A project report on

Neutrosophic Sets and Deep Learning: A Novel Approach for Leukaemia Diagnosis

ABSTRACT

Leukaemia, a life-threatening hematologic malignancy, requires precise and early diagnosis for effective treatment. Traditional image processing techniques often struggle with handling the uncertainty and noise inherent in medical images. This study presents a Neutrosophic Image Transformation and MobileNetV2-based deep learning framework for leukaemia classification. Neutrosophic logic, which models uncertainty using Truth (T), Falsehood (F), and Indeterminacy (I) components, is employed to enhance image clarity and feature representation.

The proposed approach involves converting medical images into the Neutrosophic domain, where each pixel is decomposed into T, F, and I components. These components undergo enhancement and reconstruction to improve feature separability, reducing noise and ambiguity in the dataset. The preprocessed images are then fed into MobileNetV2, a lightweight yet efficient deep learning model, for classification. The model is trained using a publicly available Kaggle leukaemia dataset, employing data augmentation techniques such as rotation, flipping, and intensity adjustments to improve robustness.

Experimental results demonstrate that integrating Neutrosophic preprocessing enhances classification accuracy by effectively preserving edge details and reducing image distortion. The proposed framework achieves high precision, recall, and F1-score, outperforming conventional image processing-based approaches. The combination of Neutrosophic logic and deep learning not only improves feature extraction but also enhances interpretability in medical diagnosis.

This research highlights the potential of Neutrosophic-enhanced deep learning models in medical image classification, paving the way for more reliable and accurate leukaemia diagnosis. Future work may involve extending this methodology to other hematologic and histopathological datasets, further optimizing the framework for real-world clinical applications.

CONTENTS PAGE NO

Chapter 1

Introduction

1.1 BACKGROUND

Leukaemia is a critical and often deadly form of blood cancer that targets the bone marrow and the generation of white blood cells. Accurate classification and early diagnosis of leukaemia are both vital for increasing survival rates and determining effective treatment protocols. Older methods of diagnosis, such as microscopic examination of blood smears and flow cytometry, typically involve specialist interpretation and can take considerable time. Recent developments in artificial intelligence (AI) and machine learning have introduced automated leukaemia detection systems that can improve accuracy and efficiency.

However, medical images commonly contain uncertainty and noise, which makes feature extraction and classification challenging. Neutrosophic logic, a mathematical model developed specifically to address indeterminacy, has proven to be an effective tool for medical image analysis. By presenting an image in terms of Truth (T), Falsehood (F), and Indeterminacy (I) components, Neutrosophic logic facilitates improved feature enhancement and noise reduction. In this research, Neutrosophic Image Processing is combined with MobileNetV2, which is a light-weight convolutional neural network (CNN), to create a stable leukaemia classification model.

1.2 PROBLEM STATEMENT

Leukaemia diagnosis based on microscopic blood smear images is a very challenging task in the presence of inherent noise, variability, and uncertainty of medical images. Conventional deep neural network models tend to fail against artifacts, non-uniform illumination, and variations in staining, which veil the key morphological information required for precise diagnosis. Additionally, conventional image processing methods are incapable of strengthening significant structural information, resulting in misclassification. Deep learning models mainly utilize raw pixel values and hence are prone to low-contrast areas, overlapping cell structures, and artefacts in the background, which negatively affect their performance. Another problem of dataset imbalance, where

some leukaemia subtypes are underrepresented in the datasets, would cause biased predictions, further lowering classification reliability.

Another significant shortcoming in deep learning-based leukaemia classification is the non-interpretability of AI models, which are used as black-box systems. Though deep neural networks provide high accuracy, the workings of their decision-making are obscure, creating concerns for clinical uses. In addition, most state-of-the-art architectures, including ResNet and EfficientNet, consume large computational resources and hence cannot be practically used in real-time deployment in resource-constrained environments. With a view to mitigating these issues, this paper combines Neutrosophic Image Processing with MobileNetV2, using Neutrosophic logic to clarify the image and minimize ambiguity prior to classification. This method is attempted to enhance classification accuracy, make the model easier to interpret, and give more efficient and resilient leukaemia detection to real-world healthcare applications.

1.3 OBJECTIVES OF THE STUDY

The main goal of this research is to propose a Neutrosophic Image Processing-based deep learning architecture for leukaemia classification. By incorporating Neutrosophic logic into MobileNetV2, the research seeks to improve feature extraction, minimize uncertainty, and enhance classification accuracy. The particular objectives are:

1. Creating a Neutrosophic Image Transformation method that separates medical images into Truth (T), Falsehood (F), and Indeterminacy (I) parts in order to enhance feature clarity and minimize noise.

2. Incorporating Neutrosophic preprocessed images into MobileNetV2 for leukaemia classification, capitalizing on its computational power and high feature extraction capacities.

3. Assessing the effect of Neutrosophic preprocessing on the performance of deep learning by utilizing critical metrics including accuracy, precision, recall, and F1-score.

