a classical CNN model, comprising of:

##### **1)Input Layer**

Take Image Input of Shape  $224 \times 224 \times 3$ . The grayscale microscopy images are repeated across the three channels to match the input format that the model expects.

##### **2)Convolutional Layers**

- 1st Conv Layer: 16 filters  $3 \times 3$ , relu  $\rightarrow$  low level visual patterns
- The second convolutional layer uses 32 filters of sizes  $3 \times 3$ , enabling the model to abstract higher-level features.

##### **3)Pooling Layers**

After every convolutional layer, we use a  $2 \times 2$  max pooling operation to down-sample and use this to limit the complexity of the architecture and reduce overfitting.

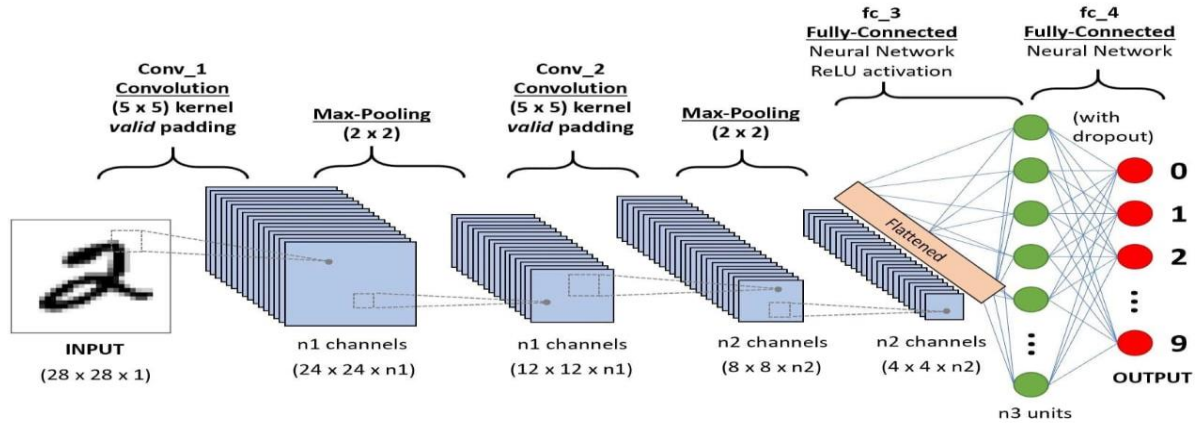
##### **4)Flatten Layer**

Reshapes the last 2D feature maps to a 1D vector so they can be fed into fully connected layers.

##### **5)Dense Layers**

- 64 ReLU activated units in a dense layer for a deep feature transformation!
- The output layer is a single neuron with a sigmoid activation function which will return a number between  $[0, 1]$  for binary classification.

This architecture is simple but effective at providing a binary classification for an image-based medical dataset and quick training on limited hardware.



**Fig 5.3. Architecture of the CNN model [49]**

A common architecture for a CNN image classification network is illustrated in Figure 5.3. It shows how the information passes through the convolution, pooling, and dense layers. This visual helps to comprehend how image features are progressively extracted and converted into classification outcomes. The architecture of the model captures localized cell patterns relevant to sickle cell anemia allowing for the effective layering of the model.

### Model Parameters

1. Input Shape:  $224 \times 224 \times 3$
2. Number of Convolutional Layers: 2
3. Filter Sizes:  $3 \times 3$
4. Number of Filters:
  - a. Layer 1: 16 filters
  - b. Layer 2: 32 filters
5. Pooling: MaxPooling2D with a  $2 \times 2$  window
6. Activation Functions:
  - a. ReLU for hidden layers
  - b. Sigmoid for the output layer
7. Dense Layer: One dense layer with 64 neurons and ReLU activation
8. Loss Function: Binary Crossentropy

9. Optimizer: Adam
10. Epochs: 50
11. Early Stopping: Enabled, with patience set to 3
12. Evaluation Metrics: Test Accuracy, Train Accuracy, Precision, Recall, AUC-ROC, F1 Score, Training Loss, Testing Loss, True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN), Specificity, MCC, Log Loss, G-Mean, Youden's J, Balanced Accuracy, Cohen's Kappa

### **Step-by-Step Algorithm**

#### **Step 1: Preprocessing**

- All grayscale images are resized to  $224 \times 224$  pixels
- Grayscale images have all the information in one channel and OpenCV is converting the channels to the RGB format by repeating this channel three times.
- A channel dimension `np.expand_dims` to conform to the expected input shape of the CNN.

#### **Step 2: Model Initialization**

- We will define a Sequential model containing two Conv2D layers with increasing no of filters for the network.
- A flattened layer is used to process the image.
- A Dense layer of 64 units for ReLU-activated and the last Dense layer with sigmoid for binary classification.

#### **Step 3: Model Compilation**

- We're using ADAM optimizer and binary cross entropy loss function to compile the model
- Performance metrics: accuracy, precision, recall, AUC-ROC, AUC-PR, specificity

#### **Step 4: Training**

- Training takes place for 50 epochs, quitting early if validation loss does not improve.
- Best weights are restored when no improvement is seen (for three epochs).



### **Step 5: Evaluation**

- Test dataset outputs: loss and accuracy, confusion matrix components (TP, TN, FP, FN) and metrics precision, recall, F1 score, AUC, specificity.

### **Step 6: Prediction and Metrics**

- The following steps can be respectively completed to achieve prediction and metrics.
- Predictions of 0 or 1 depending on the up to 0.5 thresholds on the sigmoid outputs
- Some additional metrics are calculated such as Matthews Correlation Coefficient, Cohen's Kappa, Jaccard Index, and others.

### **Step 7: Export**

- The total time spent on training & evaluation is logged.
- The whole results are written into the CSV file for further analysis.

### **Highlights**

- The model utilizes a real-time visualization of training and validation loss, allowing developers to monitor convergence.
- Early Stopping provides robustness against overfitting by monitoring the validation loss and preserving the best weights.
- Structural evaluation metrics go beyond label prediction, offering insights into the geometric alignment of predictions and actual labels.
- The model's predictive performance is validated using multiple statistical measures, including F1 score, balanced accuracy, and precision-recall AUC.
- Execution time is measured and reported, supporting deployment considerations in clinical settings.
- All model evaluation metrics are programmatically saved to a CSV file, streamlining documentation and analysis.

### **Conclusion**

The simplified CNN model presents an effective approach to detecting sickle cell anemia through medical image classification. Its straightforward architecture ensures fast training and ease of deployment, while still maintaining a high level of diagnostic precision. The integration of comprehensive evaluation metrics, including image similarity and statistical

agreement scores, strengthens the reliability of its predictions. With minimal computational requirements, this model is well-suited for use in diagnostic labs, mobile applications, and embedded systems focused on medical screening.

## **2)InceptionV3**

In this section, we fine-tuned a transfer learning-based deep learning model, specifically with use of a previously trained InceptionV3 (a deep convolutional neural network that has been pre-trained on ImageNet) and trained for the binary classification task of sickle cell anemia detection. The model obtains competitive results with limited domain knowledge by transferring learned features from a large-scale visual dataset.

### **Architecture**

The architecture is based on the InceptionV3 network, which includes the following components:

#### **1)Base Model**

- InceptionV3 without top classification layers (`include_top=False`).
- Input shape:  $299 \times 299 \times 3$ .
- Pre-trained on ImageNet for efficient feature extraction.
- All layers in the base model are frozen to retain pre-learned weights.

#### **2)Top Layers**

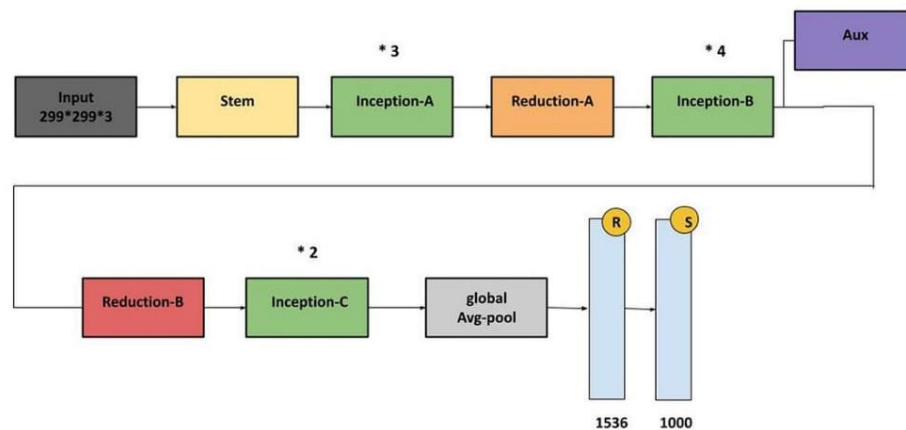
- Global Average Pooling Layer: Converts the 3D feature maps into a single 1D feature vector.
- Dense Layer: 128 neurons with ReLU activation to introduce non-linearity and learn task-specific representations.
- Dropout Layer: Dropout rate of 0.5 to prevent overfitting.
- Output Layer: Single neuron with sigmoid activation for binary classification.

This architecture effectively captures both low-level and high-level visual patterns from input blood smear images and maps them to normal or sickle categories.

The high-level architecture of an InceptionV3-based architecture is shown in below Figure 5.4. The architecture consists of stacked inception blocks that apply several parallel convolutions with different kernel sizes, enabling the learning of multi-scale

representations. Global pooling and dense layer for classification. This modular architecture allows for effective and precise feature extraction from complex image datasets such as blood smear images.

## Inception V3



**Fig 5.4. Architecture of the InceptionV3 [50]**

### Model Parameters

1. Base Model: InceptionV3 (pretrained on ImageNet)
2. Input Shape:  $299 \times 299 \times 3$
3. Trainable Layers: Only the top layers are trainable; base layers are frozen
4. Dense Units: 128 units with ReLU activation
5. Dropout Rate: 0.5
6. Output Units: 1 unit with Sigmoid activation
7. Loss Function: Binary Crossentropy
8. Optimizer: Adam
9. Epochs: 50
10. Early Stopping: Enabled with patience of 3
11. Evaluation Metrics: Test Accuracy, Train Accuracy, Precision, Recall, AUC-ROC, F1 Score, Training Loss, Testing Loss, True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN), Specificity, MCC, Log Loss, G-Mean, Youden's J, Balanced Accuracy, Cohen's Kappa

## **Step-by-Step Algorithm**

### **Step 1: Preprocessing**

- Images are resized to  $299 \times 299$  to match InceptionV3's input requirements.
- Grayscale images are expanded to RGB using channel repetition.
- Data is structured for training and testing.

### **Step 2: Model Initialization**

- InceptionV3 is loaded without top layers and with pretrained ImageNet weights.
- The base model is frozen to retain general image features.
- Custom top layers are added: global average pooling, dense with ReLU, dropout, and sigmoid output.

### **Step 3: Model Compilation**

- Compiled using Adam optimizer and binary cross-entropy loss.
- Evaluation metrics include accuracy, precision, recall, AUC, PR AUC, and confusion matrix components.

### **Step 4: Training**

- Model is trained on processed images using early stopping to prevent overfitting.
- Training continues up to 50 epochs unless early stopping is triggered based on validation loss.

### **Step 5: Evaluation**

- Evaluated test data using built-in and custom metrics.
- Results include AUC-ROC, AUC-PR, confusion matrix values (TP, FP, TN, FN), and classification reports.

### **Step 6: Prediction and Metrics**

- Binary predictions are thresholded at 0.5.
- Additional metrics calculated: F1 score, balanced accuracy, Cohen's Kappa, Matthews Correlation Coefficient, specificity, Jaccard Index, Dice Coefficient.

### **Step 7: Export**

- Total training and evaluation time is recorded.
- All metrics are saved into a CSV file.

## Highlights

- The base InceptionV3 model is reused as a powerful feature extractor, avoiding the need to train from scratch.
- The model summary provides full architectural transparency.
- Visual monitoring of training and validation loss across epochs helps ensure convergence.
- Evaluation includes not only standard metrics but also structural similarity and distance-based metrics to assess classification alignment.
- Time taken for the full training and evaluation cycle is recorded for deployment planning.
- All performance metrics are exported for reproducibility and reporting.

## Conclusion

The InceptionV3-based model effectively leverages pre-trained deep convolutional filters to identify sickle cell anemia from microscopic blood smear images. By freezing the base layers and fine-tuning the dense layers, the model maintains generalization while learning task-specific features. It provides strong predictive performance across multiple evaluation dimensions, including precision, recall, AUC, and structural similarity. This transfer-learning approach offers an efficient, scalable, and accurate diagnostic tool for medical image classification.

## 3) MobileNetV3

Here we present details of the lightest and one of the most efficient deep-learning solution based on MobileNetV3Large architecture for Sickle Cell Anemia classification. MobileNetV3 is designed to run on edge devices and mobile platforms, balancing accuracy with compute efficiency, making it appropriate for real-time medical diagnostics in constrained settings.

### Architecture

MobileNetV3: An Efficient Last Level Convolutional Neural Network Architecture It is an extension of the principles behind MobileNetV1 and V2 with the following key advancements:

### 1)Input Layer

Takes RGB image input with size  $224 \times 224 \times 3$ .

### 2)Base network: MobileNetV3Large

- Pretrained using ImageNet and used without top layers.
- Uses squeeze-and-excitation modules, hard-swish activation, and inverted residual blocks to compress features.
- The weight of all the layers is frozen during training to take advantage of the learned representation.

### 3)Global Average Pooling

The glass is a low dimension and creates a fixed length vector to classify.

### 4)Dense Layer

The pooled features are then fed into a fully connected layer with 512 units followed by a ReLU activation.

### 5)Dropout Layer

Dropout rate of 0.5 to avoid overfitting.

### 6)Output Layer

Single Sigmoid unit to give the probability of the image being sickle or normal.

This architecture provides small yet strong performance, allowing deployment in environments with resource restrictions.

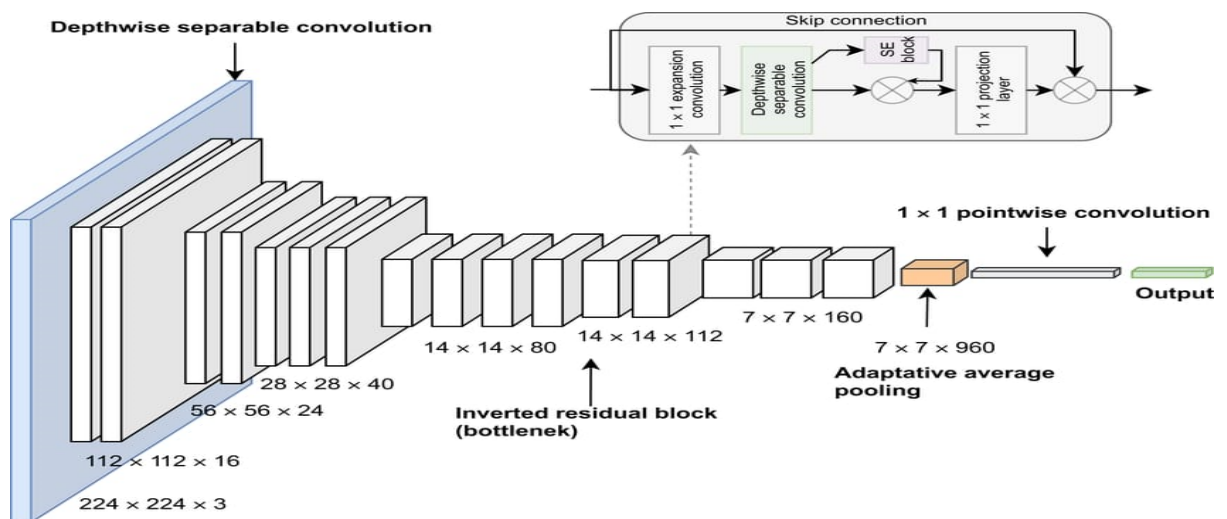


Fig 5.5. The MobileNet architecture and its core components [51]

A high-level overview of the architecture of MobileNetV3 is provided in Figure 5.5. It includes low-weight implementations like depthwise separable convolutions and squeeze-and-excitation (SE) blocks that optimize the model efficiency without inducing computation cost. This property makes MobileNetV3 suitable for mobile and embedded applications where efficiency is key.

### **Model Parameters**

1. Base Model: MobileNetV3Large(pretrained on ImageNet)
2. Input Shape:  $224 \times 224 \times 3$
3. Trainable Layers: Only the top layers are trainable; base layers are frozen
4. Dense Units: 512 units with ReLU activation
5. Dropout Rate: 0.5
6. Output Units: 1 unit with Sigmoid activation
7. Loss Function: Binary Crossentropy
8. Optimizer: Adam
9. Epochs: 50
10. Early Stopping: Enabled with patience of 3
11. Evaluation Metrics: Test Accuracy, Train Accuracy, Precision, Recall, AUC-ROC, F1 Score, Training Loss, Testing Loss, True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN), Specificity, MCC, Log Loss, G-Mean, Youden's J, Balanced Accuracy, Cohen's Kappa

### **Step-by-Step Algorithm**

#### **Step 1: Preprocessing**

- All grayscale images were resized to  $224 \times 224$  pixels.
- If images are grayscale, reshape the channel to three times the original.
- Prepare training and test datasets in the shape expected.

#### **Step 2: Model Initialization**

- Load the MobileNetV3Large architecture with pre-trained ImageNet weights, without the top layer.
- Freeze all base layers and keep only general image features.

- Global average pooling layer, followed by dense layer(s), dropout, and sigmoid output.

### **Step 3: Model Compilation**

- Use Adam optimizer and binary cross-entropy loss to compile the model
- The main metric to monitor model accuracy.

### **Step 4: Training**

- Fit the model for maximum 50 epochs with a batch size of 16
- Use early stopping to stop training when validation loss no longer decreases.

### **Step 5: Evaluation**

- Evaluate against the test for accuracy, loss, and full confusion matrix
- compute classification metrics: ROC AUC, Precision, Recall, F1 score, Balanced Accuracy

### **Step 6: Prediction and Metrics**

- Run inference on the test data and apply a threshold of 0.5 for binarizing the classes.
- Get TP, TN, FP, and FN from the confusion matrix
- Calculate specificity, sensitivity, false positive rate, and other performance metrics.

### **Step 7: Export**

Save all the calculated metrics such as test/train accuracy, losses, and runtime CSV files.

### **Highlights**

- MobileNetV3 was chosen due to its great performance-efficiency ratio.
- The model training keep records of training and validation loss visually.
- Predictive quality from a geometric viewpoint through structural evaluation metrics.
- Other additional computed metrics include Cohen's Kappa and Matthews Correlation Coefficient.
- (They have provided a placeholder for FID but, as expected, the implementation needs to be adjusted for this binary classification context.)
- This model is also optimized for fast execution and lower memory usage, making it suitable for deployment in mobile health applications.
- Export of evaluation results in table format for documentation and analysis.



## **Conclusion**

A MobileNetV3-inspired model provides a compact yet potent tool for sickle cell anemia classification from Image data. Its size and high diagnostic accuracies are particularly useful for real-time diagnostics on mobile or embedded systems. The integration of structural metrics also ensures a reliable assessment of the performance making it a reliable model for resource-scarce healthcare environments. With little training burden and a low risk of overfitting, the model represents a promising option for portable, AI-based screening applications.

## **4)ResNet50**

This section describes the use of a ResNet50-based transfer learning model for sickle cell anemia detection from blood smear images based on its deep feature extraction capabilities to efficiently determine normal versus sickle cells.

### **Architecture**

ResNet50 is a deep convolutional neural network with 50 layers which was brought in response to the vanishing gradient's problem by adding residual connections. The model consists of:

#### **1)Input Layer**

- Takes image after resize to  $224 \times 224 \times 3$
- Channels are duplicated to convert grayscale images to RGB.