4. Examining the influence of dataset augmentation methods on model generalizability and resilience, guaranteeing flexibility to varying medical image variations.

5. Improving model interpretability by evaluating feature contributions, so that AI-based decisions in leukaemia classification become more explainable and clinically relevant.

This work seeks to make a contribution towards the area of AI-based medical diagnostics by offering an uncertainty-aware, efficient, and interpretable leukaemia detection system for real-world healthcare applications.

1.4 SCOPE AND LIMITATIONS

The Scope of the study is as follows:

1. The study is specifically on leukaemia classification using a publicly shared Kaggle dataset of microscopic blood smear images.

2. It applies Neutrosophic logic-based preprocessing to augment image features prior to classification by a MobileNetV2-based CNN model.

3. The study uses conventional machine learning metrics such as accuracy, precision, recall, and F1-score to evaluate model performance.

There are nonetheless some Limitations in this study:

1. Kaggle-originating images, instead of covering wide real-life clinical variations.

2. The investigation tends to major on Neutrosophic enlargement for the task of preprocessing using images, failing to encompass alternative state-of-the-art techniques handling uncertainty.

3. Efficiency in MobileNetV2 aside, other architectures like ResNet or EfficientNet aren't being looked into this study.

1.5 PROPOSED APPROACH

The suggested leukaemia detection system combines Neutrosophic Transformation with MobileNetV2 for improved classification accuracy. The algorithm starts with image preprocessing, in which blood smear images are cleaned of noise and enhanced with contrast to enhance visibility. The image is then decomposed into three components: Truth (T), Falsehood (F), and Indeterminacy (I) by the Neutrosophic Transformation, assisting

in the handling of uncertainty in low-quality or uncertain areas. To further process these components, Neutrosophic Enhancement is utilized. The T (Truth) component receives CLAHE-based contrast enhancement and Unsharp Masking to emphasize important details. The F (Falsehood) component is treated with Total Variation Minimization (TVM) denoising, the result being effective noise reduction while maintaining structural integrity. The I (Indeterminacy) component is treated with Wavelet Shrinkage Denoising (BayesShrink) to mask uncertainty and then Multi-Scale Wavelet Sharpening for the improvement of fine details. These improvements are separately implemented on the Red, Green, and Blue (RGB) channels to ensure a noise-free, clearer, and higher-contrast leukaemia cell representation, which ultimately leads to more accurate classification.

Second, feature extraction is done via MobileNetV2, a lightweight yet efficient deep learning architecture. It uses depth wise separable convolutions and inverted residual blocks to efficiently extract leukaemia-related patterns without incurring high computational expenses. The features are then fed into a Softmax classifier to determine if the image is that of a leukemic or non-leukemic sample.

In comparison with other conventional CNNs such as ResNet, VGG, and DenseNet, MobileNetV2 achieves the best accuracy-efficiency trade-off and can be applied in real-time medical diagnostics. Accuracy, precision, recall, and F1-score are used to assess the performance of the system, providing sound and interpretable results for the diagnosis of leukaemia. The proposed method improves diagnostic accuracy and tackles uncertainty in medical imaging by incorporating Neutrosophic processing and deep learning.

This method tries to provide a strong, uncertainty-conscious, and computationally minimal leukaemia classification system and hence is applicable for real-world clinical usage

Chapter 2

Literature Review

2.1 INTRODUCTION

Leukaemia detection has been a research hotspot in medical imaging and artificial intelligence, with a number of methods proposed over the years. Classical methods, deep learning-based methods, and hybrid methods with uncertainty-handling mechanisms have all been tried. This section summarizes the current literature on leukaemia detection, identifies their shortcomings, and presents the Neutrosophic transformation, which improves feature extraction through uncertainty handling in medical images.

2.2 TRADITIONAL APPROACHES FOR LEUKAEMIA DETECTION

Traditional leukaemia detection techniques are heavily dependent on microscopic blood smear examinations, which are done manually by hematologists or traditional image processing methods. Some of these techniques are:

1. **Morphological Feature Analysi:**: Cell shape, size, and texture are analyzed to distinguish between normal and abnormal cells.

2. **Thresholding and Edge Detection**: Otsu's thresholding and Canny edge detection are applied to segment WBCs from the background.

3. **Color and Texture-Based Segmentation:** Techniques like K-means clustering and watershed segmentation segment cells according to color and texture patterns.

Although these techniques give preliminary results, they are subject to subjectivity, time, and noise and artifact susceptibility, resulting in diagnostic inconsistencies. In addition, conventional feature engineering is domain-specific and tends not to generalize across heterogeneous datasets.

2.3 DEEP LEARNING-BASED TECHNIQUES

With developments in AI, Convolutional Neural Networks (CNNs) have emerged as thego-to method for leukaemia detection and classification automatically. Some popular deep learning architectures employed in the diagnosis of leukaemia are:

1.CNN-based Models: Basic architectures such as AlexNet, VGG16, and ResNet automatically extract high-level features with less reliance on manual feature engineering.