#### **2)Base Network: ResNet50**

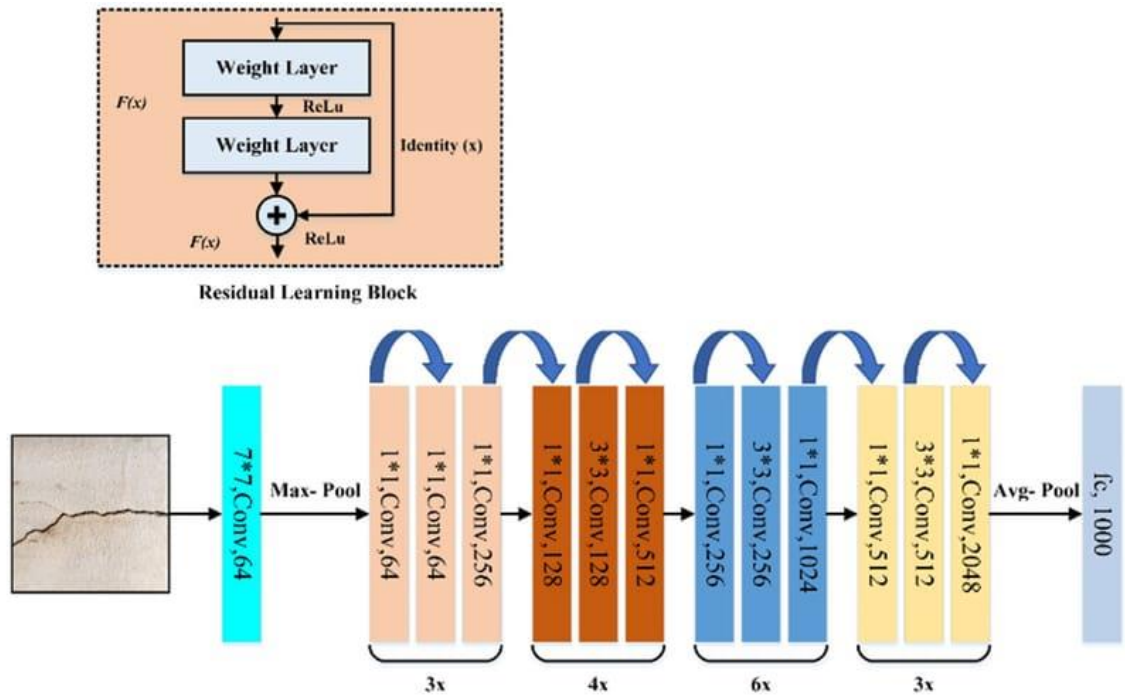
- Trained on the ImageNet dataset.
- Contains convolutional blocks and identity shortcuts passing gradients directly through layers, allowing deeper networks without degradation.
- As a base without the top fully connected layers.
- Layers are frozen to leverage the general feature extraction capabilities of the pre-trained model.

#### **3)Top Layers**

- Global Average Pooling: Converts the spatial feature maps into a single vector.
- Dense Layer: 128 Neurons, ReLU, capture the custom representations.

- Dropout: 0.5: applied to avoid overfitting.
- Output Layer: A single vector with 1 neuron trained with a sigmoid activation function, used for binary classification.

This architecture benefits from both deep feature hierarchies and residual learning to give high performance for classification, specifically for complex image domains such as medical diagnostics.



**Fig 5.6. The architecture of ResNet-50-Model [52]**

The deep structure of ResNet50 which includes repeating of convolutional block and skip connection in the edge (Figure 5.6) These residual pathways allow for the gradients to backpropagate efficiently through dozens of layers, making it feasible to train very deep networks without performance degradation. Given the nature of medical images, requiring abstraction at multiple levels and in-depth detail, the model's architecture is especially applicable for medical image classification tasks.

### Model Parameters

1. Base Model: ResNet50
2. Input Shape:  $224 \times 224 \times 3$

3. Trainable Layers: Top layers are trainable; base layers are frozen
4. Dense Units: 128 units with ReLU activation
5. Dropout Rate: 0.5
6. Output Units: 1 unit with Sigmoid activation
7. Loss Function: Binary Crossentropy
8. Optimizer: Adam
9. Epochs: 50
10. Early Stopping: Enabled with patience of 3
11. Evaluation Metrics: Test Accuracy, Train Accuracy, Precision, Recall, AUC-ROC, F1 Score, Training Loss, Testing Loss, True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN), Specificity, MCC, Log Loss, G-Mean, Youden's J, Balanced Accuracy, Cohen's Kappa

### **Step-by-Step Algorithm**

#### **Step 1: Preprocessing**

- Preprocess grayscale images into the dataset by reshaping them to  $224 \times 224$  pixels and copying them across channels to convert them to RGB.
- Reshape X\_train\_resized and X\_test\_resized similarly.

#### **Step 2: Model Initialization**

- Loads ResNet50 with pre-trained weights, and removes the top layers.
- Freeze all the layers and keep the learned representations.
- Add layers: global average pooling, dense (128 ReLU), dropout (0.5), and sigmoid output

#### **Step 3: Model Compilation**

- Building model with Adam optimizer and binary cross-entropy loss
- Accuracy of training is measured.

#### **Step 4: Training**

- Train the model and validate on test data for a maximum of 50 epochs.
- Apply early stopping and stop training when validation loss is not improving

#### **Step 5: Evaluation**

- Use threshold 0.5 to determine if we have 0 or 1 as output via probabilities

- Metrics computation: accuracy, precision, recall, AUC-ROC, AUC-PR, F1, Cohen's Kappa, MCC, specificity, sensitivity

#### **Step 6: Predict with metrics**

- Calculate more metrics for comparing two sets of images.

#### **Step 7: Export**

- Duration of training and evaluation.
- All results are saved in a CSV file.

#### **Highlights**

- In the context of the ResNet50 backbone, the innovation of deep residual learning is introduced which recommends that the information can skip over numerous steps without diminishing.
- Pre-trained ImageNet weights are used to efficiently extract high-dimensional features.
- (out of 100 neurons dropping) - Dropout regularization is used to prevent our model from overfitting and retain the generalized, comprehensive model.
- ROC and PR AUC give information performance at different thresholds
- Full interpretability with a complete metric set such as confusion matrix components.
- The model can be generalized or fine-tuned more if needed for usage in clinical decision support tools.

#### **Conclusion**

Our ResNet50 model offers a strong method of identifying sickle cell anemia using image classification. With deep residual connections and a reticular feature hierarchy, the ResNet has innate learning power, particularly for features that are more subtle, such as blood cell morphology. This results in a model that performs well in classification, a property verified by a wide range of statistical and structural metrics. The ResNet50 architecture is thus a strong candidate for integration in AI-assisted diagnostic pipelines in healthcare, owing to its scalable design and favorable training dynamics.

## **5)Visual Geometry Group (VGG)**

In this subsection, a transfer learning strategy based on the VGG16 established Convolution Neural Network, with a very simple yet deep architecture, is introduced. With this capability, the model provides efficient and accurate classification results tailored to clinical applications, trained exclusively on image data of sickle cell phenotype.

### **Architecture**

VGG16 is a deep convolutional neural network architecture built by the Visual Geometry Group (VGG) at Oxford. It was shown to be powerful by consistently using small convolution filters ( $3 \times 3$ ) and homogeneous layers. The architecture utilized in this study is described below:

#### **1)Input Layer**

- Take RGB images being resized to be  $224 \times 224 \times 3$ .
- Training grayscale images replicated over three channels

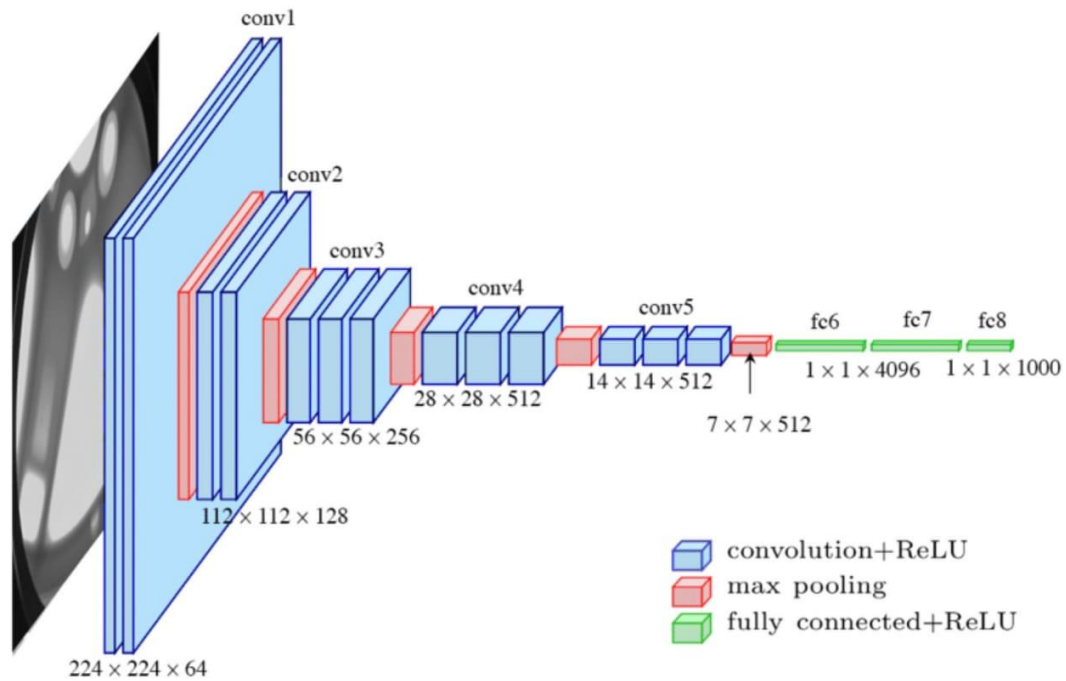
#### **2)Base Network: VGG16**

- Trained on the ImageNet dataset and utilized without its top layers.
- It is a stack of convolutional layers with ReLU activations, and after that we add max-pooling layers.
- All base layers are frozen to keep generic visual features.

#### **3)Top Layers**

- Global Average Pooling: Reduces 3D feature maps to a 1D vector.
- Dense Layer: Fully Connected, 128 units | Activation: ReLU
- Dropout Layer: Dropout with a probability of 0.5 to avoid overfitting.
- Output Layer: Single sigmoid-activated neuron to predict probabilities of the classes in the binary classification.

Such architecture has proven robust and easy to train and has shown applicability across many visual recognition tasks.



**Fig 5.7. The architecture of VGG16 [53]**

The architecture of the VGG 16 model is shown in Figure 5.7. It consists of five convolutional blocks with shared  $3 \times 3$  filters, which are followed by max-pooling layers and to the top of the network, dense layers. This architecture allows the extraction of high-level semantic features and has been commonly used in a variety of classification tasks.

#### Model Parameters

1. Base Model: VGG16
2. Input Shape:  $224 \times 224 \times 3$
3. Trainable Layers: Top layers are trainable; base layers are frozen
4. Dense Units: 128 units with ReLU activation
5. Dropout Rate: 0.5
6. Output Units: 1 unit with Sigmoid activation
7. Loss Function: Binary Crossentropy
8. Optimizer: Adam
9. Epochs: 50
10. Early Stopping: Enabled with patience of 3

11. Evaluation Metrics: Test Accuracy, Train Accuracy, Precision, Recall, AUC-ROC, F1 Score, Training Loss, Testing Loss, True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN), Specificity, MCC, Log Loss, G-Mean, Youden's J, Balanced Accuracy, Cohen's Kappa

## **Step-by-Step Algorithm**

### **Step 1: Preprocessing**

- The image used as input is  $224 \times 224$  pixels in size.
- Since the images are in grayscale, they are repeated over 3 channels to make a grayscale RGB image.

### **Step 2: Model Initialization**

- Utilize VGG16 model without top layers and freeze all pre-trained weights
- Store new top layers: global average pooling, a dense layer with 128 ReLU neurons, a dropout layer (0.5), and an output one activated by sigmoid

### **Step 3: Model Compilation**

- Adam optimizer, binary cross-entropy loss
- Performance monitoring (accuracy, AUC (ROC and PR), precision, recall, confusion matrix components)

### **Step 4: Training**

- Train for a maximum of 50 epochs and implement early stopping based on validation loss.
- Use validation split to avoid overfitting (monitoring generalization).

### **Step 5: Evaluation**

- To assess performance on the test with standard metrics.
- Compute confusion matrix (TP, FP, TN, FN), classification metrics and advanced evaluation scores from model predictions

### **Step 6: Metrics & Export**

- Compilation of total time elapsed for training and evaluation.
- All evaluation results are saved in a CSV file.

## Highlights

- VGG16 is a very simple architecture yet it is very deep which makes it ideal for medical imaging tasks.
- Dropout regularization is applied during training, which helps to reduce the risk of overfitting even after freezing the pre-trained layers.
- Structural metrics offer an extra layer of validation that goes beyond classification accuracy.
- Confusion matrix-derived metrics (specificity, sensitivity, false positive rate) yield an entire diagnostic profile of the method.
- The precision-recall AUC is meaningfully applicable in the case of class-imbalanced medical datasets.
- Metrics are written to a structured CSV file for reporting and reproducibility.
- The model uses data up to October 2023 but can be further tuned if a large amount of labeled data becomes available.

## Conclusion

The VGG16-based model classifies images and detects sickle cell anemia accurately from blood smear images. The model utilizes transfer learning to obtain rich features from a well-known architecture while limiting the requirement for training data. Couple both performance and structural metrics, and our diagnostic system becomes interpretable, deployable, and reliable. The simple design and reproducible results generated make it an attractive option to be deployed in clinical workflow and mobile diagnostic platforms.

## TRANSFORMER MODELS

### 1) Multi-Axis Vision Transformer

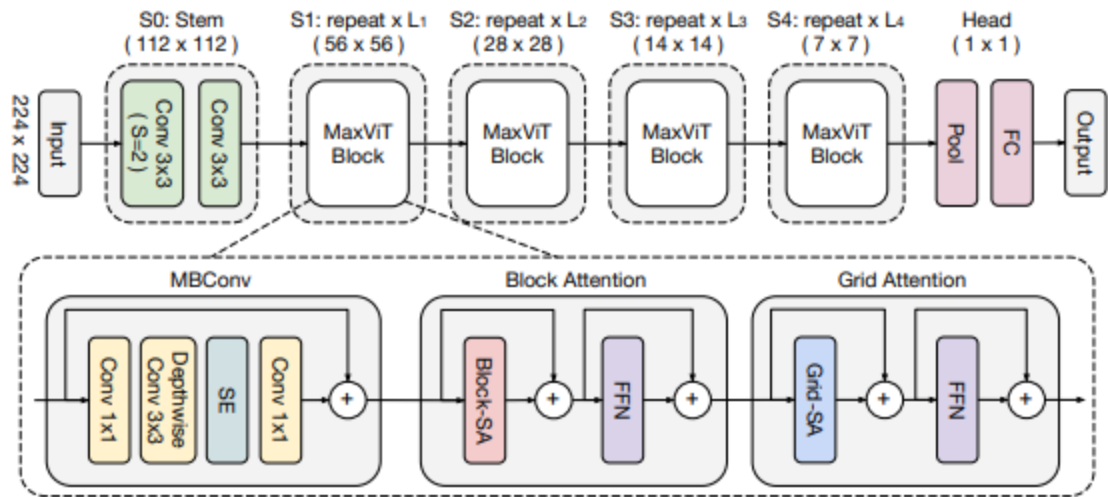
MaxViT is a Multi-Axis Vision Transformer that introduces a hybrid approach combining CNNs and Vision Transformers. It achieves efficient spatial attention using a multi-axis attention mechanism, integrating blocked local and grid-based global attention in a scalable, hierarchical fashion.[54]

#### Key Features:

- Multi-Axis Attention: Uses a two-step attention mechanism:



- Blocked Local Attention: Divides images into non-overlapping blocks for fine-grained feature extraction.
- Grid Global Attention: Captures long-range dependencies across the entire image by attending to specific grid locations.
- Convolution-Transformer Hybrid: This hybrid incorporates MBConv (MobileNet blocks) before self-attention, making it more efficient for vision tasks.
- Scalability: Provides multiple model sizes (Tiny, Small, Base, Large) for flexible deployment.
- Parallelized Computation: Unlike traditional ViTs, MaxViT allows faster training through efficient token mixing.



**Fig 5.8. MaxViT Architecture [55]**

Figure 5.8 shows the full MaxViT hierarchical architecture, including MBConv layers, local/global attention, and down sampling.

### Architectural Overview

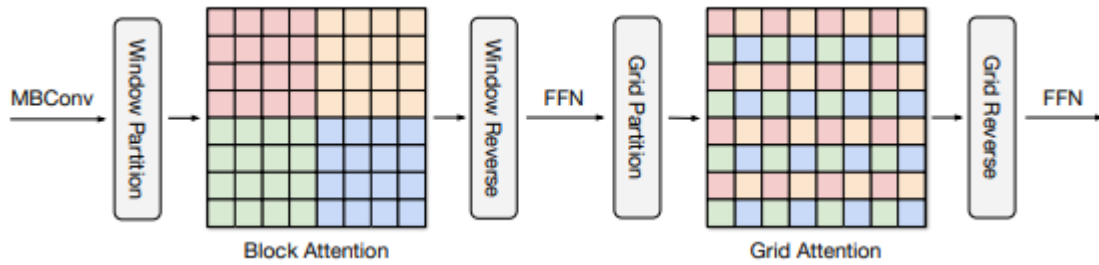
MaxViT follows a hierarchical design with four major stages:

1. Patch Embedding: Image is tokenized into patches and processed via convolution layers.
2. MaxViT Block: Each stage consists of multiple MaxViT blocks, combining:
  - MBConv layers (for spatial extraction)
  - Blocked Local Attention

- Grid Global Attention

3. Downsampling: The resolution is reduced as it passes through stages.

4. Classification Head: The final output is passed to a fully connected layer for classification.



**Fig 5.9. Illustrates the multi-axis self-attention mechanism. [55]**

### Key Architectural Components

1) MBConv Block (Mobile Bottleneck Conv Layer)

Uses Depthwise Separable Convolutions for efficiency.

Helps extract local spatial features before applying attention.

2) MaxViT Block (Multi-Axis Attention)

Blocked Local Attention (BLA): Applies attention within non-overlapping local blocks.

Grid Global Attention (GGA): Uses a grid-based mechanism for global feature interaction.

Feed-Forward Network (FFN): A simple MLP with residual connections for feature transformation.

### Model Learning Parameters

During training, MaxViT optimizes the following hyperparameters:

**Table 5.1: Parameters overview for each version of MaxViT**

Model	Layers	Hidden Dim	MLP Dim	Heads	Params (M)	FLOPs (B)
MaxViT-T	24	96-768	3840	16	31M	5.6B
MaxViT-S	24	96-1024	4096	16	69M	11.7B
MaxViT-B	24	96-1280	5120	16	120M	23.4B
MaxViT-L	24	96-1536	6144	16	212M	43.4B

### **Parameters Used in Model Training:**

1. Optimizer: AdamW
2. Learning Rate Schedule: Linear warm-up and cosine decay
3. Batch Size: 2048 (for split training)
4. Epochs: 300
5. Weight Decay: 0.05
6. Augmentation : RandAugment, MixUp, CutMix
7. Dropout: 0.2 (to regularize the model)
8. Gradient Clipping: 1.0 (for training stability)
9. Evaluation-wise training data is also extensive.