2.Transfer Learning Approaches: Pre-trained models such as InceptionV3 and EfficientNet utilize big image datasets to enhance accuracy using minimal medical data.

3. Hybrid Models: A few studies blend CNNs with conventional feature extraction techniques to enhance interpretability and efficiency.

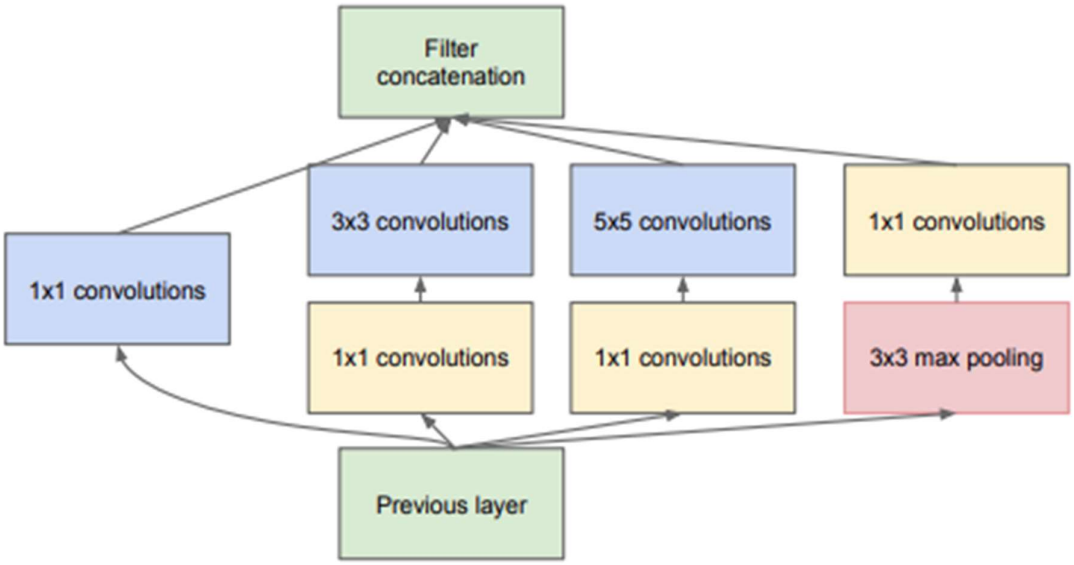


Fig1: Filter Concatenation

Despite their effectiveness, deep learning models suffer from a number of challenges, namely poor interpretability, high computational expense, and sensitivity to noise and dataset imbalance. Traditional CNNs deal with raw pixel values, hence making them sensitive to low-contrast and noisy areas, which cause misclassification.

2.4 NEUTROSOPHIC TRANSFORMATION & ENHANCEMENT

Neutrosophic Logic, brought forward by Florentin Smarandache, is a mathematical structure created to deal with uncertainty, indeterminacy, and contradiction in information. In medical imaging, the Neutrosophic transformation separates an image into three parts:

- **Truth (T):** Represents clear and certain features (e.g., well-defined cell boundaries).
- **Falsehood (F):** Represents irrelevant or noisy features (e.g., background noise, artifacts).
- **Indeterminacy (I):** Represents ambiguous or uncertain regions in the image (e.g., low-contrast areas like edges).

Then, Enhancement is done to refine and optimize the transformed components (**T - Truth, F - Falsehood, and I - Indeterminacy**) for improved feature clarity and classification accuracy.

2.5 WORKING OF NEUTROSOPHIC TRANSFORMATION

1. **Image Preprocessing:** Input blood smear image is converted from RGB to RGB, enhanced contrast, and noise removal
2. **Neutrosophic Decomposition:** Each pixel is transformed into the Neutrosophic domain (T, F, I components).

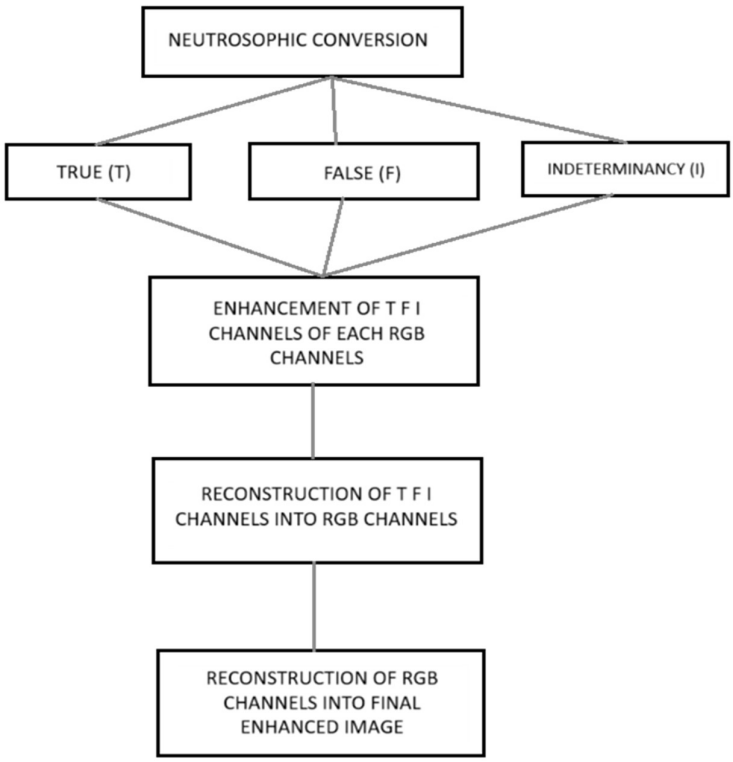


Fig2: Illustration of Neutrosophic Image Transformation Process:

3. **Feature Enhancement:** The Indeterminacy (I) component is refined using morphological filtering and adaptive thresholding.
4. **Reconstruction:** The transformed image is reconstructed with enhanced clarity and noise reduction, making it more suitable for deep learning models.

Advantages of Neutrosophic Transformation:

- Decreases uncertainty and noise, enhancing accuracy in classification.
- It increases contrast and sharpens relevant features for deep learning models.
- Offers improved interpretability than conventional image processing methods.

2.6 TRANSFER LEARNING MODELS

Transfer learning has become a robust method in medical image analysis in recent years, enabling pre-trained deep models to be used to adapt for domain-specific tasks using limited data. Rather than training models from the beginning, transfer learning exploits feature representations that are learned on large-scale data like ImageNet and fine-tunes them to adapt to particular applications, for instance, detecting leukaemia. This method greatly lowers computational expenses, accelerates training, and improves model performance, and it is very effective for medical diagnosis.

A number of pre-trained deep learning models have been investigated for leukaemia classification. ResNet is very common for its power to efficiently train deep networks by resolving the vanishing gradient issue using skip connections. VGG, with its sequential deep layers, excels at hierarchical feature learning at the cost of high computational requirements. DenseNet improves the reuse of features and enhances the flow of gradients, so it's very efficient for medical imaging applications.

MobileNetV2, a light CNN optimized for performance, is utilized here in this work as the backbone for feature extraction. MobileNetV2 is pre-trained on ImageNet and extracts basic features such as edges, textures, and shapes, which are essential in detecting leukaemia-related abnormalities. Through fine-tuning the model with leukaemia data, it acquires domain-specific patterns along with maintaining universal image feature representations. Adding the Neutrosophic transformation further increases the clarity of the features, increasing the model to better deal with uncertainty and noise in blood smear images.

2.7 EVALUATION METRICS

The performance of a model can be measured in terms of multiple metrics like accuracy, precision, recall, F1-score, and confusion matrix. These all give a clearer picture of the performance of the model on multiple classes. Accuracy gives the correctness of the model as a whole, but sometimes it might not give the overall picture when the class distribution is not balanced. Conversely, precision and recall can be applied to measure how well a model performs in terms of detecting positive instances, while F1-score gives a balanced estimate of precision and recall.

Classification report from scikit-learn is utilized within this project in order to obtain a detailed report consisting of precision, recall, and F1-score per class. This can be especially handy for model comparison on imbalanced datasets or on multiclass tasks.

2.8 SUMMARY AND RESEARCH GAP

Leukaemia diagnosis has long been dependent on manual microscopic inspection, which is time-consuming, subjective, and susceptible to human error. To address these limitations, machine learning and deep learning methods have been investigated for automatic classification. Traditional CNN-based architectures like ResNet, VGG, and DenseNet have shown encouraging results in leukaemia detection. Nevertheless, they are limited in dealing with uncertain, noisy, and low-contrast blood smear images, resulting in misclassifications.

To solve these challenges, Neutrosophic transformation has been proposed as a mathematical theory to handle uncertainty in medical images. It breaks down images into Truth (T), Indeterminacy (I), and Falsehood (F) components, improving feature discriminability and stability in classification. Furthermore, transfer learning using pre-trained deep learning models has dramatically enhanced performance for medical image analysis. MobileNetV2, pre-trained on ImageNet, is especially effective because it is lightweight and has high feature extraction capability.

Through the combination of Neutrosophic transformation and MobileNetV2, the suggested methodology increases classification performance, stability, and computational effectiveness, making it a good candidate for real-time clinical use. This integration

directly responds to fundamental research loopholes in uncertainty management, model explainability, and computational viability, thus making it a sound candidate for automated leukaemia detection.

2.9 CHALLENGES AND FUTURE DIRECTIONS

While deep learning has greatly enhanced leukaemia detection, current models are still challenged to deal with uncertain, noisy, and low-contrast blood smear images. Manual microscopic inspection is used in traditional methods, which is time-consuming and susceptible to human error. Deep learning algorithms like ResNet, VGG, and DenseNet have enhanced classification accuracy but perform poorly on unclear image areas, causing misclassifications.