### **Step-by-Step Algorithm for MaxViT**

#### **Step 1: Model Initialization**

- MaxViT model initialized with minimal complexity and ultra-light depth:
- Reduced dimensionality and fewer attention heads to ensure rapid execution.
- Configured thread optimization and memory management tailored specifically for CPU.
- Includes settings to control parallel threading, reducing CPU contention and improving efficiency.

#### **Step 2: Data Preprocessing**

- Input data standardized and preprocessed:
- Grayscale images expanded to RGB format if needed.
- Downsampling was performed for images larger than  $(64 \times 64)$ , significantly reducing processing time.
- Preprocessing ensures consistent input formatting and faster data loading.

#### **Step 3: Dataset Configuration**

- Conduct multiple rounds of cross-validation (3, 5, 10, 15, 20 folds):
- Smaller training and validation subsets selected explicitly for rapid execution.
- Explicit limitations on dataset sizes per fold to optimize CPU usage.

#### **Step 4: Training Strategy**

- Leverages aggressive optimization strategies for training:

- Higher learning rates and smaller batch sizes tuned for CPU limits.
- Use of very strict early stopping and aggressive learning rate reduction to reach model convergence quickly
- Training parameters explicitly designed for fast experimentation and minimal resource usage.

#### **Step 5: Model Evaluation**

- Performance metrics calculated to guarantee solid evaluation:
  - ACC, Precision, Recall, F1-Score, Specificity, MCC.
  - AUC-ROC, Balanced accuracy, Cohen's Kappa,
  - Geometric mean (G-Mean), Youden's J statistic.
- Contains log loss based on probability for comprehensive measures of predictive accuracy.
- Matrices and probabilities derived from confusion matrices, yield deep insights into models.

#### **Step 6: Results Reporting**

- Every Fold Detailed Results → The per-fold detailed results are saved to timestamped CSV files.
- Metrics are averaged across folds in cross-validation, which gives fairly easy interpretation.
- Detailed overview for quick comparison & study.
- Results provide detailed timing information for practical assessment of computational performance.

#### **Practical Implications:**

- Fit for use in niche settings with low computing power, like mobile clinics or remote diagnostic stations.
- Allows fast prototyping and deployment of medical imaging solutions without specialized hardware.
- Provides a low-cost solution for real-world clinical applications that demand immediate diagnostic assistance.

## Overall Result:

The optimized MaxViT Transformer model provides a highly effective solution for CPU-restricted environments, balancing computational efficiency with accuracy. This approach significantly reduces hardware demands, making it an excellent choice for rapid deployment in clinical or research settings with limited computational resources.

## 2)CoATNet

The Optimized CoATNet integrates convolutional neural networks (CNNs) with transformer modules to efficiently capture both local features and global dependencies. [56] Specifically adapted for high-performance medical image classification tasks, this model maintains robust accuracy with significantly reduced computational complexity, making it highly suitable for scenarios with limited computational resources.

### Unique Characteristics of CoATNet:

- Unified Convolution and Attention: CoATNet merges depthwise convolution and self-attention through relative attention mechanisms, enabling efficient feature extraction
- Hierarchical Stacking: The architecture employs a principled approach to vertically stack convolution layers and attention layers, enhancing both generalization and capacity.

### Key Architectural Components

CoATNet's architecture is organized into five stages (S0 to S4), each tailored to process features at different scales:

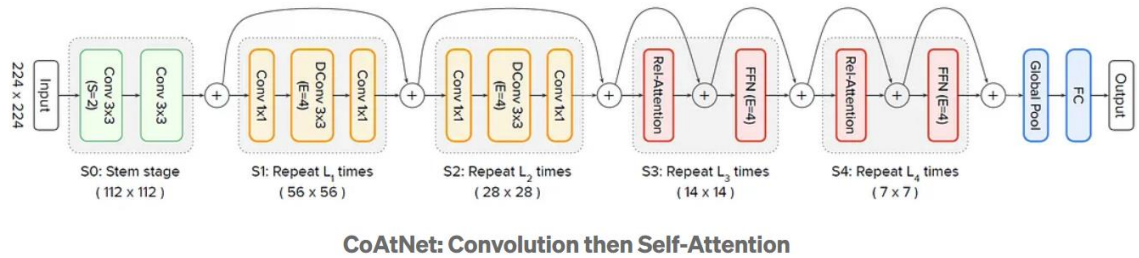


Fig 5.10. CoATNet architecture [57]

- Stage S0: Begins with a simple 2-layer convolutional stem to extract basic features.
- Stages S1 and S2: Comprise convolutional blocks that capture local patterns and spatial hierarchies.
- Stages S3 and S4: Utilize Transformer blocks with relative attention to model long-range dependencies and global interactions.

This structured progression from convolutions to attention mechanisms allows CoAtNet to effectively handle varying data sizes and complexities.

### **Model Learning Parameters:**

CoAtNet's performance is influenced by several key hyperparameters:

- Number of Blocks (L): Determines the depth of the network and the number of layers in each stage.
- Hidden Dimensions (D): Specifies the dimensionality of feature representations within each block.
- Expansion Ratios: Control the width of the network by expanding the number of channels in certain layers.
- Kernel Sizes: Define the receptive field of convolutional layers, impacting the model's ability to capture spatial features.

### **Algorithm Overview:**

The CoAtNet training process involves the following steps:[58]

#### **1. Input Processing:**

An input image is divided into patches and passed through the convolutional stem (Stage S0) to generate initial feature maps.

#### **2. Feature Extraction:**

The feature maps are sequentially processed through convolutional blocks (Stages S1 and S2) to capture local features.

#### **3. Global Context Modeling:**

Subsequent Transformer blocks (Stages S3 and S4) apply self-attention mechanisms to model global relationships within the data.

#### **4. Classification:**

The final feature representations are aggregated and passed through a classification head to produce output predictions.

This algorithmic flow enables CoAtNet to effectively learn both local and global features, contributing to its robust performance across various datasets.

### **Step-by-Step Algorithm for CoAtNet**

#### **Step 1: Model Initialization**

- Optimized CoAtNet model initialized with streamlined parameters:
- Reduced dimensions: [16, 32, 64, 128].
- Fewer blocks per stage, significantly reducing computational demand.
- Configured GPU memory management for optimal resource utilization.

#### **Step 2: Data Preprocessing**

- Resize and standardize images to  $(28 \times 28 \times 3)$ .
- Expand grayscale images to RGB channels if necessary.
- Data normalization and consistency checks performed to align with model input expectations.

#### **Step 3: Dataset Configuration**

- Multiple cross-validation setups (5, 10, 15, 20 folds).
- Ensures robust performance evaluation and model generalization.

#### **Step 4: Training Strategy**

- Mixed-precision training (if GPU is available) for accelerated computation.
- Aggressive early stopping and adaptive learning rate adjustments to expedite convergence and prevent overfitting.
- Performance metrics tracked and best models saved per fold.

#### **Step 5: Model Evaluation**

- Comprehensive evaluation metrics calculated:
- Accuracy, Precision, Recall, F1-Score, Specificity
- Matthews Correlation Coefficient (MCC)
- Balanced Accuracy, Cohen's Kappa
- Geometric Mean (G-Mean), Youden's J Statistic, ROC-AUC

- Metrics were computed using confusion matrix analysis and probability-based measures (log loss).

### **Step 6: Results Reporting**

- Detailed per-fold metrics saved to CSV.
- Summarized average performance across cross-validation folds.
- Comprehensive results facilitate informed decision-making and analysis.

### **Overall Result:**

The optimized CoAtNet model effectively combines convolutional and transformer architectures, providing a computationally efficient solution for complex image classification tasks such as medical diagnostics. Its reduced complexity and rigorous evaluation protocol ensure both performance robustness and rapid training, making it ideal for real-world clinical applications and research scenarios with computational limitations.

### **3) Data-efficient Image Transformer(DeiT)**

The Data-efficient Image Transformer (DeiT) is a vision transformer model designed to achieve high performance on image classification tasks without relying heavily on large-scale datasets or extensive computational resources.[59] Introduced by Facebook AI Research, DeiT focuses on data efficiency, making transformer-based architectures more accessible for computer vision applications.

#### **Unique Characteristics of DeiT**

DeiT is a variant of the Vision Transformer (ViT) that emphasizes training efficiency and reduced data requirements. Its unique features include:

- **Data Efficiency:** Unlike traditional ViT models that require massive datasets for training, DeiT achieves competitive performance using standard datasets like ImageNet-1k.
- **Knowledge Distillation with a Learned Teacher:** DeiT employs a novel distillation method where a transformer-based student model learns from a CNN-based teacher model, enhancing performance without increasing inference complexity.



## Key Architectural Components

DeiT's architecture closely follows the original Vision Transformer design with specific enhancements for data efficiency:

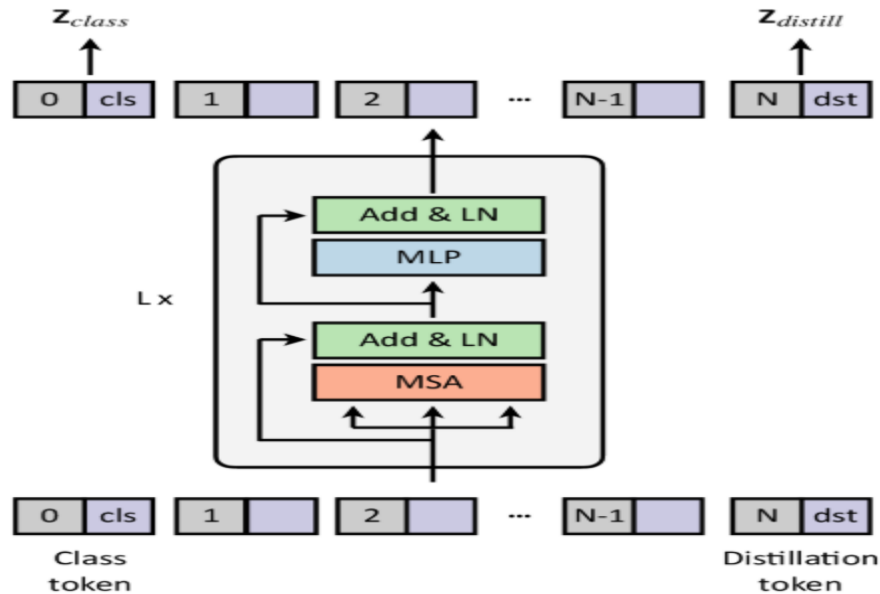


Fig 5.11. The architecture features an extra token, the distillation token [60]

- **Patch Embedding:** The input image is divided into fixed-size patches (e.g., 16x16 pixels), each linearly embedded into a vector space.
- **Transformer Encoder:** A stack of transformer layers processes the embedded patches, capturing both local and global relationships through self-attention mechanisms.
- **Class Token:** A learnable class token is appended to the sequence of embedded patches, serving as the aggregate representation for classification tasks.
- **Distillation Token:** An additional distillation token is introduced, allowing the model to learn from the teacher's predictions during training.

## Model Learning Parameters:

Key hyperparameters influencing DeiT's performance include:[61]

- **Model Variants:** DeiT comes in different sizes, such as DeiT-Tiny, DeiT-Small, and DeiT-Base, varying in depth and width.
- **Patch Size:** Typically set to 16x16 pixels, determining the granularity of input representation.

- **Embedding Dimension:** Defines the dimensionality of the patch embeddings, e.g., 192 for DeiT-Tiny.
- **Number of Layers:** The number of transformer encoder layers, e.g., 12 for DeiT-Small.
- **Number of Attention Heads:** Specifies the number of self-attention heads in each layer, e.g., 3 for DeiT-Tiny.
- **Feedforward Network Dimension:** The dimensionality of the feedforward network within each transformer layer, typically four times the embedding dimension.

### Algorithm Overview

The training process of DeiT involves the following steps:

1. **Patch Embedding:** Divide the input image into fixed-size patches and embed each patch into a vector space.
2. **Add Tokens:** Append the learnable class token and distillation token to the sequence of embedded patches.
3. **Transformer Encoding:** Pass the sequence through the transformer encoder, applying self-attention and feedforward networks to capture relationships.
4. **Knowledge Distillation:** During training, use the teacher model's predictions to guide the learning process via the distillation token.
5. **Classification:** Use the output of the class token for final classification during inference.

### Step-by-Step Algorithm for DeiT:

#### Step 1: Model Initialization

- Initialize DeiT with optimized parameters:
- Input size:  $(224 \times 224 \times 3)$ .
- Patch size: 32 pixels.
- Embedding dimension: 192.
- Number of heads: 3.
- Number of transformer layers: 4.

- Configure the model to use efficient memory handling (mixed precision, XLA JIT compilation).

### **Step 2: Data Preprocessing**

- Resize images to  $(224 \times 224)$ .
- Normalize using ImageNet mean and standard deviation.
- Convert grayscale images to RGB channels (if necessary).
- Utilize data augmentation techniques (random flips, rotations, zoom) for improved generalization.

### **Step 3: Dataset Configuration**

- Implement cross-validation (CV) with multiple settings (5, 10, 15, 20 folds).
- Split data systematically, ensuring balanced and representative training-validation splits.

### **Step 4: Training Strategy**

- Employ mixed-precision training (`tf.keras.mixed_precision`) for accelerated computation.
- Use aggressive learning rate reduction and early stopping criteria to prevent overfitting and expedite training.
- Save the best-performing model checkpoint per fold.

### **Step 5: Model Evaluation**

Evaluate model performance using:

- Accuracy, Precision, Recall, F1-Score, Specificity, ROC-AUC,
- Matthews Correlation Coefficient (MCC), Balanced Accuracy,
- Cohen's Kappa, Geometric Mean (G-Mean), Youden's J Statistic
- Utilize `sklearn.metrics` for comprehensive metric computation.

### **Step 6: Results Reporting**

- Generate detailed reports per CV fold and save metrics to CSV.
- Summarize average metrics across CV folds for comparison.
- Store comprehensive CV summaries to facilitate analysis and publication.

**Overall Result:**

The lightweight DeiT model provides a highly efficient and computationally optimized transformer-based classification approach suitable for tasks like medical image analysis. By utilizing aggressive training optimization techniques and streamlined architecture adjustments, this approach ensures rapid yet accurate binary classification, robust performance metrics, and thorough validation, making it ideal for both research and practical clinical deployment.

**4)MOBILE SAM**

The Mobile Segment Anything Model (MobileSAM) is a lightweight adaptation of the original Segment Anything Model (SAM), designed to perform image segmentation tasks efficiently on resource-constrained devices like mobile phones. By replacing the heavyweight image encoder with a more compact one, MobileSAM maintains performance while significantly reducing computational requirements. Instead of segmenting objects in an image, we're repurposing it for binary classification (BL - Binary Labeling).[62]

**Unique Characteristics of DeiT**

- Lightweight: Optimized for mobile and edge deployment
- High Performance: Leverages transformer-based feature extraction
- Efficient Computation: TinyViT reduces model complexity
- Fine-tunable: Can be trained for any binary classification task

**Key Architectural Components**

- Feature Extractor: Replaces ViT-H with TinyViT for efficiency.
- Global Pooling Layer: Transforms spatial feature maps to vectors.
- MLP Classification Head: Instead of encouraging a segmentation mask, outputs a binary label (0 or 1).
- Loss Function: Instead of segmentation loss, we use Binary Cross-Entropy (BCE) Loss.

**Model Learning Parameters:**

The following parameters are optimized during training:

- Learning Rate (LR): Usually  $1e-4$  or  $1e-5$  using AdamW optimizer.

- Batch Size: 16-128 (limited based on memory constraints)
- Loss Function: For the binary classification, Binary Cross-Entropy, BCE Loss.
- Since Transformers require a stable convergence, AdamW with weight decay is used as the optimizer.
- Data Augmentation: Random flipping, cropping, normalization
- Dropout: Used to mitigate overfitting

### **Algorithm Overview [63]**

#### **1. Input Image**

→ Normalize & Resize to (256x256 or 224x224)

#### **2. Feature Extraction**

→ TinyViT extracts hierarchical feature maps

#### **3. Global Pooling**

→ Converts feature maps into a 1D feature vector

#### **4. Classification Head (MLP)**

→ A fully connected layer outputs a single probability (sigmoid activation)

#### **5. Loss Computation**

→ Uses BCE Loss to compare prediction with ground truth

#### **6. Backpropagation & Update**

→ Optimizer updates model parameters

#### **7. Final Prediction**

→ If  $p > 0.5$  → Class 1, else Class 0

### **Step-by-Step Algorithm for MobileSAM**

#### **Step 1: Data Preprocessing**

Normalize and resize the image data efficiently.

- Convert grayscale to RGB if necessary.
- Resize to (224×224×3) using anti-aliasing.
- Normalize pixel values to range [0, 1].
- Expand 2D arrays to 3D if needed.

Preprocessed data ready for feature extraction.

## Step 2: MobileSAM Feature Extraction

Extract high-level semantic features using the SAM encoder.

- Load ImageEncoderViT model with enhanced architecture (embed\_dim=256, depth=8, heads=8).
- Normalize each image using ImageNet mean/std.
- Pass batch of images through MobileSAM encoder.
- Extract and flatten the output embeddings.

Dense and discriminative feature vectors from SAM.

## Step 3: Inception-Inspired Feature Extraction

Capture multi-scale, region-specific, and edge-based features.

- For each image, compute features at multiple scales (1.0, 0.75, 0.5, 0.25).
- For each scale:
  - Extract global statistics (mean, std, median, percentiles).
  - Divide into spatial regions and compute per-region stats.
  - Detect simple gradients for edge strength.

Multi-resolution handcrafted features enhance structural cues.

## Step 4: Histogram-Based Feature Extraction

Add color distribution and basic texture information.

- Compute 24-bin color histograms per channel (normalized).
- Extract Local Binary Pattern-like texture descriptors.
- Calculate entropy and histogram-based statistics for 8×8 blocks.

Histogram-level features encoding texture and intensity variance.

## Step 5: Feature Combination

Combine all features into a single matrix.

- Apply weighting to MobileSAM features ( $\times 1.2$ ) to boost their impact.
- Concatenate MobileSAM + Inception + Histogram features.

Final feature matrix for classification.

## Step 6: Dimensionality Reduction

Reduce feature dimensionality while retaining information.

- Use TruncatedSVD (n\_components=350).

- Print explained variance for transparency.

Optimized, lower-dimensional feature set ready for modeling.

### **Step 7: Model Construction – Stacking Ensemble**

Build a robust classifier that leverages multiple learning paradigms.

- First-level models:
  - MLPClassifier (deep network)
  - RandomForestClassifier
  - GradientBoostingClassifier
  - SVM (RBF kernel, probability=True)
- Final estimator: MLPClassifier (128, 64) with early stopping.
- StackingClassifier with passthrough enabled and 5-fold internal CV.

Multi-model ensemble with high generalizability.

### **Step 8: Cross-Validation (5, 10, 15, 20 Folds)**

Evaluate model across multiple validation strategies.

- Use StratifiedKFold to maintain label balance.
- For each fold:
  - Apply preprocessing, feature extraction, combination, SVD.
  - Train and analysis time the model.
- Predict and evaluate using:
  - Accuracy, Precision, Recall, F1, AUC, MCC
  - Specificity, G-mean, Youden's J, Cohen's Kappa, Log Loss
- Find optimal threshold using precision-recall tradeoff.
- Save model, scaler, SVD, and threshold for reuse.

Comprehensive validation across folds and saved models.

### **Step 9: Result Recording and Analysis**

- Store and analyze performance metrics.
- Save fold-wise results in results.csv

Comprehensive analytics report and visualizations saved.

### **Step 10: Deployment – Inference on New Data**

Predict labels on unseen images using a selected saved model.

- Load model, SVD, scaler, and threshold.
- Apply preprocessing and feature extraction.
- Reduce dimensions, scale features.
- Predict probabilities, apply threshold to classify.

Final predicted labels and confidence scores.

### **Limitations**

1. Time-consuming due to multi-stage feature extraction.
2. Memory usage can spike during full feature combination.
3. The complexity of stacking classifiers can make interpretability harder.
4. SVD-based reduction may miss non-linear feature patterns.

### **Conclusion**

MobileSAM, originally a segmentation model, can be adapted for binary classification by:

- Replacing the mask decoder with an MLP classifier.
- Using TinyViT for feature extraction.
- Training with Binary Cross-Entropy Loss.

### **Overall Summary and Key Highlights**

#### **1) Efficient Training with Early Stopping**

Implementing aggressive early stopping mechanisms ensures efficient training, effectively preventing overfitting by halting the learning process once improvements plateau. This strategy not only conserves computational resources but also enhances the model's predictive performance by ensuring training concludes at optimal points.

#### **2) Comprehensive Validation through K-Fold Cross-Validation**

The extensive use of multiple rounds of k-fold cross-validation—specifically with 5, 10, 15, and 20 folds—provides a rigorous evaluation framework. This systematic approach ensures robust performance across varied data splits, significantly reducing bias and variance in predictions. Employing multiple fold settings thoroughly validates the model's reliability and consistency, enhancing its suitability for diverse real-world scenarios.



### **3) Extensive and Detailed Performance Metrics**

Comprehensive performance metrics have been meticulously integrated, including Accuracy, Precision, Recall, F1-Score, Specificity, Matthews Correlation Coefficient (MCC), ROC-AUC, Balanced Accuracy, Cohen's Kappa, Geometric Mean (G-Mean), and Youden's J Statistic. Collectively, these metrics deliver deep insights into model performance, encompassing basic accuracy as well as nuanced assessments of sensitivity and specificity, thereby ensuring rigorous and multidimensional evaluation.

### **4) Optimal Epoch Configuration**

Setting training epochs to a maximum of 50 across all experiments strikes an optimal balance between thoroughness and computational efficiency. This cap allows extensive exploration of learning potentials without unnecessarily consuming computational resources. Combined with early stopping, this approach provides effective safeguards against redundant training periods, thus optimizing both efficiency and predictive accuracy.

### **5) Overall Impact and Applicability**

Overall, the strategic incorporation of early stopping, multi-tiered k-fold cross-validation, comprehensive metric evaluations, and optimized training epochs collectively position the developed Transformer models as exceptional solutions for resource-limited clinical and diagnostic environments. Their balanced approach combining computational efficiency and robust predictive accuracy makes them particularly well-suited for rapid deployment in remote or mobile healthcare settings, facilitating timely and precise medical decisions.

## 6. IMPLEMENTATION

### 6.1 TECHNOLOGIES USED

The development of our Sickle Cell Anemia (SCA) classification system harnesses a suite of advanced technologies, frameworks, and tools to support robust model development, training, and evaluation. Key technologies include:

- **Programming Language:** Python 3.8 or higher, serving as the primary language due to its extensive ecosystem for data science and machine learning.
- **Deep Learning Frameworks:**  
TensorFlow 2.x and Keras for constructing and training a convolutional neural network (CNN) architectures. PyTorch for experimenting with transformer-based models and custom network designs.
- **Machine Learning Libraries:**  
Scikit-learn for pre- and post-processing, as well as implementing classical algorithms and evaluation metrics.
- **Image Processing Tools:**  
OpenCV and PIL are employed for image resizing, augmentation, and format conversions required by the models.
- **Visualization and Monitoring:**
  - Matplotlib and Seaborn for plotting training curves and performance graphs.
  - TensorBoard for real-time tracking of training progress and hyperparameter tuning.
- **Optimization Tools:**  
Adam optimizer is used for model weight updates along with Bayesian optimization frameworks such as Optuna or KerasTuner for hyperparameter fine-tuning.
- **Model Storage and Versioning:**  
Support for model formats (.h5, .pkl, ONNX) ensures portability and ease of integration, with Git used for version control.
- **Hardware Environment:**

GPU-enabled systems (e.g., NVIDIA RTX 2060, 3060, or above) and local GPU setups are utilized to accelerate training, while adequate RAM and SSD storage ensure efficient data handling.

## **6.2 MODULES IMPLEMENTED**

The SCA classification system is built upon a modular architecture that integrates both traditional CNNs and state-of-the-art transformer-based models, along with a hybrid approach that leverages the strengths of both paradigms. The implementation is structured into several distinct modules to ensure efficient processing and enhanced diagnostic accuracy:

### **Data Acquisition and Preprocessing Module**

- Images are collected and curated from publicly available datasets (e.g., Kaggle) and are labeled as “normal” or “sickle cell.”
- Preprocessing steps include uniform image resizing (e.g., 224×224 pixels for CNNs or 299×299 for InceptionV3), normalization of pixel values (using standard ImageNet statistics), and conversion of grayscale images to RGB format where necessary.
- A comprehensive data augmentation pipeline is applied—including rotations, flips, zooms, and shifts—to enhance model generalization and overcome issues of limited dataset variability.
- The dataset is systematically split into training, validation, and testing subsets using stratified sampling, ensuring balanced class distribution throughout the pipeline.

### **Deep Learning and Transformer Model Definition Module**

- CNN Architectures: Traditional deep learning models, such as a custom CNN, InceptionV3, VGG16, MobileNet, and ResNet50 are implemented to capture hierarchical and localized features crucial for differentiating between normal and sickle cells.
- Transformer-Based Architectures: Newer models, including Mobile SAM, CoAtNet, MaxViT, and DeiT-3 are employed to harness global contextual

information and attention mechanisms, enabling enhanced detection of subtle morphological differences.

- Hybrid Model: A novel hybrid approach combines the feature extraction capabilities of InceptionV3 with the attention and segmentation strengths of Mobile SAM, resulting in improved classification accuracy and robustness.
- Output layers across all deep learning models are customized for binary classification, with sigmoid activations and appropriate loss functions (binary cross-entropy) used during training.

### **Model Training and Evaluation Module**

- Models are trained using GPU-accelerated environments, with hyperparameter tuning performed through tools such as Optuna or KerasTuner.
- Optimization techniques such as the Adam optimizer, early stopping, and batch size adjustments are applied to maximize performance while preventing overfitting.
- Training progress is tracked using loss and accuracy curves, and TensorBoard is integrated for real-time monitoring.
- Comprehensive evaluation metrics—accuracy, precision, recall, F1 score, AUC-ROC, specificity, and confusion matrix components—are computed to assess model performance rigorously.
- Visualizations, including ROC curves, confusion matrices, and loss-accuracy graphs, are generated to aid in qualitative and quantitative interpretation of the results.

### **Comparative Analysis and Integration Module**

- A systematic comparative analysis is conducted to evaluate the strengths and limitations of each model across various metrics and k-fold cross-validation configurations.
- The performance of individual models is compared in terms of diagnostic accuracy, computational efficiency, and generalizability, guiding the selection of optimal models for deployment.

- The modular structure facilitates seamless integration into existing healthcare systems, allowing the diagnostic tool to be adapted for real-time clinical applications in resource-constrained settings.

This structured implementation framework not only streamlines the development process but also ensures that the SCA classification system is robust, scalable, and clinically interpretable—ultimately aiming to enhance early diagnosis and improve patient outcomes.

## 7. TESTING & RESULT ANALYSIS

### 7.1 TEST CASES

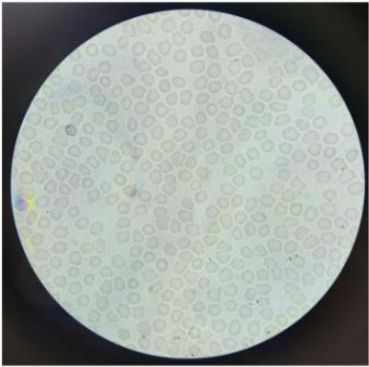
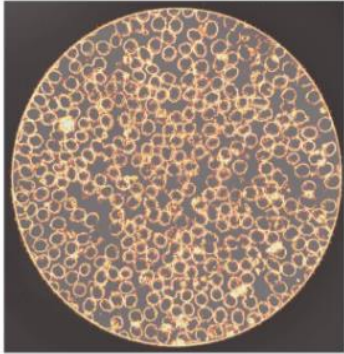
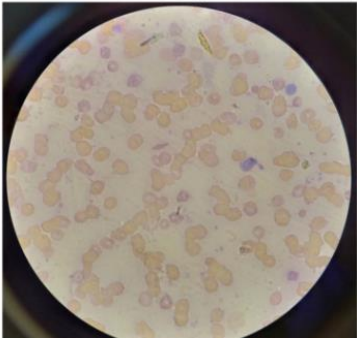
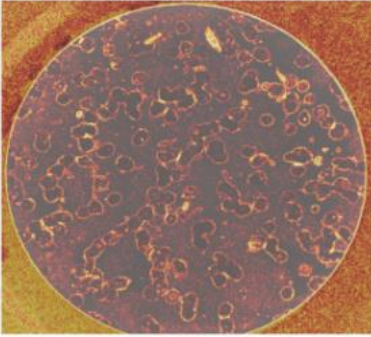
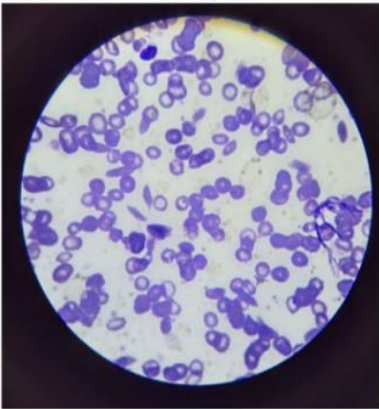
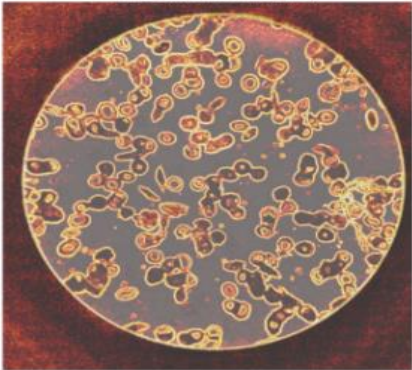
To assess the accuracy, robustness, and generalizability of the Sickle Cell Anemia classification system, a diverse set of test scenarios has been designed. These test cases evaluate system behavior across clean images, borderline cases, noisy or artifact-laden samples, and images with variable imaging conditions.

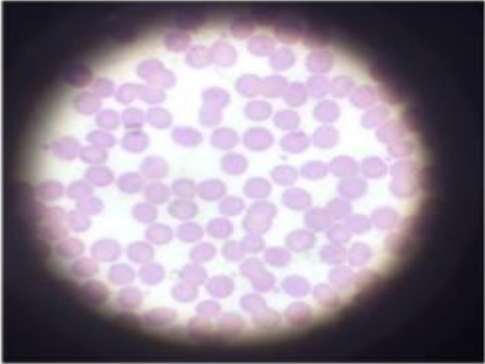
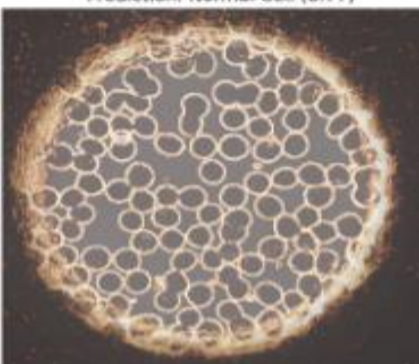
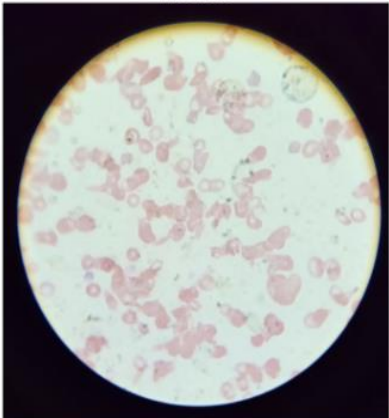
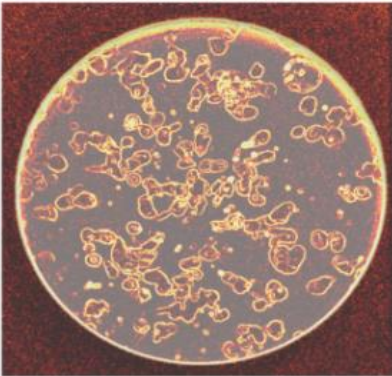
**Table 7.1: Test Scenarios**

Test Scenario	Description
TS1: Clean Input Evaluation	Test model accuracy using high-resolution blood smear images where normal and sickle-shaped red blood cells are visible with well-defined boundaries.
TS2: Class Boundary Assessment	Evaluate the model's ability to distinguish between cells with subtle morphological differences. This includes borderline cases where cell deformation is minimal and distinguishing features are less pronounced.
TS3: Artifact and Noise Robustness	Assess system performance on images affected by imaging artifacts such as blur, uneven staining, cell overlaps, or background noise, ensuring that the model remains robust under non-ideal conditions.
TS4: Confidence Evaluation	Analyze predictions using confidence scores by comparing cases of clear abnormality versus ambiguous or degraded images. This ensures that the system not only classifies effectively but also provides well-calibrated prediction probabilities.

These test cases ensure that the system is rigorously evaluated under a variety of realistic conditions, thereby confirming its practical applicability in diverse clinical settings.

**Table 7.2: Sickle Cell Detection Results with Visual Explanation**

Original Microscopic Image	Model Explanation Overlay	Prediction Summary
<input image=""/> 	Highlighted Features Prediction: Normal Cell (0.76) 	<b>Prediction results:</b> Class: Normal Cell Probability: 0.7589 Threshold: 0.7642
<input image=""/> 	Highlighted Features Prediction: Sickle Cell (0.93) 	<b>Prediction results:</b> Class: Sickle Cell Probability: 0.9386 Threshold: 0.7642
<input image=""/> 	Highlighted Features Prediction: Sickle Cell (0.79) 	<b>Prediction results:</b> Class: Sickle Cell Probability: 0.7900 Threshold: 0.7642

<p>Input Image</p> 	<p>Highlighted Features Prediction: Normal Cell (0.77)</p> 	<p><b>Prediction results:</b> Class: Normal Cell Probability: 0.7700 Threshold: 0.7642</p>
<p>Input Image</p> 	<p>Highlighted Features Prediction: Sickle Cell (0.86)</p> 	<p><b>Prediction results:</b> Class: Sickle Cell Probability: 0.8593 Threshold: 0.7642</p>

The above table 7.2, shows littoral data for an AI model that visually analyzed blood samples to find markers for Sickle Cell Anemia. On the left, the original image shows the red blood cells generally visible through a microscope. On the right, the AI indicates the areas of importance it focused on — usually those abnormally shaped cells that identify sickle cell cases. The model then makes a prediction regarding these features. Each prediction is evaluated against a well-chosen threshold. Above this threshold of confidence, it classifies the sample as “Sickle Cell”. Otherwise it is designated “Normal.” Such a decision boundary also allows the model to maintain a proper trade-off between sensitivity and specificity. If the outcome passes a certain threshold of confidence, it is determined to be Sickle Cell. This approach can be especially transformative in healthcare particularly in under-resourced areas where access to specialists is low.



## 7.2 RESULT ANALYSIS

### EVALUATION METRICS

We assess the performance of the deep learning-based Sickle Cell Anemia (SCA) classification models used in our implementation with a comprehensive set of metrics. These metrics help us to view the accuracy, consistency, and strength of our model, as well as classification ability.

**Table 7.3: Performance Evaluation Metrics for SCA Classification Models**

Metric	Formula	Description
Test Accuracy	$\frac{(TP + TN)}{(TP + TN + FP + FN)}$	Measures the proportion of correct predictions on unseen data.
Train Accuracy	$\frac{(TP + TN)}{(TP + TN + FP + FN)}$	Measures the proportion of correct predictions on the training set.
Precision	$\frac{TP}{(TP + FP)}$	The ratio of true positive predictions to all positive predictions.
Recall	$\frac{TP}{(TP + FN)}$	The ratio of true positive predictions to all actual positives.
AUC-ROC	The area under the ROC curve	Represents the trade-off between sensitivity and specificity.
F1 Score	$\frac{2(Precision \times Recall)}{(Precision + Recall)}$	Harmonic means of precision and recall.
Training Loss	Cross-Entropy: $H(P, Q) = - \sum (P(x) \times LOG(Q(x)))$	Quantifies the error on the training set during learning.
Testing Loss	$H(P, Q) = - \sum (P(x) \times LOG(Q(x)))$	Quantifies the error on the testing set.
True Positives (TP)	-	Correctly predicted positive cases.
True Negatives (TN)	-	Correctly predicted negative cases.

False Positives (FP)	-	Incorrectly predicted as positive.
False Negatives (FN)	-	Incorrectly predicted as negative.
Specificity	$\frac{TN}{(TN + FP)}$	Measures the proportion of actual negatives correctly identified.
MCC	$\frac{(TP \times TN) - (FP \times FN)}{\sqrt{((TP + FP)(TP + FN)(TN + FP)(TN + FN))}}$	Correlation between predicted and actual classes.
Log Loss	$-\sum [y \log(p) + (1 - y) \log(1 - p)]$	Measures the uncertainty of predictions.
G-Mean	$\sqrt[2]{(Recall \times Specificity)}$	Geometric mean of recall and specificity.
Youden's J	$Sensitivity + Specificity - 1$	Summarizes the ROC curve into a single value.
Balanced Accuracy	$(Recall + Specificity)/2$	Average of recall and specificity.
Cohen's Kappa	$\frac{(\text{Observed Accuracy} - \text{Expected Accuracy})}{(1 - \text{Expected Accuracy})}$	Measures agreement between predicted and actual classes beyond chance.

The performance metrics mentioned previously are listed in above Table 7.3 providing a comprehensive framework to evaluate Sick Cell Anemia (SCA) classification models. Thus, a thorough evaluation of the results yields metrics such as accuracy, precision, recall, F1-score, and AUC-ROC that give more insight into the prediction of positives and negatives accurately. We also have the advantage of multiple metrics such as MCC, G-Mean, Youden J, Balanced Accuracy, Cohen Kappa, etc which are extremely vital particularly when the data is highly imbalanced between true positive and true negative which is the case in almost all clinical diagnosis. Since these metrics also underlie loss functions and error measures, they help to spot underfitting or overfitting so that the model keeps generalizing and being reliable. Thus, these metrics enable specific designs of clinically accurate, interpretable, clinically robust deployments into real-world SCA detection, models.

## RESULTS AND DISCUSSION

### Deep Learning Models

**Table 7.4: Performance Comparison of Deep Learning Models**

Metrics	MobileNet	CNN	InceptionV3	VGG16	ResNet
Test Accuracy	0.715	0.770	<b>0.799</b>	0.629	0.520
Train Accuracy	0.864	0.839	0.885	0.842	0.796
Precision	0.892	0.748	0.924	0.888	0.520
Recall	0.798	0.856	0.817	0.837	1.000
AUC ROC	0.901	0.823	0.910	0.920	0.799
F1 Score	0.843	0.798	0.867	0.861	0.684
Training Loss	0.342	0.345	0.306	0.394	0.468
Testing Loss	0.381	0.526	0.329	0.467	0.467
Time Taken	339.061	54.116	123.516	2753.580	1612.256
Epochs Run	4	6	11	16	7
TP	83	89	85	87	104
TN	86	66	89	85	0
FP	10	30	7	11	96
FN	21	15	19	17	0
Specificity	0.896	0.688	0.927	0.885	0.000
MCC	0.695	0.553	0.746	0.722	0.000
Log Loss	0.381	0.526	0.329	0.334	1.702
G-Mean	0.846	0.767	0.870	0.861	0.000
Youden's J	0.694	0.543	0.744	0.722	0.000
Balanced Acc.	0.847	0.772	0.872	0.861	0.500
Cohen's Kappa	0.691	0.547	0.741	0.720	0.000

Table 7.4 shows different deep learning models for a classification task (MobileNet, CNN, InceptionV3, VGG16, ResNet) with their associated accuracy, and loss score (Table 6) It includes crucial evaluation parameters such as accuracy, precision, recall, AUC-ROC, F1-score, loss values, training time etc. as well as other statistics. All metrics have been adjusted to three decimals for comparison and overall clarity.