The fundamental gap in current work is an ineffective handling of uncertainty in medical images. Classical CNNs model all pixels indiscriminately and hence are prone to noise changes and staining patterns. Deep models are also lack interpretable meaning, and physicians are unable to verify predictions from them. Its high computational expenses limit real-time clinical implementation in another limitation. The suggested fusion of Neutrosophic transformation with MobileNetV2 seeks to close these gaps by enhancing robustness, interpretability, and efficiency in the detection of leukaemia.

Hybrid deep learning models, e.g., CNN-Transformer architectures, should be the focus of future studies in leukaemia detection to improve feature extraction. Explainable AI (XAI) methods such as Grad-CAM and SHAP can enhance model interpretability for clinical applications. Federated learning will make privacy-preserving AI possible, while Edge AI will make real-time leukaemia detection on mobile and embedded systems possible. Increased development in Neutrosophic logic, fuzzy sets, and rough set theory will enhance uncertainty management in medical imaging. These technologies will increase accuracy, efficiency, and real-world applicability, making AI-aided leukaemia diagnosis more reliable and accessible.

2.10 HYBRID APPROACHES

Along with the use of feature extraction based on deep learning, conventional methods may also be integrated to improve leukaemia classification. Under our method, once the Neutrosophic Transformation is applied, decomposed parts (T - Truth, F - Falsehood, and I - Indeterminacy) may undergo texture-based processing to emphasize microscopic variation between malignancy and benign cells. Gray-Level Co-Occurrence Matrix (GLCM) is utilized for the T component to record spatial relations among pixel intensities, aiding in pattern recognition for leukaemia cells. Local Binary Patterns (LBP) are employed to examine the textural patterns in the I component, efficiently separating uncertainty variations within cell regions. Morphological characteristics like nucleus area, cell shape, and perimeter are also obtained from the F component to evaluate leukaemia-infected cell irregularities. While MobileNetV2 is the main deep learning model, these manually crafted features can be used to pass through an auxiliary classifier, such as Support Vector Machine (SVM) or a Fully Connected Neural Network (FCNN), to augment predictions. This hybrid technique combines both deep learning and domain-specific feature extraction to achieve a more complete leukaemia detection framework specific to the dataset and transformation used in this research.

Through the combination of deep learning and hand-engineered feature extraction methods, the model provides a holistic evaluation of leukaemia images, enhancing classification accuracy and reliability in actual medical diagnosis.

Chapter 3

System Architecture and Design

3.1 OVERVIEW OF THE ARCHITECTURE

The designed leukaemia detection system is meant to process blood smear images effectively and determine whether they are leukemic or non-leukemic based on a deep learning model augmented by Neutrosophic transformation. The architecture is made up of several interacting stages: image preprocessing, Neutrosophic transformation, feature extraction with MobileNetV2, classification, and performance evaluation. All of these modules play their part in enhancing the precision and dependability of the process of leukaemia diagnosis.

The system starts with image preprocessing, in which input blood smear images are first converted to grayscale, then filtered to reduce noise, and finally enhanced for contrast. These processes help retain important cellular structures while removing unnecessary noise, which may otherwise interfere with the feature extraction process. Once the images are preprocessed, they are then converted into the Neutrosophic domain, in which every pixel is defined by three independent components: Truth (T), Indeterminacy (I), and Falsehood (F). The Truth component takes well-defined structures of leukaemia cells, the Falsehood component picks out irrelevant or noisy areas, and the Indeterminacy component picks out uncertain or ambiguous areas. This conversion enhances the resistance of the classification model, particularly when handling low-contrast and highly uncertain images.

After transforming the images, feature extraction is carried out with MobileNetV2, which is a light-weight convolutional neural network (CNN) for high-speed image processing. MobileNetV2 uses depth wise separable convolutions and inverted residual blocks to derive essential leukaemia-related features without sacrificing computational efficiency. This is what makes it ideal for real-time medical diagnostics where efficiency counts the most. The feature maps that are extracted are then input to a Softmax classifier that predicts whether the input image belongs to a leukemic or a non-leukemic case.

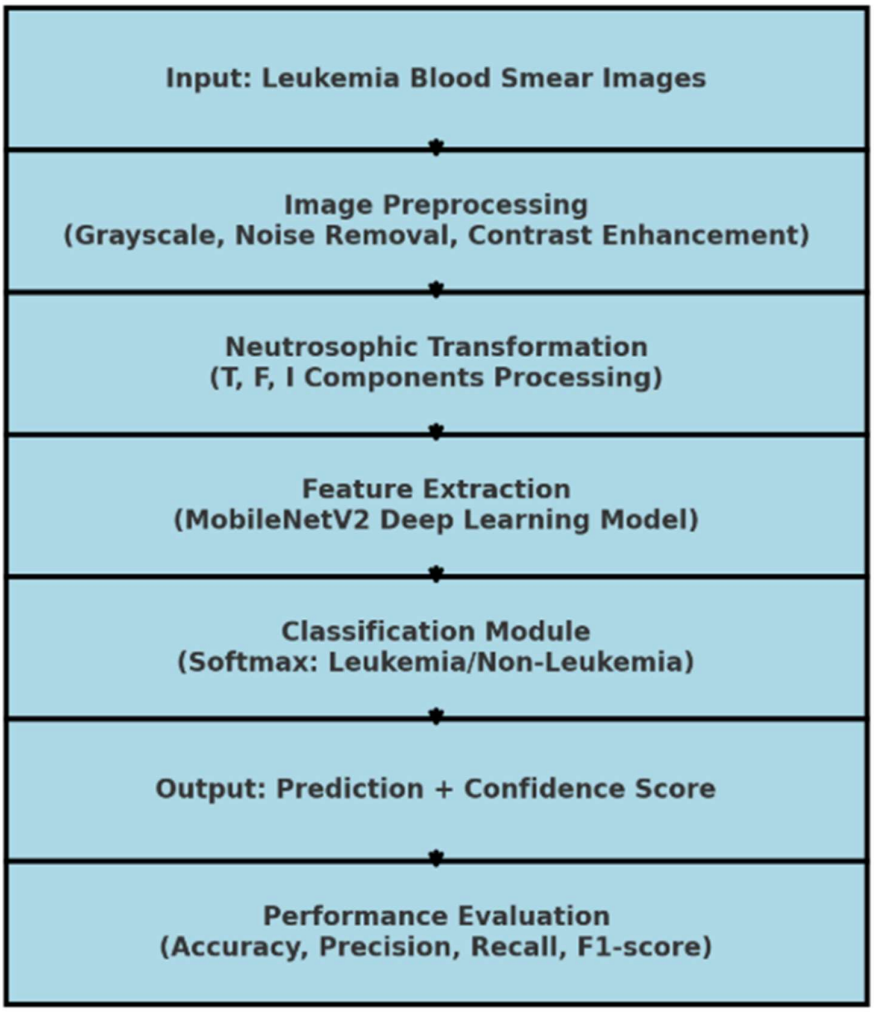


Fig 3: Overall Architecture Diagram

Aside from MobileNetV2, other deep learning models like ResNet, VGG, and DenseNet have been extensively applied in the detection of leukaemia. ResNet uses residual learning to solve the vanishing gradient problem, enabling deeper networks to learn effectively. VGG, through its deep sequential layers, is good at hierarchical feature extraction but requires more computational power. DenseNet, by dense inter-layer connectivity, enhances feature propagation and gradient flow and is thus very efficient for medical imaging applications. MobileNetV2, however, was used in this research because it has a better balance between accuracy and computational cost, which guarantees that the model can function well in real-time diagnostic environments.

Lastly, the system measures its performance based on traditional classification metrics including accuracy, precision, recall, and F1-score. The metrics guarantee that the model does not only exhibit high detection precision but also has low false positives and false negatives, which are essential in clinical diagnostics. With the combination of Neutrosophic transformation and deep learning-based feature extraction, the developed architecture efficiently maximizes the reliability and interpretability of leukaemia detection, thus representing a useful device for clinical application.

3.2 DATA FLOW AND PROCESSING

The image flow in leukaemia detection starts with microscopic blood smear image acquisition, normally taken at 224×224 pixel resolution or better to preserve cellular morphology. Such images are prone to staining irregularities, noise, and varying illumination and therefore require a robust preprocessing chain to improve feature extraction and classification accuracy.

Preprocessing commences with image resizing, whereby all blood smear images are resized to a fixed size to ensure uniformity. This is to ensure compatibility with the deep learning model. Noise reduction methods, i.e., filtering, are used to improve image quality and eliminate spurious artifacts that may contaminate feature extraction. Normalization is then performed to normalize pixel values within the range 0 to 1 or -1 to 1 to enhance model convergence and stability. Post-data cleaning, data augmentation is applied to artificially augment the dataset to enhance generalization. Rotation, flipping, adjusting brightness, and adding noise incorporate variability while retaining critical features. This enables diverse patterns to be learned by the model, making it less prone to overfitting and more accurate in classification.

To cope with uncertainty and noise in medical images, Neutrosophic transformation comes into play. This method dissects the image into Truth (T), Indeterminacy (I), and Falsity (F) parts to better highlight discriminative features for classification. Also, Neutrosophic Enhancement is performed on the Truth (T), Indeterminacy (I), and Falsity (F) parts individually. It is separately applied to the Red, Green, and Blue (RGB) channels so that each color component maintains meaningful features. The final, enhanced images present better contrast, less noise, and clearer feature detection, which helps with more accurate classification of the leukaemia cells.