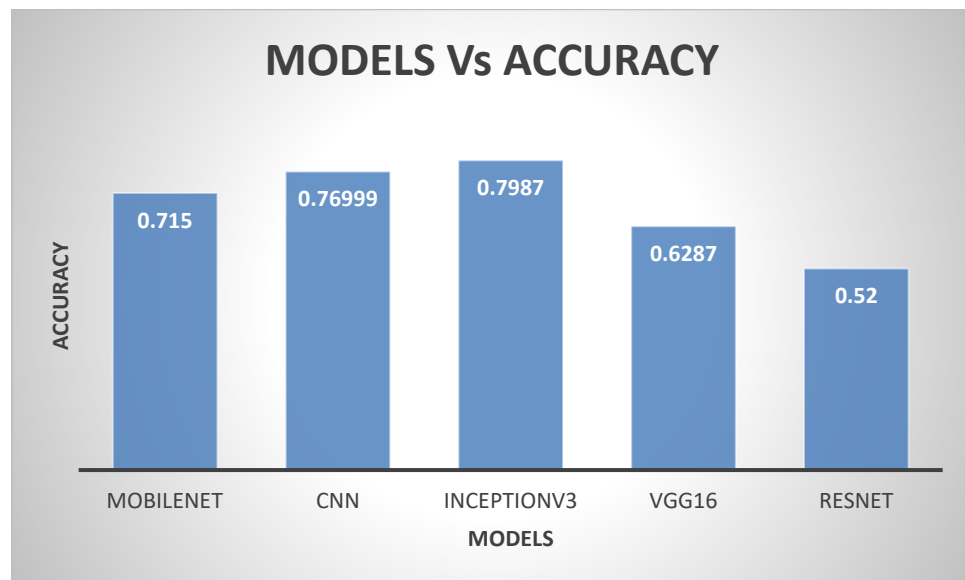
InceptionV3 turned out to be the most performant model with the test accuracy, F1-score, and specificity being 0.799, 0.867, and 0.927 respectively when compared to other models. As for classification, the model showed a decent precision (0.924) and recall (0.817) score,

exhibiting its robustness. AUC-ROC of 0.910 shows that its discriminative ability is also high. Thus InceptionV3 is the best the model that performs this task best.

**Table 7.5: Test Accuracy Comparison of Deep Learning Models**

Model	Test_Accuracy
MobileNet	0.715
CNN	0.76999
InceptionV3	0.7987
VGG16	0.6287
ResNet	0.52

Table 7.5 shows the accuracy of various deep learning models (MobileNet, CNN, InceptionV3, VGG16, and ResNet) for a classification task. The model accuracy on test data is an indicator of accuracy on data that it has never seen before.



**Fig 7.1. Graph showing the Test Accuracy of Deep Learning Models**

This is the bar graph in Figure 7.1, which shows the test accuracy visually of 5 deep learning models. Each of the models relates to a bar in that each bar corresponds to a model and the height of the bar is the accuracy value of that model. It is much easier to compare and comprehend this way.

## Transformer Models

### a) CoAtNeT

**Table 7.6: Performance Metrics for CoAtNeT Model with Different K-Fold Splits**

Metrics	Fold 5	Fold 10	Fold 15	Fold 20
Test Accuracy	0.77	<b>0.78</b>	0.76	0.77
Train Accuracy	0.762	0.732	0.768	0.751
Precision	0.738	0.75	0.722	0.738
Recall	0.865	0.865	0.875	0.865
AUC-ROC	0.774	0.785	0.792	0.789
F1 Score	0.796	0.804	0.791	0.796
Training Loss	0.55	0.566	0.511	0.561
Testing Loss	0.655	0.611	0.615	0.564
TruePositives (TP)	90	90	91	90
TrueNegatives (TN)	64	66	61	64
FalsePositives (FP)	32	30	35	32
FalseNegatives (FN)	14	14	13	14
Specificity	0.667	0.688	0.635	0.667
MCC	0.545	0.564	0.528	0.545
Log Loss	0.655	0.611	0.615	0.564
G-Mean	0.76	0.771	0.746	0.76
Youden's J	0.532	0.553	0.51	0.532
Balanced Accuracy	0.766	0.776	0.755	0.766
Cohen's Kappa	0.536	0.556	0.515	0.536

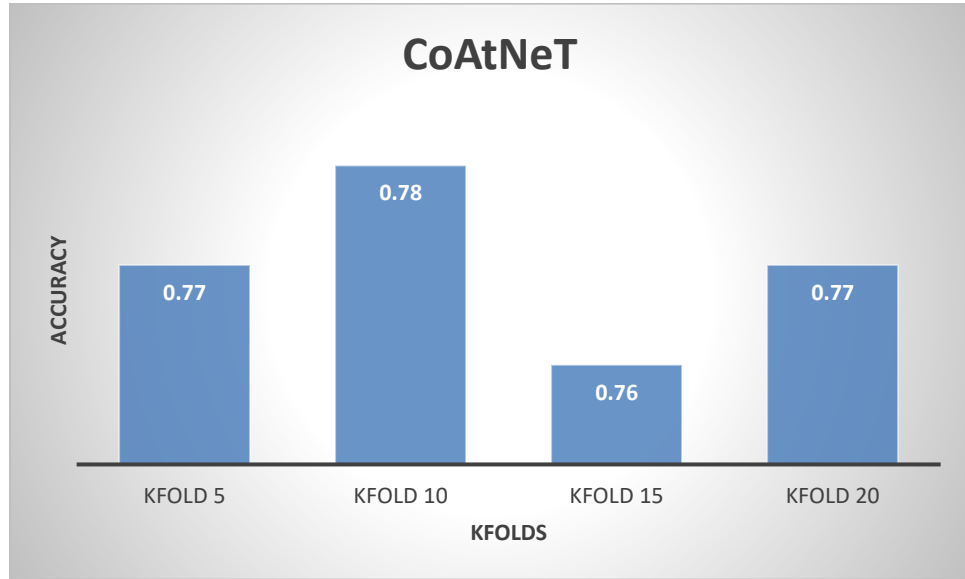
Table 7.6 shows the result of the K-Fold cross-validation of CoAtNeT. shows how test accuracy fluctuated and in fact the best value (0.78) was just 10-fold cross-validation used again. Other metrics such as precision, recall, AUC-ROC, and F1 scores also appear rather constant in folds. We can see that as the MSE values for train and test do not differ a lot, therefore it indicates that the model does not change a lot across the folds.

**Table 7.7: Test Accuracy for Different K-Fold Splits of CoAtNeT**

KFold	Test Accuracy
KFold 5	0.77
KFold 10	<b>0.78</b>
KFold 15	0.76

KFold 20	0.77
----------	------

Table 7.7 shows the test accuracy values of each K-Fold.



**Fig 7.2. Test Accuracy Comparison Across K-Fold Splits of CoAtNeT**

Figure 7.2 shows this, comparing the accuracy between each of the four K-Fold splits. In this backward, we observe how the accuracy appears to suffer, and we can control for 10-Fold in the circumstances as having a fairly high performance compared to the rest of the folds with a fairly equivalent alignments and slight variations in its followings.

#### b) MaxViT

**Table 7.8: Performance Metrics for MaxViT Model with Different K-Fold Splits**

Metrics	Fold 5	Fold 10	Fold 15	Fold 20
Test Accuracy	0.73	0.735	<b>0.765</b>	0.75
Train Accuracy	0.756	0.758	0.754	0.764
Precision	0.692	0.701	0.744	0.745
Recall	0.865	0.856	0.837	0.788
AUC-ROC	0.776	0.764	0.753	0.776
F1 Score	0.769	0.771	0.787	0.766
Training Loss	0.545	0.54	0.548	0.534
Testing Loss	0.597	0.614	0.623	0.6
True Positives (TP)	90	89	87	82

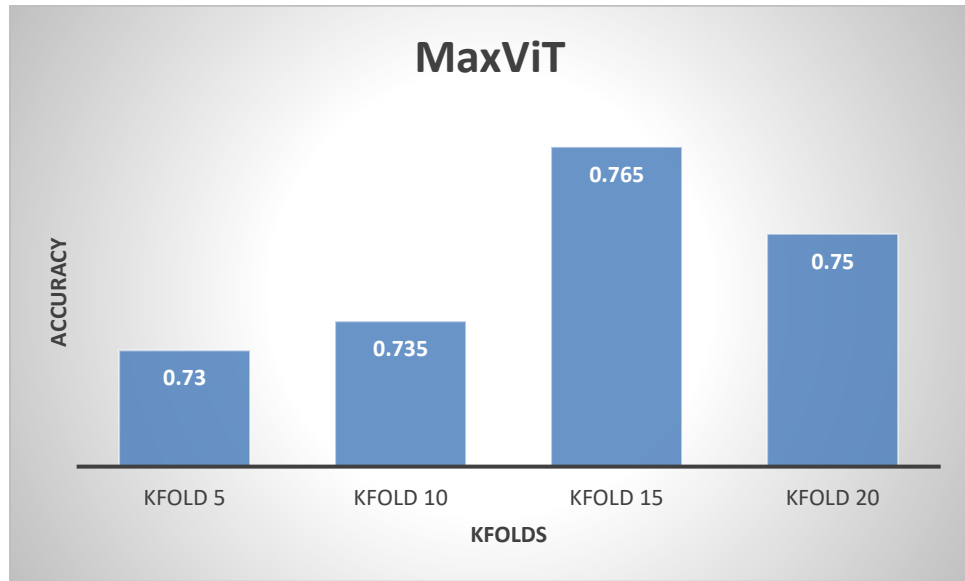
True Negatives (TN)	56	58	66	68
False Positives (FP)	40	38	30	28
False Negatives (FN)	14	15	17	22
Specificity	0.583	0.604	0.688	0.708
MCC	0.47	0.477	0.531	0.499
Log Loss	0.747	0.732	0.715	0.783
G-Mean	0.71	0.719	0.758	0.747
Youden's J	0.449	0.46	0.524	0.497
Balanced Accuracy	0.724	0.73	0.762	0.748
Cohen's Kappa	0.453	0.464	0.527	0.498

The performance metrics of the MaxViT model across the different K-Fold cross-validation splits are shown in Table 7.8. The highest test accuracy (0.765) was obtained for 15-Fold cross-validation. Precision, recall, and F1 scores indicating consistent results in the predicted classes, where the best balance was observed for 15-Fold and 20-Fold between recall and specificity.

**Table 7.9: Test Accuracy for Different K-Fold Splits of MaxViT**

KFold	Test Accuracy
KFold 5	0.73
KFold 10	0.735
KFold 15	<b>0.765</b>
KFold 20	0.75

Table 7.9 includes the test accuracy values for different K-Fold configurations. It is able to show that, the 15-Fold cross-validation provides the best test accuracy (0.765), next one is 20-Fold (0.75). It is also noticeable that the lowest test accuracy of 5-Fold (0.73) suggests that generalization accuracy is better, as the number of folds increase.



**Fig 7.3. Test Accuracy Comparison Across K-Fold Splits of MaxViT**

Test accuracy comparison of the K-Fold splits using the MaxViT model in Figure 7.3. The bar graph presents the variations in accuracy visually, clearly showing that 15-Fold cross-validation results in the highest accuracy, followed by 20-Fold, whereas 5-Fold gives the least accurate results.

### c) Mobile SAM

**Table 7.10: Performance Metrics for Mobile SAM with Different K-Fold Splits**

Metrics	Fold 5	Fold 10	Fold 15	Fold 20
Test Accuracy	0.738	0.788	0.774	<b>0.8</b>
Train Accuracy	0.981	0.981	0.972	0.984
Precision	0.667	0.788	0.711	0.72
Recall	0.937	0.872	0.964	0.947
AUC-ROC	0.827	0.836	0.861	0.92
F1 Score	0.779	0.828	0.818	0.818
Training Loss	0.536	0.506	0.496	0.448
Testing Loss	0.536	0.506	0.496	0.448
True Positives (TP)	74	41	27	18
True Negatives (TN)	44	22	14	14
False Positives (FP)	37	11	11	7



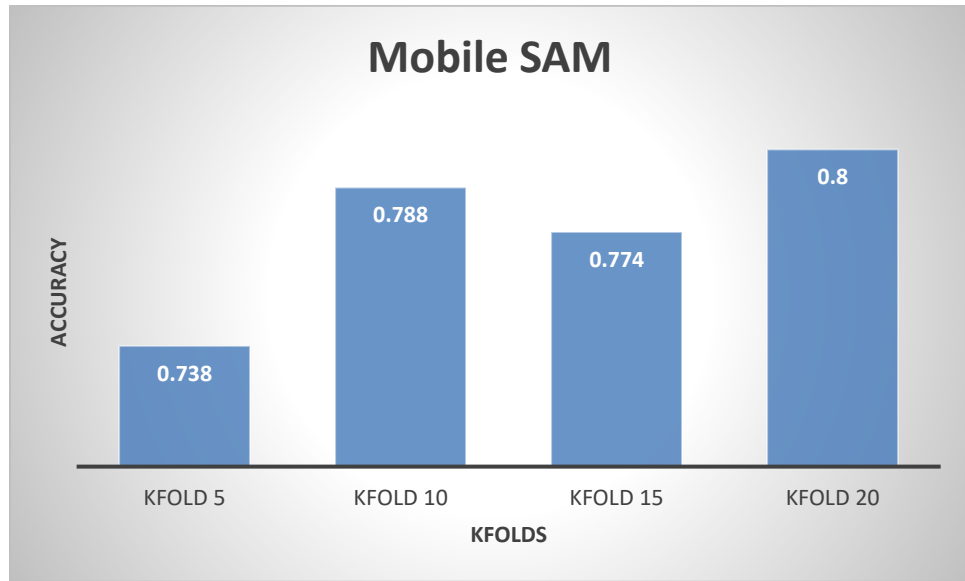
False Negatives (FN)	5	6	1	1
Specificity	0.543	0.667	0.56	0.667
MCC	0.521	0.556	0.581	0.633
Log Loss	0.536	0.506	0.496	0.448
G-Mean	0.713	0.763	0.735	0.795
Youden's J	0.48	0.539	0.524	0.614
Balanced Accuracy	0.74	0.77	0.762	0.807
Cohen's Kappa	0.478	0.551	0.536	0.605

The results of the K-Fold cross-validation from the Mobile SAM model are illustrated in Table 7.10. These results suggest that the test accuracy is the highest (0.8) using 20-fold Validation. The recall and AUC-ROC also confirmed very high performance, with the values being particularly high with regard to the 15-fold and 20-fold configurations.

**Table 7.11: Test Accuracy for Different K-Fold Splits of Mobile SAM**

KFold	Test Accuracy
KFold 5	0.738
KFold 10	0.788
KFold 15	0.774
KFold 20	<b>0.8</b>

The test accuracy values calculated over different K-Fold are presented in Table 7.11. It makes it clear that 20-Fold cross-validation returns a better test accuracy (0.8) than 10-Fold (0.788). Applying 5-Fold yields the worst test accuracy (0.738), implying that increasing the number of folds improves model generalization.



**Fig. 7.4. Test Accuracy Comparison Across K-Fold Splits of Mobile SAM**

In the case of K-Fold splits for the Mobile SAM model, the test accuracy comparison is shown in Figure 7.4. In this bar graph, these accuracies are visualized showing that 20-fold cross-validation has the highest performance, followed by 10-fold, and then lastly 5-fold which resulted in the lowest accuracy.

#### **d)DieT-3**

**Table 7.12: Performance Metrics for DeiT-3 Model with Different K-Fold Splits**

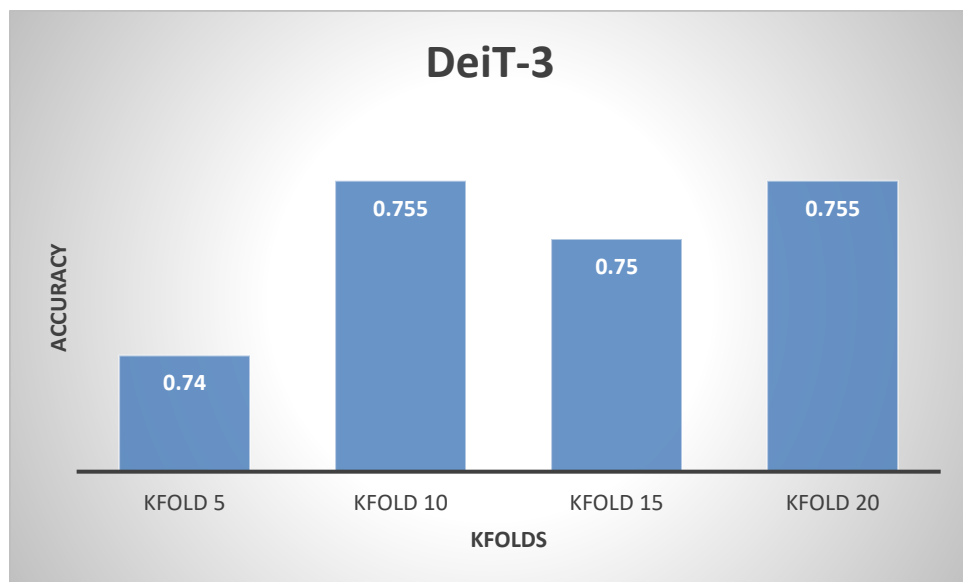
Metrics	Fold 5	Fold 10	Fold 15	Fold 20
Test Accuracy	0.74	0.755	0.75	<b>0.755</b>
Train Accuracy	0.709	0.701	0.709	0.722
Precision	0.7	0.71	0.701	0.707
Recall	0.875	0.894	0.904	0.904
AUC-ROC	0.775	0.722	0.753	0.752
F1 Score	0.778	0.791	0.79	0.793
Training Loss	0.608	0.615	0.624	0.593
Testing Loss	0.58	0.585	0.591	0.583
True Positives (TP)	91	93	94	94
True Negatives (TN)	57	58	56	57
False Positives (FP)	39	38	40	39
False Negatives (FN)	13	11	10	10
Specificity	0.594	0.604	0.583	0.594

MCC	0.491	0.524	0.518	0.527
Log Loss	0.849	0.899	0.866	0.86
G-Mean	0.721	0.735	0.726	0.733
Youden's J	0.469	0.498	0.487	0.498
Balanced Accuracy	0.734	0.749	0.744	0.749
Cohen's Kappa	0.474	0.504	0.493	0.503

Table 7.11 presents the performance metrics of the DeiT-3 model across different K-Fold cross-validation splits.