The preprocessed and enhanced images are then fed into MobileNetV2, where deep feature extraction occurs. The feature maps obtained through extraction are input into a fully connected classifier, which outputs whether the sample is leukemic or not. This organized data flow guarantees an efficient, precise, and generalizable deep learning-based leukaemia detection system.

3.3 NEUTROSOPHIC TRANSFORMATION & ENHANCEMENT

As a system architecture's central component, Neutrosophic Transformation is used to manage uncertainty, noise, and indeterminate areas in blood smear images to improve feature sharpness for detecting leukaemia. Conventional image processing methods tend to fail when dealing with uncertain pixel intensities, staining variations, and overlapping cell structures, causing misclassification. Neutrosophic theory overcomes these issues by breaking down an image into three basic components:

- **Truth (T)** – Represents the degree of certainty or clarity in an image region.
- **Indeterminacy (I)** – Captures uncertain or ambiguous regions, often caused by noise or poor contrast.
- **Falsity (F)** – Identifies misleading or false information in the image, helping to remove irrelevant details.

In preprocessing, the input image is initially normalized and mapped to a Neutrosophic domain, with each pixel being allocated values in these three sets. With adaptive thresholding and filtering applied, the transformation extracts prominent leukaemia-related features while minimizing uncertainty and noise. Subsequently, Neutrosophic Enhancement is used to obtain the enhanced image with enhanced contrast, decreased noise, and clearer features. The fine-tuned image is then fed into MobileNetV2, where deep feature extraction happens. This combination enhances classification resilience in that only significant and well-crafted patterns are involved in making decisions.

With the inclusion of Neutrosophic Transformation and improvement in the system structure, the model becomes less sensitive to image quality fluctuations, resulting in more precise leukaemia classification in practical medical practice.

3.4 FEATURE EXTRACTION AND CLASSIFICATION

The feature extraction and classification process is the key part of the system design, wherein deep learning methods are applied to process preprocessed blood smear images and differentiate them as leukemic or non-leukemic. Once they pass through Neutrosophic Transformation, the processed images are passed through MobileNetV2, a light weight convolutional neural network (CNN) whose strength lies in feature extraction.
MobileNetV2 utilizes depth wise separable convolutions, with a drastic cut in computational complexity while maintaining spatial hierarchies of leukaemia cells. The network processes the input images using multiple convolutional layers, learning high-dimensional features like cell shape, nucleus texture, and cytoplasm intensity variation, which are essential for leukaemia detection. The feature maps extracted are flattened and fed into a fully connected layer for classification.

The classification task is performed by a fully connected layer with a Softmax activation function that gives probability scores to various classes (leukemic or non-leukemic). The output prediction is the maximum of the highest probability score. For further improving classification performance, methods such as batch normalization and dropout regularization are used to avoid overfitting.

By taking advantage of MobileNetV2's low-complexity feature extraction and a strong classification framework, the system proposed is of high accuracy, computational speed, and stable leukaemia detection, which is appropriate for real-time clinical use.

Chapter 4

Methodology

4.1 DATA COLLECTION

Data collection is a very important process in developing a successful machine learning model, particularly for tasks such as image classification. In this project, we concentrate on collecting a representative and varied set of images, which will be utilized in training and testing the performance of the deep learning models.

Acute Lymphoblastic Leukaemia (ALL) is a very common type of blood cancer that needs invasive and time-consuming diagnostic tests for definite diagnosis. One of the key initial screening tests for ALL is the review of peripheral blood smear (PBS) images, which serves to distinguish between cancerous and non-cancerous cases. Yet, visual inspection of these images by laboratory personnel is susceptible to diagnostic mistakes, since the non-specificity of ALL symptoms usually leads to misdiagnosis. To overcome these challenges, automated analysis of PBS images using deep learning has proved to be an effective method of early detection of leukaemia.

In this research work, the dataset was acquired from the bone marrow laboratory at Tehran's Taleqani Hospital in Iran. It comprises 3,256 images of PBS that were obtained from 89 ALL suspected patients. The blood samples were carefully prepared and stained by skilled laboratory professionals to obtain high-quality images. The dataset is divided into two broad categories: benign and malignant. The benign category includes hematogones, while the malignant category consists of ALL cases, which are further categorized into three subcategories: Early Pre-B, Pre-B, and Pro-B ALL. All images were taken on a Zeiss microscope with a high-definition camera at 100x magnification and saved in JPG format.

To be accurate in classification, the images were labelled by a specialist in haematology through flow cytometry analysis, which is accurate in differentiating ALL subtypes. This well-organized and varied dataset is a good basis for training deep learning models, which leads to automated detection of leukaemia with high accuracy.

4.2 DATA PREPROCESSING AND AUGMENTATION

Data preprocessing and data augmentation are crucial processes in building a strong leukaemia detection model. Medical images are likely to include noise, lighting variations, and staining inconsistencies, hence the preprocessing process ensures that the images are cleaned and normalized for deep learning models. Augmentation, on the other hand, improves the dataset through the inclusion of variation, enhancing the model's generalizability and avoiding overfitting.