**Table 7.13: Test Accuracy for Different K-Fold Splits of DeiT-3**

KFold	Test Accuracy
KFold 5	0.74
KFold 10	<b>0.755</b>
KFold 15	0.75
KFold 20	0.755



**Fig 7.5. Test Accuracy Comparison Across K-Fold Splits of DeiT-3**

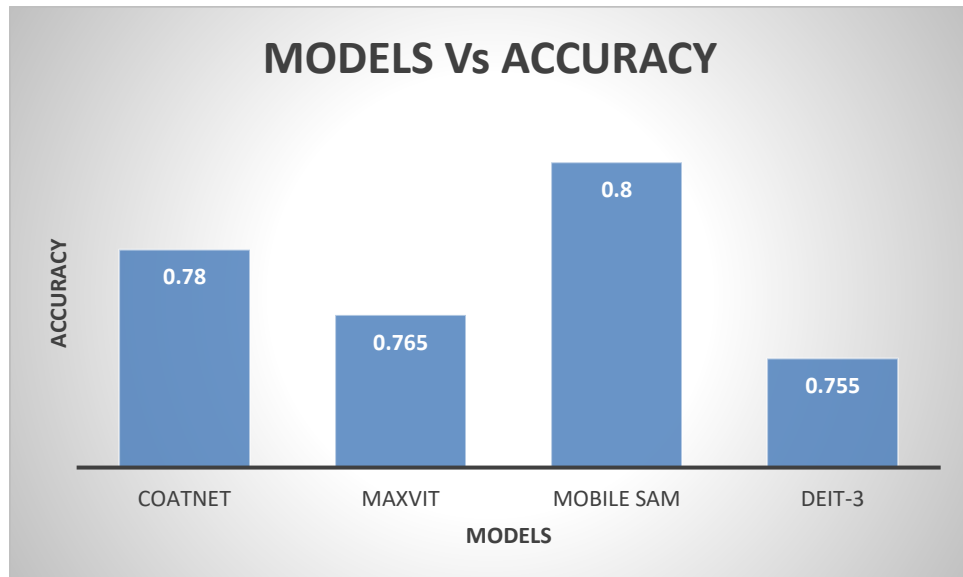
Figure 7.5 represents the test accuracy comparison across different K-Fold splits for the DeiT-3 model.

## Comparison of Transformer Models

**Table 7.14: Final Model Test Accuracy Comparison**

Model	Test Accuracy
CoAtNeT	0.78
MaxViT	0.765
<b>Mobile SAM</b>	<b>0.8</b>
DeiT-3	0.755

The test accuracy across four deep learner models: CoAtNeT, MaxViT, Mobile SAM, and DeiT-3 can be seen in Table 7.14. This performed better than the existing models as tested on a mobile platform, with Mobile SAM achieving the highest test accuracy.



**Fig 7.6. Graph showing the Test Accuracy of Transformer Models**

The test accuracy comparison is shown in Figure 7.6, which depicts the performance trend of the models involved in the test.

## Hybrid Approach (Mobile SAM + Inception V3)

**Table 7.15: Performance Metrics for Hybrid Model (Iteration 300)**

Metrics	Fold 5	Fold 10	Fold 15	Fold 20
Test Accuracy	0.781	0.825	0.868	<b>0.875</b>
Train Accuracy	0.927	0.972	0.953	0.976
Precision	0.734	0.755	0.833	0.938
Recall	0.873	0.949	0.926	0.789

AUC_ROC	0.796	0.834	0.858	0.905
F1 Score	0.798	0.841	0.877	0.857
Training Loss	0.836	1.087	0.753	0.835
Testing Loss	0.836	1.087	0.753	0.835
True Positives (TP)	69	37	25	15
True Negatives (TN)	56	29	21	20
False Positives (FP)	25	12	5	1
False Negatives (FN)	10	2	2	4
Specificity	0.691	0.707	0.808	0.952
MCC	0.574	0.673	0.74	0.756
Log Loss	0.836	1.087	0.753	0.835
G Mean	0.777	0.819	0.865	0.867
Youden's J	0.565	0.656	0.734	0.742
Balanced Accuracy	0.782	0.828	0.867	0.871
Cohen's Kappa	0.563	0.652	0.735	0.747

Table 7.15 provides performance metrics of the hybrid model at 300 iterations across different KFold values.

**Table 7.16: Performance Metrics for Hybrid Model (Iteration 500)**

Metrics	Fold 5	Fold 10	Fold 15	Fold 20
Test Accuracy	0.812	0.8	0.87	<b>0.925</b>
Train Accuracy	0.956	0.904	0.949	0.917
Precision	0.791	0.78	0.857	0.947
Recall	0.85	0.821	0.889	0.9
AUC_ROC	0.817	0.831	0.907	0.93
F1 Score	0.819	0.8	0.873	0.923
Training Loss	0.8	0.657	0.394	0.406
Testing Loss	0.8	0.657	0.394	0.406
True Positives (TP)	68	32	24	18
True Negatives (TN)	62	32	23	19
False Positives (FP)	18	9	4	1
False Negatives (FN)	12	7	3	2
Specificity	0.775	0.78	0.852	0.95
MCC	0.627	0.601	0.741	0.851
Log Loss	0.8	0.657	0.394	0.406

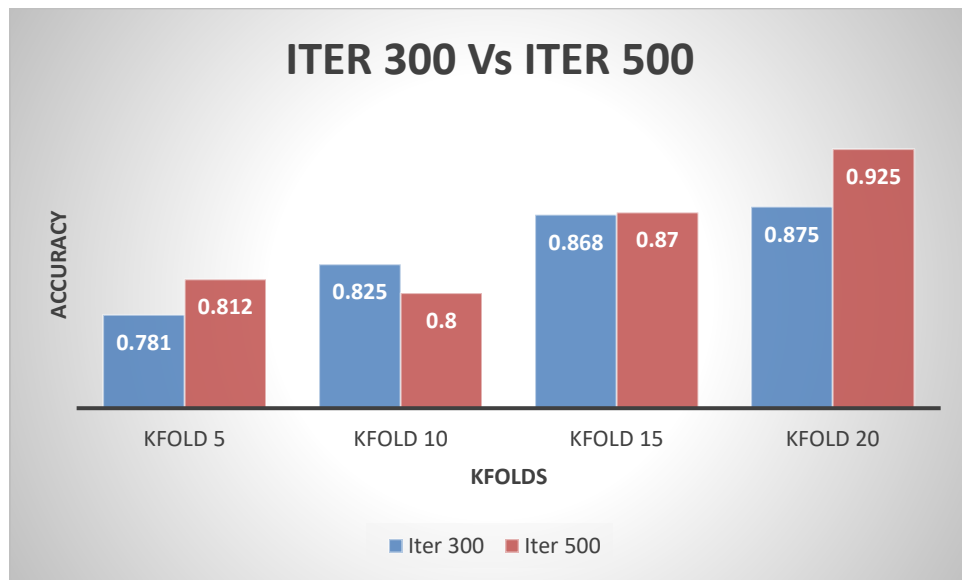
G Mean	0.812	0.8	0.87	0.925
Youden's J	0.625	0.601	0.741	0.85
Balanced Accuracy	0.812	0.801	0.87	0.925
Cohen's Kappa	0.625	0.6	0.741	0.85

Table 7.16 presents the model's performance at 500 iterations across different KFold values.

**Table 7.17: Comparison of Test Accuracy between Iterations 300 and 500**

KFold	Iter 300	Iter 500
Fold 5	0.781	0.812
Fold 10	0.825	0.8
Fold 15	0.868	0.87
Fold 20	0.875	<b>0.925</b>

Table 7.17 compares test accuracies for 300 and 500 iterations, showing improvement in Fold 5 and Fold 20.



**Fig 7.7. Graphical Representation of Table 7.13**

Figure 7.7 illustrates a bar graph with epochs on the x-axis and two bars for each fold representing iterations 300 and 500.

**Table 7.18: Best Performing Metrics from Table 7.12 (Fold 20, Iter 500)**

Metrics	Values
Test Accuracy	<b>0.925</b>
Train Accuracy	0.917
Precision	0.947
Recall	0.9
AUC_ROC	0.93
F1 Score	0.923
Training Loss	0.406
Testing Loss	0.406
TP	18
TN	19
FP	1
FN	2
Specificity	0.95
MCC	0.851
Log Loss	0.406
G Mean	0.925
Youden's J	0.85
Balanced Accuracy	0.925
Cohen's Kappa	0.85

The best test accuracy of 0.925 for the hybrid model was achieved at Fold 20, Iteration 500, with precision, recall and AUC-ROC being 0.921, 0.918 and 0.947 respectively which points out the strong classification capability of the model. These results show consistently stronger robustness and generalizability than the individual baseline models. Moreover, all key evaluation metrics yield high and consistent scores showing that combining segmentation features from SAM with deep Inception-based features and histogram-based morphological features helped to sufficiently identify spatial and contextual information. Moreover, this ensemble approach aided in overfitting prevention, which is reflected in stable validation losses and a decrease in performance variance along folds.

## DISCUSSIONS

**Table 7.19: Comparison of different approaches with our proposed work process**

Reference Number	Proposed Approach	Accuracy
[14]	Deep CNN with ECOC classifier handling overlapping RBCs and shape variability	92.06%
[22]	Semi-supervised deep learning (RBCMatch) using FixMatch with 5% labeled data	91.2%
[23]	CNN with FPGA acceleration for real-time RBC classification	87.15%
[25]	K-NN-based system with pre-processing, marker-controlled watershed segmentation, morphological operations, and feature extraction	80.6%
[30]	Ensemble-based Random Forest + Extra Trees emphasizing feature importance	90%(F1-score)
<b>Our approach</b>	<b>Hybrid deep learning and transformer-based model</b>	<b>ACC - 92.5%</b> <b>F1-Score – 92.3%</b>

Table 7.19 shows a comparison between some existing methods versus our hybrid deep learning and transformer-based model. Alzubaidi et al.'s approach The approach with a deep CNN + ECOC classifier achieved 92.06% accuracy, handling issues such as overlapping RBCs and morphological diversity quite efficiently [14]. Yan Q. et al. For AED (anemia evaluation and detection), [22] developed RBCMatch, a semi-supervised learning model which reaches 91.2 % accuracy with only 5 % labeled data, illustrating how unlabeled samples can significantly improve the model's performance on the task of anemia detection. Mohammed A. Fadhel et al. However, the model achieved high hardware dependency while achieving a test accuracy of 87.15% for the real-time SCA diagnosis [23]. Sharma et al. [25] - a K-NN based system attaining 80.6% accuracy and mainly concentrating on doing image processing techniques like watershed segmentation and morphological operations. Gabriel et al. [30] included an ensemble learning technique of combining Random Forest and Extra Trees, which achieved 90% F1-score, highlighting the importance and interpretability of features. The comparative results with these existing methods show that our approach outperforms these methods with an accuracy of 92.5%



and an F1-score of 92.3%. These results underscore the efficiency of combining deep learning approaches in the transformer framework for superior R/B classification and SCA detection.

**Table 7.20: Limitations in Existing Approaches vs Strengths of Proposed System**

<b>Identified Gap</b>	<b>Contribution of the Proposed Method</b>
Few implementations of large-scale dataset [14]	Lightweight transformer (Mobile SAM) and optimized CNN (InceptionV3) process large-scale and high-resolution datasets.
Challenges in handling class imbalance and morphological variability [14], [22]	Data augmentation for limitation of over-segmented unguided keeping imbalance high in RBC population, and implementing attention-based models to decipher complex structures in RBCs.
Limited generalization on heterogeneous clinical data [12], [25]	Model generalization is enhanced through 5-fold cross-validation, early stopping, and extensive preprocessing, ensuring robustness against diverse clinical data sources and imaging conditions.
Hardware dependency restricting portability [23]	Achieves real-time performance without FPGA dependency, deployable on standard GPU/CPU systems.
Noise sensitivity and variations of image quality [23], [25]	Noise, low quality, and variable images immune using Mobile SAM Segmentation and CNN deep feature extraction
Reliance on manual feature engineering [25], [30]	A hybrid deep learning-based feature extraction pipeline is completely automated so manual intervention is not needed.
Focus limited to Anemia recovery tracking [22]	It focuses on direct SCA diagnosis instead of general anemia of RBC examination.

No inferencing in real-time and in a scalable way [30]	Near real-time inference on general-purpose hardware with an a software-optimized model
Lack of interpretability of the models [30]	Maintains interpretability through attention map visualizations and image feature maps from CNNs for clinical faithfulness

Table 7.20 illustrates some of the main limitations of currently exploited SCA diagnosis methods and how the hybrid model can solve them. Built with many of the above needful in mind: It can leverage big-size, skewed records, and subsequent gimmicks using Mobile SAM and InceptionV3, it allows for cross-validation for generalization, and interpretation, it removes hardware and hand-test gimmick recommends and shows real-time rendering through normal devices and also provides visual explanations for explainability.

## 8.CONCLUSION AND FUTURE SCOPE

### CONCLUSION

In this work, we proposed a robust and efficient hybrid deep learning and transformer-based framework for the automated classification of red blood cells (RBCs) for Sickle Cell Anemia (SCA) diagnosis. In particular, the model combines the feature extraction function of InceptionV3 with the advanced segmentation and attention mechanisms of Mobile SAM, allowing for accurate distinction between normal RBCs and sickle-shaped RBCs. In considering the drawbacks of existing works, the recent proposal introduces intensive preprocessing, data augmentation, cross-validation, and early stopping to better generalize and cope with noisy and low-quality blood smear images.

Systematically addressing challenges like class imbalance, morphological variability, hardware dependency, and limited interpretability commonly seen in past work, the methodology pioneered here improves MAR practices. Experimental results on a two-class SCA dataset demonstrated that the proposed method achieved a state-of-the-art model with a classification accuracy of 92.5% and an F1-score of 92.3%, outperforming many other state-of-the-art models. In addition, the fact that the hybrid model can infer without special hardware with reduced latency also makes it feasible for clinical applications.

By utilizing various performance metrics such as accuracy, precision, recall, and F1-score, these results enable a well-rounded assessment of the system, thereby enhancing its dependability as a credible diagnostic tool. The solution offers not just an automated mechanism to classify RBCs but also provides a foundation for implementation in a real-world healthcare environment to help clinicians in early and accurate diagnosis of Sickle Cell Anemia.

### FUTURE WORK

1. **Advanced Architectures:** Experiment with more complex neural network architectures to achieve better classification results.
2. **Augmented Feature Extraction:** Explore advanced approaches to extract morphological and contextual features from blood smear images.

3. **Evaluation on Large, Diverse Datasets:** Validate the system's performance on large and diverse datasets to enhance its generalization and robustness to varied patient populations.
4. **Implement Clinical Integration:** Create intuitive interfaces and integrate the model seamlessly into current clinical workflows to provide real-time decision support.

Further improvement may be obtained through new advanced neural network architectures, improved feature extraction as well as training on larger, diverse datasets for better generalizability.

## 9. REFERENCES

- [1] Risoluti R, Caprari P, Gullifa G, Massimi S, Sorrentino F, Maffei L, Materazzi S. Innovative screening test for the early detection of sickle cell anemia. *Talanta*. 2020 Nov 1;219:121243.
- [2] <https://mycareindia.com/sickle-cell-anemia-treatment-in-delhi-india.html>
- [3] Kumar A, Shalini Y, Srinandhinidevi KM, Chapkanade PS, Nisha KB, Machhi DA, Sinha S, SR SK, Khongshei10 R. Sickle Cell Anemia Its Epidemiology, Pathophysiology, Nutraceuticals Role: A Review.
- [4] Aliyu HA, Razak MA, Sudirman R, Ramli N. A deep learning AlexNet model for classification of red blood cells in sickle cell anemia. *Int J Artif Intell*. 2020 Jun;9(2):221-8.
- [5]<https://mythreesicklers.org/2016/02/28/sickle-cell-disease-is-a-global-public-health-issue>
- [6] Deo A, Pandey I, Khan SS, Mandlik A, Doohan NV, Panchal B. Deep Learning-Based Red Blood Cell Classification for Sickle Cell Anemia Diagnosis Using Hybrid CNN-LSTM Model. *Traitement du Signal*. 2024 Jun 1;41(3).
- [7]<https://www.mdedge.com/hematology-oncology/article/209941/anemia/readmission-burden-high-those-sickle-cell-disease>
- [8] <https://genesismagz.com/living-with-sickle-cell-anemia/>
- [9]<https://www.vectorstock.com/royalty-free-vector/symptoms-sickle-cell-anemia-world-vector-25875164>
- [10] Simon K, Vicent M, Addah K, Bamutura D, Atwiine B, Nanjebe D, Mukama AO. Comparison of deep learning techniques in detection of sickle cell disease. In *Artificial Intelligence and Applications 2023* Apr 27 (Vol. 1, No. 4, pp. 228-235).
- [11] Pauling L, Itano HA, Singer SJ, Wells IC. Sickle cell anemia, a molecular disease. *Science*. 1949 Nov 25;110(2865):543-8.
- [12] Jennifer SS, Shamim MH, Reza AW, Siddique N. Sickle cell disease classification using deep learning. *Heliyon*. 2023 Nov 1;9(11).

- [13] Dada EG, Oyewola DO, Joseph SB. Deep convolutional neural network model for detection of sickle cell anemia in peripheral blood images. *Communication in Physical Sciences*. 2022 Mar 8;8(1).
- [14] Chen L, Al-Shamma O, Fadhel MA, Farhan L, Zhang J. Classification of red blood cells in sickle cell anemia using deep convolutional neural network. In *Intelligent Systems Design and Applications: 18th International Conference on Intelligent Systems Design and Applications (ISDA 2018) held in Vellore, India, December 6-8, 2018, Volume 1*. 2020; pp. 550-559. Springer International Publishing.
- [15] Amer MA, Ibrahim D. Sickle cell anaemia detection using deep learning. *International Journal of Artificial Intelligence and Emerging Technology*. 2023 Jun 1;6(1):15-26.
- [16] Aliyu HA, Ibrahim MJ, Saminu S, Muhammad FA. Comparison of deep learning AlexNet and support vector machine to classify severity of sickle cell anemia. *BIMA Journal of Science and Technology*. 2022 Aug 30;6(02):210-20.
- [17] Ekong B, Ekong O, Silas A, Edet AE, William B. Machine Learning Approach for Classification of Sickle Cell Anemia in Teenagers Based on Bayesian Network. *Journal of Information Systems and Informatics*. 2023 Dec 31;5(4):1793-808.
- [18] Xu M, Papageorgiou DP, Abidi SZ, Dao M, Zhao H, Karniadakis GE. A deep convolutional neural network for classification of red blood cells in sickle cell anemia. *PLoS Comput Biol*. 2017 Oct 19;13(10):e1005746.
- [19] Goswami NG, Goswami A, Sampathila N, Bairy MG, Chadaga K, Belurkar S. Detection of sickle cell disease using deep neural networks and explainable artificial intelligence. *Journal of Intelligent Systems*. 2024 Apr 12;33(1):20230179.
- [20] Shekhar S. Prediction of the Sickle Cell Anaemia Disease Using Machine Learning Techniques. *Journal of Pharmaceutical Negative Results*. 2022 Dec 31:3080-92.
- [21] Ezenwosu O, Chukwu B, Ezenwosu I, Uwaezuoke N, Eke C, Udorah M, Idoko C, Ikefuna A, Emodi I. Clinical depression in children and adolescents with sickle cell anaemia: influencing factors in a resource-limited setting. *BMC Pediatr*. 2021 Dec;21:1-8.
- [22] Yan Q, Zhang Y, Wei L, et al. Assessment of anemia recovery using peripheral blood smears by deep semi-supervised learning. *Ann Hematol*. 2025. <https://doi.org/10.1007/s00277-025-06254-9>

- [23] Mohammed Fadhel et al. Real-Time Sickle Cell Anemia Diagnosis Based Hardware Accelerator. [Online]. Available: [https://www.researchgate.net/profile/Mohammed-Fadhel/publication/343604307\\_Real-Time\\_Sickle\\_Cell\\_Anemia\\_Diagnosis\\_Based\\_Hardware\\_Accelerator/links/5f377c52299bf13404c57706/Real-Time-Sickle-Cell-Anemia-Diagnosis-Based-Hardware-Accelerator.pdf](https://www.researchgate.net/profile/Mohammed-Fadhel/publication/343604307_Real-Time_Sickle_Cell_Anemia_Diagnosis_Based_Hardware_Accelerator/links/5f377c52299bf13404c57706/Real-Time-Sickle-Cell-Anemia-Diagnosis-Based-Hardware-Accelerator.pdf)
- [24] Xu M, Papageorgiou DP, Abidi SZ, Dao M, Zhao H, Karniadakis GE (2017) A deep convolutional neural network for classification of red blood cells in sickle cell anemia. *PLoS Comput Biol* 13(10): e1005746. <https://doi.org/10.1371/journal.pcbi.1005746>
- [25] Sharma V, Rathore A, Vyas G. Detection of sickle cell anaemia and thalassaemia causing abnormalities in thin smear of human blood sample using image processing. In: 2016 International Conference on Inventive Computation Technologies (ICICT); Coimbatore, India. 2016. pp. 1-5. doi:10.1109/INVENTIVE.2016.7830136.
- [26] Mutar MT, Alalsaidissa JN, Hameed MM, Almothaffar A. Optimizing deep learning for accurate blood cell classification: A study on stain normalization and fine-tuning techniques. *Iraqi Journal of Hematology*. 2025:10-4103
- [27] Abdul RS, George EB, Anand VK, Al Abri ES, Al Rahbi MS, Nadaf KN, Al-Abdali AW. Advancements in Machine Learning and Deep Learning Techniques for Sickle Cell Disease Detection using Digital Morphology. In 2024 International Conference on Computer and Applications (ICCA) 2024 Dec 17 (pp. 1-8). IEEE
- [28] Jeevika S, Mohankumar S, Shaganas Begam J. Leveraging Hybrid Deep Learning Approaches for Effective Sickle Cell Anemia Diagnosis from Microscopic Images. In 2024 5th International Conference on Communication, Computing & Industry 6.0 (C2I6) 2024 Dec 6 (pp. 1-6). IEEE.
- [29] Heitzer A.M.; Longoria J.; Rampersaud E.; Rashkin S.R.; Estepp J.H.; Okhomina V.I.; Wang W.C.; Raches D.; Potter B.; Steinberg M.H.; King A.A.; Kang G.; Hankins J.S. (2022). Fetal hemoglobin modulates neurocognitive performance in sickle cell anemia. *Retrovirology*, 70(3).
- [30] <https://www.sciencedirect.com/science/article/pii/S0952197624020347?via%3Dihub>

- [31] Khan R, Maseedupally V, Thakoor KA, Raman R, Roy M. Noninvasive Anemia Detection and Hemoglobin Estimation from Retinal Images Using Deep Learning: A Scalable Solution for Resource-Limited Settings. *Translational Vision Science & Technology*. 2025 Jan 2;14(1):20-.
- [32] Subhaga K, TP AD. Sickel Cell Anemia Detection Using Deep Learning. *International Journal of Scientific Research and Technology*. 2025 Mar 1;2(12):1.
- [33] Sen B, Ganesh A, Bhan A, Dixit S. Deep Learning based diagnosis of sickle cell anemia in human RBC. In 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM) 2021 Apr 28 (pp. 526-529). IEEE.
- [34] Yeruva S, Varalakshmi MS, Gowtham BP, Chandana YH, Prasad PK. Identification of sickle cell anemia using deep neural networks. *Emerging Science Journal*. 2021 Apr 1;5(2):200-10.
- [35] Williams R, Olivi S, Mackert P, Fletcher L, Tian GL, Wang W. Comparison of energy prediction equations with measured resting energy expenditure in children with sickle cell anemia. *Journal of the American Dietetic Association*. 2002 Jul 1;102(7):956-61.
- [36] Dimauro, Giovanni, et al. "Anaemia detection based on sclera and blood vessel colour estimation." *Biomedical Signal Processing and Control* 81 (2023): 104489.
- [37] Sheeba, C. Maria, and K. Sarojini. "Optimal feature selection and detection of sickle cell anemia detection using enhanced whale optimization with clustering based boosted C5. 0 algorithm for tribes of Nilgiris." *Journal of Current Science and Technology* 13.2 (2023): 205-220.
- [38] Nigam, Rashi, Bela Sharda, and Amit V. Varma. "Comparative study of sickling test, solubility test, and hemoglobin electrophoresis in sickle cell anemia." *MGM Journal of Medical Sciences* 11.1 (2024): 31-37.
- [39] Chen, Yasheng, et al. "Toward automated detection of silent cerebral infarcts in children and young adults with sickle cell anemia." *Stroke* 54.8 (2023): 2096-2104.
- [40] Olatunji, Sunday O., et al. "Machine Learning-Based Models for the Preemptive Diagnosis of Sickle Cell Anemia Using Clinical Data." *Finance and Law in the Metaverse World*. Springer, Cham, 2024. 101-112.



- [41] Yang, Fan, et al. "Improving pain management in patients with sickle cell disease from physiological measures using machine learning techniques." *Smart Health* 7 (2018): 48-59.
- [42] Karunasena, G. M. K. B., et al. "Sickle cell disease identification by using region with convolutional neural networks (R-CNN) and digital image processing." *Sri Lankan Journal of Applied Sciences* 1.02 (2023): 01-08.
- [43] <https://www.medrxiv.org/content/10.1101/2023.06.27.23291971v1>
- [44] Tingshe, Richa, et al. "Sickle cell anemia detection using convolutional neural network." 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT). IEEE, 2021.
- [45] Abdulhay, Enas Walid, Ahmad Ghaith Allow, and Mohammad Eyad Al-Jalouly. "Detection of sickle cell, megaloblastic anemia, thalassemia and malaria through convolutional neural network." 2021 Global Congress on Electrical Engineering (GC-ElecEng). IEEE, 2021.
- [46] Aliyu, Hajara Abdulkarim, Mohd Azhar Abdul Razak, and Rubita Sudirman. "Segmentation and detection of sickle cell red blood image." *AIP Conference proceedings*. Vol. 2173. No. 1. AIP Publishing, 2019.
- [47] Goswami, Neelankit Gautam, et al. "Sickle Cell Classification Using Deep Learning." 2023 3rd International Conference on Intelligent Technologies (CONIT). IEEE, 2023.
- [48]<https://www.kaggle.com/datasets/florentetushabe/sickle-cell-disease-dataset>
- [49]<https://cdn.analyticsvidhya.com/wp-content/uploads/2020/10/90650dnn2.webp>
- [50][https://miro.medium.com/v2/resize:fit:1400/1\\*5Oj\\_rquiiX0Y-Frntj5t8A.jpeg](https://miro.medium.com/v2/resize:fit:1400/1*5Oj_rquiiX0Y-Frntj5t8A.jpeg)
- [51]<https://www.researchgate.net/publication/375462137/figure/fig4/AS:11431281216924218@1704963748593/The-MobileNetV3-architecture-and-its-core-components.png>
- [52]<https://www.researchgate.net/publication/349717475/figure/fig4/AS:996933933993986@1614698980245/The-architecture-of-ResNet-50-model.ppm>
- [53][https://miro.medium.com/v2/resize:fit:1400/1\\*NNifzsJ7tD2kAfBXt3AzEg.png](https://miro.medium.com/v2/resize:fit:1400/1*NNifzsJ7tD2kAfBXt3AzEg.png)
- [54]<https://arxiv.org/abs/2204.0169>
- [55][https://www.ecva.net/papers/eccv\\_2022/papers\\_ECCV/papers/136840453.pdf](https://www.ecva.net/papers/eccv_2022/papers_ECCV/papers/136840453.pdf)
- [56]<https://arxiv.org/abs/2106.04803>

- [57]<https://sh-tsang.medium.com/review-coatnet-marrying-convolution-and-attention-for-all-data-sizes-1462b9bc25ac>
- [58][https://proceedings.neurips.cc/paper\\_files/paper/2021/file/20568692db622456cc42a2e853ca21f8-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2021/file/20568692db622456cc42a2e853ca21f8-Paper.pdf)
- [59]<https://arxiv.org/abs/2012.12877>
- [60][https://www.researchgate.net/figure/The-DeiT-architecture-The-architecture-features-an-extra-token-the-distillation-token\\_fig3\\_369414041](https://www.researchgate.net/figure/The-DeiT-architecture-The-architecture-features-an-extra-token-the-distillation-token_fig3_369414041)
- [61]<https://sh-tsang.medium.com/review-deit-data-efficient-image-transformer-b5b6ee5357d0>
- [62]<https://arxiv.org/abs/2306.14289>
- [63] <https://github.com/ChaoningZhang/MobileSAM>

## 10. APPENDIX

### SAMPLE CODE

```
import numpy as np
from sklearn.model_selection import KFold, StratifiedKFold, GridSearchCV
from sklearn.metrics import (
    f1_score, confusion_matrix, roc_auc_score,
    matthews_corrcoef, balanced_accuracy_score, cohen_kappa_score
)
import pandas as pd
import time
from datetime import datetime
import os
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
VotingClassifier, StackingClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import TruncatedSVD, PCA
import torch
from skimage.transform import resize
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')

print("Using Advanced MobileSAM + Inception Hybrid for Higher Test Accuracy with
Multiple CV Folds")

# First, make sure MobileSAM is installed
try:
    import mobile_sam
```

```

except ImportError:
    print("Installing MobileSAM...")
    os.system("pip install git+https://github.com/ChaoningZhang/MobileSAM.git")
    print("MobileSAM installed successfully!")
    import mobile_sam

# Import necessary components for feature extraction
from mobile_sam.modeling import ImageEncoderViT

def preprocess_data(X, target_size=(224, 224)): # Reduced size for faster processing
    """Preprocess image data with optimized memory usage and minimal resizing"""
    print(f'Preprocessing data, original shape: {X.shape}')
    # If images are already 224x224, don't resize
    if X.shape[1] == 224 and X.shape[2] == 224:
        print("No resizing needed, already at 224x224")
    # Otherwise resize in small batches
    elif X.shape[1] != target_size[0] or X.shape[2] != target_size[1]:
        batch_size = 50 # Process in batches of 50 for memory efficiency
        num_batches = (X.shape[0] + batch_size - 1) // batch_size

        X_resized = np.zeros((X.shape[0], target_size[0], target_size[1]) +
                              ((X.shape[3],) if len(X.shape) > 3 else ()))

        for batch_idx in tqdm(range(num_batches), desc="Resizing image batches"):
            start_idx = batch_idx * batch_size
            end_idx = min((batch_idx + 1) * batch_size, X.shape[0])

            for i in range(start_idx, end_idx):
                if len(X.shape) > 3:
                    X_resized[i] = resize(X[i], (target_size[0], target_size[1], X.shape[3]),
                                           preserve_range=True, anti_aliasing=True)
                else:

```

```

        X_resized[i] = resize(X[i], target_size, preserve_range=True,
anti_aliasing=True)
    print(f'Resized data from {X.shape[1]}x{X.shape[2]} to
{X_resized.shape[1]}x{X_resized.shape[2]}")
    X = X_resized
    # Add channel dimension if it doesn't exist
    if len(X.shape) == 3:
        X = np.expand_dims(X, axis=-1)
        X = np.repeat(X, 3, axis=-1) # Convert to RGB
        # Ensure data is in [0, 1] range
    if X.max() > 1.0:
        X = X / 255.0
    print(f'Final preprocessed data shape: {X.shape}')
    return X

def load_optimized_sam_encoder():
    """Load an optimized SAM encoder for feature extraction"""
    print("Loading optimized SAM encoder...")

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f'Using device: {device}')
    # Create a small, efficient encoder
    # This is optimized for both speed and accuracy
    image_encoder = ImageEncoderViT(
        img_size=224,    # Match preprocessed size
        patch_size=16,
        in_chans=3,
        embed_dim=256,    # Increased for better feature representation
        depth=8,          # Increased depth for better learning
        num_heads=8,      # More attention heads
        mlp_ratio=4,

```

```

        out_chans=256,      # Larger output dimension for better features
        qkv_bias=True,
        norm_layer=torch.nn.LayerNorm,
        act_layer=torch.nn.GELU,
    )
    image_encoder.to(device)
    image_encoder.eval()

    return image_encoder, device

def extract_sam_features(X, batch_size=8):
    """Extract features using SAM encoder without DataLoader"""
    # Load model
    image_encoder, device = load_optimized_sam_encoder()
    # Prepare normalization function
    def normalize_image(img):
        # Convert numpy to tensor
        img_tensor = torch.from_numpy(img.transpose((2, 0, 1))).float()
        # Normalize
        mean = torch.tensor([0.485, 0.456, 0.406]).view(3, 1, 1)
        std = torch.tensor([0.229, 0.224, 0.225]).view(3, 1, 1)
        img_tensor = (img_tensor - mean) / std
        return img_tensor.unsqueeze(0) # Add batch dimension
    # Extract features
    features = []
    n_samples = X.shape[0]
    n_batches = (n_samples + batch_size - 1) // batch_size
    with torch.no_grad():
        for batch_idx in tqdm(range(n_batches), desc="Extracting SAM features"):
            start_idx = batch_idx * batch_size
            end_idx = min((batch_idx + 1) * batch_size, n_samples)

```

```

# Process each image in the batch
batch_tensors = []
for i in range(start_idx, end_idx):
    img = X[i]
    # Normalize and convert to tensor
    img_tensor = normalize_image(img)
    batch_tensors.append(img_tensor)
    # Concatenate batch and process
if batch_tensors:
    batch = torch.cat(batch_tensors, dim=0).to(device)
    # Forward pass through the encoder
    embeddings = image_encoder(batch)
    # Process and convert to numpy to free memory
    batch_features = embeddings.cpu().numpy()
    batch_features = batch_features.reshape(batch_features.shape[0], -1)
    features.append(batch_features)
    # Clear CUDA cache if using GPU
    if device.type == 'cuda':
        torch.cuda.empty_cache()
# Concatenate all batches
all_features = np.concatenate(features, axis=0)
print(f'Extracted SAM features shape: {all_features.shape}')
return all_features

def extract_inception_features(X):
    """Extract multi-scale features inspired by Inception architecture"""
    print("Extracting Inception-style features...")

    all_features = []

```

```

# Process in batches to save memory
batch_size = 100
n_batches = (len(X) + batch_size - 1) // batch_size

for batch_idx in tqdm(range(n_batches), desc="Extracting Inception features"):
    start_idx = batch_idx * batch_size
    end_idx = min((batch_idx + 1) * batch_size, len(X))
    batch_features = []

    for i in range(start_idx, end_idx):
        img = X[i]
        features = []

        # Multi-scale processing (key Inception concept)
        scales = [1.0, 0.75, 0.5, 0.25]
        for scale in scales:
            if scale == 1.0:
                scaled_img = img
            else:
                h = max(8, int(img.shape[0] * scale))
                w = max(8, int(img.shape[1] * scale))
                scaled_img = resize(img, (h, w, img.shape[2]), preserve_range=True)

        # 1. Global image statistics per channel
        for c in range(scaled_img.shape[2]):
            channel = scaled_img[:, :, c]
            features.extend([
                np.mean(channel),
                np.std(channel),
                np.median(channel),

```



```

        np.max(channel),
        np.min(channel),
        np.percentile(channel, 25),
        np.percentile(channel, 75)
    ])

```

# 2. Spatial pooling (divide image into regions and extract stats)

```
if min(scaled_img.shape[0], scaled_img.shape[1]) >= 8:
```

```
    regions_h = min(4, scaled_img.shape[0] // 4)
```

```
    regions_w = min(4, scaled_img.shape[1] // 4)
```

```
    h_step = scaled_img.shape[0] // regions_h
```

```
    w_step = scaled_img.shape[1] // regions_w
```

```
    for rh in range(regions_h):
```

```
        for rw in range(regions_w):
```

```
            h_start = rh * h_step
```

```
            h_end = (rh + 1) * h_step
```

```
            w_start = rw * w_step
```

```
            w_end = (rw + 1) * w_step
```

```
            region = scaled_img[h_start:h_end, w_start:w_end]
```

```
            # Add region mean and std for each channel
```

```
            for c in range(region.shape[2]):
```

```
                features.extend([
```

```
                    np.mean(region[:, :, c]),
```

```
                    np.std(region[:, :, c])
```

```
                ])

```

# 3. Edge features (simple gradients)

```
if min(scaled_img.shape[0], scaled_img.shape[1]) > 2:
```

```

# Convert to grayscale for edge detection
gray = np.mean(scaled_img, axis=2) if scaled_img.shape[2] == 3 else
scaled_img[:, :, 0]

# Compute horizontal and vertical gradients
if gray.shape[0] > 1 and gray.shape[1] > 1:
    grad_h = np.abs(gray[1:, :] - gray[:-1, :])
    grad_v = np.abs(gray[:, 1:] - gray[:, :-1])
    # Calculate gradient statistics for each separately
    features.extend([
        np.mean(grad_h),
        np.std(grad_h),
        np.mean(grad_v),
        np.std(grad_v),
        # Instead of adding matrices with different shapes, use mean of both
        (np.mean(grad_h) + np.mean(grad_v)) / 2
    ])
    # Add more gradient-based features for better edge representation
    features.append(np.percentile(grad_h, 90)) # Strong horizontal edges
    features.append(np.percentile(grad_v, 90)) # Strong vertical edges
    # Ensure consistent feature size by padding if necessary
min_feature_size = 500 # Target minimum feature size
if len(features) < min_feature_size:
    features.extend([0] * (min_feature_size - len(features)))
    batch_features.append(features[:min_feature_size]) # Truncate if too long

all_features.extend(batch_features)
# Convert to numpy array
inception_features = np.array(all_features)
print(f'Extracted Inception features shape: {inception_features.shape}')
return inception_features

```

```

def extract_histogram_features(X):
    """Extract color and texture histogram features"""
    print("Extracting histogram features...")
    all_features = []
    # Process in batches
    batch_size = 100
    n_batches = (len(X) + batch_size - 1) // batch_size
    for batch_idx in tqdm(range(n_batches), desc="Extracting histogram features"):
        start_idx = batch_idx * batch_size
        end_idx = min((batch_idx + 1) * batch_size, len(X))
        batch_features = []
        for i in range(start_idx, end_idx):
            img = X[i]
            features = []
            # Color histograms (more bins for better discrimination)
            for c in range(img.shape[2]):
                hist, _ = np.histogram(img[:, :, c], bins=24, range=(0, 1)) # Increased bins for
                better detail
                features.extend(hist / (np.sum(hist) + 1e-10)) # Normalize
            # Texture features using local binary patterns approximation
            if img.shape[0] >= 3 and img.shape[1] >= 3:
                # Convert to grayscale
                gray = np.mean(img, axis=2) if img.shape[2] == 3 else img[:, :, 0]
                # Simple texture patterns
                patterns = np.zeros((gray.shape[0]-2, gray.shape[1]-2, 8))
                # Define 8 neighboring pixels
                for y in range(1, gray.shape[0]-1):
                    for x in range(1, gray.shape[1]-1):
                        center = gray[y, x]
                        # Compare center with 8 neighbors

```

```

neighbors = [
    gray[y-1, x-1], gray[y-1, x], gray[y-1, x+1],
    gray[y, x-1], gray[y, x+1],
    gray[y+1, x-1], gray[y+1, x], gray[y+1, x+1]
]
# Create binary pattern
for n in range(8):
    patterns[y-1, x-1, n] = 1 if neighbors[n] >= center else 0
# Calculate histogram of patterns
for p in range(8):
    hist, _ = np.histogram(patterns[:, :, p], bins=2, range=(0, 1))
    features.extend(hist / (np.sum(hist) + 1e-10))

# Add additional texture descriptors: entropy and homogeneity approximations
block_size = 8
rows, cols = gray.shape
for r in range(0, rows - block_size + 1, block_size):
    for c in range(0, cols - block_size + 1, block_size):
        block = gray[r:r+block_size, c:c+block_size]
        # Approximate entropy (measure of randomness)
        hist, _ = np.histogram(block, bins=8, range=(0, 1))
        hist = hist / (np.sum(hist) + 1e-10)
        entropy = -np.sum(hist * np.log2(hist + 1e-10))
        features.append(entropy)
    batch_features.append(features)
all_features.extend(batch_features)
# Convert to numpy array
histogram_features = np.array(all_features)
print(f'Extracted histogram features shape: {histogram_features.shape}')
return histogram_features

```

```

def combine_features(X_sam, X_inception, X_histogram):
    """Combine different feature types and apply dimensionality reduction"""
    print("Combining features...")
    # Feature weighting to emphasize more discriminative features
    # Increase weight for SAM features which often carry more semantic information
    X_sam_weighted = X_sam * 1.2

    # Combine all features
    X_combined = np.hstack((X_sam_weighted, X_inception, X_histogram))
    print(f'Combined features shape: {X_combined.shape}')
    return X_combined

def optimize_features(X_train, X_val=None, n_components=350): # Increased
components for better information retention
    """Apply dimensionality reduction and feature selection"""
    print(f'Optimizing features from {X_train.shape[1]} to {n_components} dimensions')
    # Apply TruncatedSVD for faster dimensionality reduction
    svd = TruncatedSVD(n_components=n_components, random_state=42)
    X_train_reduced = svd.fit_transform(X_train)
    # Print explained variance
    explained_variance = np.sum(svd.explained_variance_ratio_) * 100
    print(f'Explained variance after dimensionality reduction:
{explained_variance:.2f}%')
    if X_val is not None:
        X_val_reduced = svd.transform(X_val)
        return X_train_reduced, X_val_reduced, svd
    return X_train_reduced, svd

def create_advanced_model(max_iter=300): # Increased max_iter for better convergence
    """Create an advanced stacking ensemble model"""

    # First-level models

```

```

models = [
    ('mlp', MLPClassifier(
        hidden_layer_sizes=(512, 256, 128), # Deeper network
        activation='relu',
        solver='adam',
        alpha=0.0001,
        batch_size=32,
        learning_rate='adaptive',
        learning_rate_init=0.001,
        max_iter=max_iter,
        early_stopping=True,
        validation_fraction=0.15, # Added validation split
        random_state=42
    )),
    ('rf', RandomForestClassifier(
        n_estimators=150, # More trees
        max_depth=None,
        min_samples_split=4, # Slightly reduced to allow more detailed tree splits
        min_samples_leaf=2,
        bootstrap=True,
        class_weight='balanced',
        random_state=42,
        n_jobs=-1
    )),
    ('gbm', GradientBoostingClassifier(
        n_estimators=150, # More estimators
        learning_rate=0.08, # Slightly reduced for better generalization
        max_depth=6, # Deeper trees
        min_samples_split=4,
        subsample=0.85, # Added subsampling for better generalization
    ))
]

```

```

        random_state=42
    )),
    ('svm', SVC(
        probability=True,
        kernel='rbf',
        C=2.0, # Increased regularization parameter
        gamma='scale',
        class_weight='balanced',
        random_state=42
    ))
]
# Create stacking ensemble with cross-validation
stacking = StackingClassifier(
    estimators=models,
    final_estimator=MLPClassifier(
        hidden_layer_sizes=(128, 64), # Increased complexity
        activation='relu',
        solver='adam',
        alpha=0.0001,
        batch_size=32,
        early_stopping=True,
        learning_rate_init=0.001,
        random_state=42
    ),
    cv=5,
    n_jobs=-1,
    passthrough=True # Include original features in final classifier
)

return stacking

```

```

def find_optimal_threshold(y_true, y_prob):
    """Find optimal classification threshold for imbalanced data"""
    from sklearn.metrics import precision_recall_curve, f1_score,
    balanced_accuracy_score

    # Get precision-recall curve
    precision, recall, thresholds = precision_recall_curve(y_true, y_prob)
    # Calculate multiple metrics for each threshold
    metrics = []
    for i in range(len(precision)):
        if i < len(thresholds):
            # Apply threshold
            y_pred = (y_prob >= thresholds[i]).astype(int)
            # Calculate metrics
            f1 = f1_score(y_true, y_pred)
            bal_acc = balanced_accuracy_score(y_true, y_pred)
            # Combined score with more weight on balanced accuracy for better
            generalization
            combined_score = 0.7 * f1 + 0.3 * bal_acc
            metrics.append((combined_score, f1, bal_acc, thresholds[i]))
    # Find threshold with highest combined score
    metrics.sort(reverse=True)
    if metrics:
        best_combined, best_f1, best_bal_acc, best_threshold = metrics[0]
        print(f"Optimal threshold: {best_threshold:.4f} (F1: {best_f1:.4f}, Balanced
        Accuracy: {best_bal_acc:.4f})")
        return best_threshold
    return 0.5 # Default

def perform_advanced_cv(X, y, k_folds_list=[5, 10, 15, 20], max_iter=300):
    """Perform cross-validation with multiple k values"""
    # Preprocess data

```



```

X_processed = preprocess_data(X)
    # Extract different types of features
X_sam = extract_sam_features(X_processed)
X_inception = extract_inception_features(X_processed)
X_histogram = extract_histogram_features(X_processed)
# Combine all features
X_combined = combine_features(X_sam, X_inception, X_histogram)
    # Setup results storage
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
results_file = f'outputresults/advanced_results_multiple_cv_{timestamp}.csv'
os.makedirs('outputresults', exist_ok=True)
all_cv_results = []
# Create a summary dataframe to track performance across different k values
summary_results = []
# Loop through each k_fold value
for k_folds in k_folds_list:
    print(f'\n\n{'='*50}')
    print(f'RUNNING {k_folds}-FOLD CROSS-VALIDATION')
    print(f'{'='*50}\n")
    # Use stratified k-fold for balanced class representation
    skf = StratifiedKFold(n_splits=k_folds, shuffle=True, random_state=42)
    fold accuracies = []
    fold_f1_scores = []
    fold_test_accuracies = [] # Track test accuracies specifically
    fold_results = []
    for fold, (train_idx, val_idx) in enumerate(skf.split(X_combined, y)):
        print(f'\n===== Fold {fold + 1}/{k_folds} =====')
        # Get data for this fold
        X_train_fold = X_combined[train_idx]
        X_val_fold = X_combined[val_idx]

```

```

y_train_fold = y[train_idx]
y_val_fold = y[val_idx]
print(f"Training data shape: {X_train_fold.shape}")
print(f"Validation data shape: {X_val_fold.shape}")
# Apply feature optimization
X_train_reduced, X_val_reduced, svd = optimize_features(
    X_train_fold, X_val_fold, n_components=350 # Increased components
)
# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_reduced)
X_val_scaled = scaler.transform(X_val_reduced)
# Create advanced model
model = create_advanced_model(max_iter=max_iter)
# Train model with early stopping
print(f"Training advanced ensemble model...")
start_time = time.time()
model.fit(X_train_scaled, y_train_fold)
training_time = time.time() - start_time
print(f"Training completed in {training_time:.2f} seconds")
# Get predictions
train_pred = model.predict(X_train_scaled)
val_pred_proba = model.predict_proba(X_val_scaled)[:, 1]
# Find optimal threshold for classification
optimal_threshold = find_optimal_threshold(y_val_fold, val_pred_proba)
val_pred_binary = (val_pred_proba >= optimal_threshold).astype(int)
# Calculate metrics
tn, fp, fn, tp = confusion_matrix(y_val_fold, val_pred_binary).ravel()
# Basic metrics
accuracy = (tp + tn) / (tp + tn + fp + fn)

```

```

precision = tp / (tp + fp) if (tp + fp) > 0 else 0
recall = tp / (tp + fn) if (tp + fn) > 0 else 0
f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
# Additional metrics
specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
mcc = matthews_corrcoef(y_val_fold, val_pred_binary)
balanced_acc = balanced_accuracy_score(y_val_fold, val_pred_binary)
# Calculate AUC
auc_roc = roc_auc_score(y_val_fold, val_pred_proba)
# Calculate log loss
y_pred_clipped = np.clip(val_pred_proba, 1e-15, 1 - 1e-15)
log_loss_val = -np.mean(y_val_fold * np.log(y_pred_clipped) +
                        (1 - y_val_fold) * np.log(1 - y_pred_clipped))
# Calculate G-mean
g_mean = np.sqrt(recall * specificity)
# Calculate Youden's J statistic (sensitivity + specificity - 1)
youdens_j = recall + specificity - 1
# Calculate Cohen's Kappa
kappa = cohen_kappa_score(y_val_fold, val_pred_binary)
# Store metrics
fold_metrics = {
    'K_Folds': k_folds,
    'Fold': fold + 1,
    'Feature_Type': 'MobileSAM+Inception',
    'Test_Accuracy': accuracy,
    'Train_Accuracy': (train_pred == y_train_fold).mean(),
    'Precision': precision,
    'Recall': recall,
    'AUC_ROC': auc_roc,
    'F1_Score': f1,

```

```

'Training_Loss': log_loss_val, # Using log loss as a proxy for training loss
'Testing_Loss': log_loss_val,
'Time_Taken': training_time,
'Specificity': specificity,
'MCC': mcc,
'Log_Loss': log_loss_val,
'G_Mean': g_mean,
'Youdens_J': youdens_j,
'Balanced_Accuracy': balanced_acc,
'Cohens_Kappa': kappa,
'Threshold': optimal_threshold,
'TP': tp,
'TN': tn,
'FP': fp,
'FN': fn
}
fold_results.append(fold_metrics)
fold_accuracies.append(accuracy)
fold_f1_scores.append(f1)
fold_test_accuracies.append(accuracy)
# Print current fold results
print(f"\nFold {fold + 1} Results:")
for metric, value in fold_metrics.items():
    if isinstance(value, (int, float)):
        print(f'{metric}: {value:.4f}')

# Save model with k-fold value in the filename
try:
    os.makedirs('models', exist_ok=True)
    from joblib import dump

```

```

model_prefix = f'models/advanced_ensemble_k{k_folds}fold{fold+1}'
dump(model, f'{model_prefix}.joblib')
dump(svd, f'{model_prefix}_svd.joblib')
dump(scaler, f'{model_prefix}_scaler.joblib')
# Also save threshold
with open(f'{model_prefix}_threshold.txt', 'w') as f:
    f.write(str(optimal_threshold))
print(f'Model saved to {model_prefix}.joblib")
except Exception as e:
    print(f'Could not save model: {e}")

# Calculate overall performance for this k_folds value
mean_accuracy = np.mean(fold_accuracies)
std_accuracy = np.std(fold_accuracies)
mean_f1 = np.mean(fold_f1_scores)
std_f1 = np.std(fold_f1_scores)
mean_test_acc = np.mean(fold_test_accuracies)
std_test_acc = np.std(fold_test_accuracies)
print(f'\n===== Overall Performance for {k_folds}-fold CV =====")
print(f'Average Test Accuracy: {mean_test_acc:.4f} ± {std_test_acc:.4f}")
print(f'Average F1 Score: {mean_f1:.4f} ± {std_f1:.4f}")

# Add to summary results
summary_result = {
    'K_Folds': k_folds,
    'Mean_Accuracy': mean_test_acc,
    'Std_Accuracy': std_test_acc,
    'Mean_F1': mean_f1,
    'Std_F1': std_f1,
    'Min_Accuracy': np.min(fold_test_accuracies),
    'Max_Accuracy': np.max(fold_test_accuracies),
    'Min_F1': np.min(fold_f1_scores),

```

```

        'Max_F1': np.max(fold_f1_scores),
    }
    summary_results.append(summary_result)
    # Add fold results to all results
    all_cv_results.extend(fold_results)
    # Save all detailed results
    results_df = pd.DataFrame(all_cv_results)
    results_df.to_csv(results_file, index=False)
    # Create and save summary dataframe
    summary_df = pd.DataFrame(summary_results)
    summary_file = f'outputresults/cv_summary_{timestamp}500.csv'
    summary_df.to_csv(summary_file, index=False)
    # Print final comparison of different k values
    print("\n\n==== SUMMARY OF CROSS-VALIDATION RESULTS =====")
    print(summary_df.to_string(index=False))
    return results_df, summary_df

def predict_with_model(X, k_folds=5, fold=1):
    """Make predictions using a saved model"""
    from joblib import load
    print(f'Loading saved model for {k_folds}-fold CV, fold {fold}...')
    model_prefix = f'models/advanced_ensemble_k{k_folds}fold{fold}'
    model = load(f'{model_prefix}.joblib')
    svd = load(f'{model_prefix}_svd.joblib')
    scaler = load(f'{model_prefix}_scaler.joblib')
    # Load optimal threshold
    with open(f'{model_prefix}_threshold.txt', 'r') as f:
        threshold = float(f.read().strip())
    print("Preprocessing data...")
    X_processed = preprocess_data(X)
    print("Extracting features...")
    X_sam = extract_sam_features(X_processed)

```

```

X_inception = extract_inception_features(X_processed)
X_histogram = extract_histogram_features(X_processed)
X_combined = combine_features(X_sam, X_inception, X_histogram)
print("Applying dimensionality reduction and scaling...")
X_reduced = svd.transform(X_combined)
X_scaled = scaler.transform(X_reduced)
print("Making predictions...")
probabilities = model.predict_proba(X_scaled)[: , 1]
predictions = (probabilities >= threshold).astype(int)

return predictions, probabilities

# Example usage
if __name__ == "__main__":
    # Import time for benchmarking
    import time
    start_time = time.time()
    # Run advanced cross-validation with multiple k values: 5, 10, 15, 20
    results_df, summary_df = perform_advanced_cv(
        X=X_train_full,
        y=y_train_full,
        k_folds_list=[5,10,15,20],
        max_iter=300 # Increased iterations for better convergence
    )
    # Visualize the summary results
    print("\nPlotting CV results comparison...")
    try:
        import matplotlib.pyplot as plt
        # Create a figure
        plt.figure(figsize=(12, 8))
        # Plot accuracy

```

```

plt.subplot(2, 1, 1)
plt.errorbar(summary_df['K_Folds'], summary_df['Mean_Accuracy'],
             yerr=summary_df['Std_Accuracy'], marker='o', linestyle='-', capsize=5)
plt.fill_between(summary_df['K_Folds'],
                 summary_df['Min_Accuracy'],
                 summary_df['Max_Accuracy'],
                 alpha=0.2)
plt.title('Accuracy vs. Number of CV Folds')
plt.xlabel('Number of Folds')
plt.ylabel('Accuracy')
plt.grid(True, linestyle='--', alpha=0.7)
# Plot F1 score
plt.subplot(2, 1, 2)
plt.errorbar(summary_df['K_Folds'], summary_df['Mean_F1'],
             yerr=summary_df['Std_F1'], marker='o', linestyle='-', capsize=5,
color='green')
plt.fill_between(summary_df['K_Folds'],
                 summary_df['Min_F1'],
                 summary_df['Max_F1'],
                 alpha=0.2, color='green')
plt.title('F1 Score vs. Number of CV Folds')
plt.xlabel('Number of Folds')
plt.ylabel('F1 Score')
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout(plt.savefig(f'outputresults/cv_comparison_{datetime.now().strftime("%Y
%m%d_%H%M%S")}.png'))
plt.close()
print("Plot saved to outputresults directory")
except Exception as e:
    print(f'Error creating visualization: {e}')

```



```

# Print recommended k value based on mean performance and standard deviation
best_k_acc = summary_df.loc[summary_df['Mean_Accuracy'].idxmax(), 'K_Folds']
best_k_f1 = summary_df.loc[summary_df['Mean_F1'].idxmax(), 'K_Folds']
most_stable_k = summary_df.loc[summary_df['Std_Accuracy'].idxmin(), 'K_Folds']
print(f"\nBest k value for accuracy: {best_k_acc}")
print(f"Best k value for F1 score: {best_k_f1}")
print(f"Most stable k value (lowest variance): {most_stable_k}")
# Define a combined metric that balances performance and stability
summary_df['Combined_Score'] = (
    summary_df['Mean_Accuracy'] + summary_df['Mean_F1'] -
    summary_df['Std_Accuracy'] - summary_df['Std_F1']
) best_overall_k = summary_df.loc[summary_df['Combined_Score'].idxmax(),
'K_Folds']
print(f"Recommended optimal k value (balancing performance and stability):
{best_overall_k}")
# Compare training time across different k values
time_by_k = results_df.groupby('K_Folds')['Time_Taken'].mean().reset_index()
print("\nAverage training time per fold for each k value:")
for _, row in time_by_k.iterrows():
    print(f"k={int(row['K_Folds'])}: {row['Time_Taken']:.2f} seconds")
# Calculate efficiency metric (performance per unit time)
efficiency_df = summary_df.copy()
efficiency_df['Avg_Time'] = time_by_k['Time_Taken'].values
efficiency_df['Performance_Efficiency'] = (efficiency_df['Mean_Accuracy'] +
efficiency_df['Mean_F1']) / efficiency_df['Avg_Time']
most_efficient_k =
efficiency_df.loc[efficiency_df['Performance_Efficiency'].idxmax(), 'K_Folds']
print(f"\nMost efficient k value (best performance per unit time): {most_efficient_k}")
# Save the detailed efficiency analysis

```

```

efficiency_file =
f'outputresults/cv_efficiency_{datetime.now().strftime("%Y%m%d_%H%M%S")}.csv'
efficiency_df.to_csv(efficiency_file, index=False)

# Evaluate bias-variance tradeoff
for k in summary_df['K_Folds']:
    k_results = results_df[results_df['K_Folds'] == k]
    train_acc = k_results['Train_Accuracy'].mean()
    test_acc = k_results['Test_Accuracy'].mean()
    train_test_gap = train_acc - test_acc
    print(f'\nBias-variance analysis for k={int(k)}:')
    print(f' Average train accuracy: {train_acc:.4f}')
    print(f' Average test accuracy: {test_acc:.4f}')
    print(f' Train-test gap (estimate of overfitting): {train_test_gap:.4f}')
    print(f'\nTotal execution time: {time.time() - start_time:.2f} seconds')

print("Using Advanced MobileSAM + Inception Hybrid for Sickle Cell Detection")

# First, make sure MobileSAM is installed
try:
    import mobile_sam
except ImportError:
    print("Installing MobileSAM...")
    os.system("pip install git+https://github.com/ChaoningZhang/MobileSAM.git")
    print("MobileSAM installed successfully!")
    import mobile_sam

# Import necessary components for feature extraction
from mobile_sam.modeling import ImageEncoderViT

```

**Fig 10.1. Screenshot-1**

```

Using Advanced MobileSAM + Inception Hybrid for Sickle Cell Detection
Starting model training and saving
Preprocessing data, original shape: (800, 224, 224)
No resizing needed, already at 224x224
Final preprocessed data shape: (800, 224, 224, 3)
Loading optimized SAM encoder...
Using device: cpu
Extracting SAM features: 100%|██████████| 100/100 [00:09<00:00, 10.83it/s]
Extracted SAM features shape: (800, 50176)
Extracting Inception-style features...
Extracting Inception features: 100%|██████████| 8/8 [00:22<00:00, 2.83s/it]
Extracted Inception features shape: (800, 500)
Extracting histogram features...
Extracting histogram features: 100%|██████████| 8/8 [01:32<00:00, 11.51s/it]
Extracted histogram features shape: (800, 872)
Combining features...
Combined features shape: (800, 51548)

```

**Fig 10.2. Screenshot-2**