The preprocessing process starts with resizing images to a standard size, often 224×224 pixels, for compatibility with deep learning models like MobileNetV2. To ensure consistency in pixel intensity, normalization rescales the pixel intensity between [0,1] or [-1,1], so that no specific intensity range overpowers the learning process of the model. Contrast enhancement and histogram equalization enhance the contrast of cellular structures, so that leukaemia cells are more visible.

After preprocessing, data augmentation is used to enhance the model's capability to identify leukaemia cells under different conditions. Random rotations, flipping, zooming, and cropping add diversity so that the model does not depend on a specific orientation or size when detecting cancerous cells. For simulating real-world variations in imaging, Gaussian noise is added, and brightness and contrast changes enable the model to accommodate different lighting conditions. Elastic deformations bring about minor distortions, emulating the intrinsic variation in blood smear images and increasing the model's robustness.

To avoid class imbalance, oversampling and undersampling strategies are used if required. When there are more normal cells than leukaemia-positive ones in the dataset, augmentation serves to create synthetic variations of the minority class so as not to bias. Augmentation techniques like Mixup and Cutmix that use sophisticated mixing techniques of many images enhance generalization of the model.

By using a well-designed preprocessing and augmentation pipeline, the leukaemia detection model is more capable of distinguishing between malignant and healthy cells. These methods improve classification accuracy and make the model work well in real-

world clinical environments, where image variations are prevalent due to variations in laboratory conditions and equipment.

4.3 NEUTROSOPHIC TRANSFORMATION

Neutrosophic transformation is an important process in the methodology of leukaemia detection, increasing the model's power to deal with uncertain, imprecise, or incomplete information. In medical imaging, images of blood smears usually include noise, uneven illumination, and unclear boundaries among various cell types, for which conventional image processing methods are less effective. Neutrosophic theory, an extension of classical binary and fuzzy logic by including three basic ingredients—truth (T), indeterminacy (I), and falsity (F)—provides a more flexible and resilient approach to image analysis.

In leukaemia categorization, every pixel of the input image is converted into a neutrosophic space, where it is defined in terms of three membership values: T (membership to leukemic cells), I (uncertainty level in classification), and F (non-leukemic nature). This conversion aids in retaining important information with a minimized effect of noise and artifacts, thus giving better feature extraction. The neutrosophic set is built up using intensity-based segmentation and edge-preserving filters, which ensure that the pertinent cellular structures are emphasized without the irrelevant background features being accentuated.

After transformation, entropy-based thresholding is performed to specify the best segmentation of leukaemia cells so that the model can concentrate on diagnostically relevant areas. The revised image is subsequently input into a deep learning pipeline, where classification and feature extraction are better performed because of the improved representation of uncertainty. Neutrosophic processing improves the robustness of the model for detecting leukaemia by resolving the inherent vagueness in medical images, with improved classification accuracy and generalization. By incorporating this change into the method, the system becomes more trustworthy for actual leukaemia screening and diagnosis.

4.4 NEUTROSOPHIC ENHANCEMENT

The **Neutrosophic Enhancement** process refines the transformed image components (**T, F, and I**) to improve leukaemia cell classification. Each component undergoes specialized enhancement techniques to optimize contrast, sharpness, and noise reduction.

1. **Truth (T) Component Enhancement**
    - The **Contrast Limited Adaptive Histogram Equalization (CLAHE)** enhances local contrast, making important features more distinguishable.
    - The **Unsharp Masking technique** further sharpens the details in the T component, improving edge definition.
2. **Falsehood (F) Component Enhancement**
    - The **Total Variation Minimization (TVM) denoising** method reduces image noise while preserving crucial structural details, ensuring a cleaner representation of falsehood information.
3. **Indeterminacy (I) Component Enhancement**
    - **Wavelet Shrinkage Denoising** (using BayesShrink) suppresses uncertainty by removing unwanted noise without blurring important details.
    - **Multi-Scale Wavelet Sharpening** further enhances the I component by refining fine details using Discrete Wavelet Transform (DWT) with the 'db2' wavelet.

The **enhancement process is performed after the Neutrosophic Transformation** to refine and optimize the transformed components (**T - Truth, F - Falsehood, and I - Indeterminacy**) for improved feature clarity and classification accuracy. Here's why:

1. **Preserving Neutrosophic Information**
    - The Neutrosophic transformation decomposes the image into three meaningful components that capture certainty (T), falsity (F), and uncertainty (I). Directly applying enhancement to the raw image may distort essential features, whereas enhancing after transformation ensures that each component is optimized without losing its interpretability.
2. **Targeted Enhancement**
    - Each neutrosophic component represents different aspects of the image. **T needs contrast enhancement and sharpening**, **F requires noise suppression**, and **I benefits from uncertainty reduction**. Enhancing these

components separately allows for more precise adjustments, ensuring better feature extraction for classification.

3. **Reducing the Impact of Noise and Artifacts**
    - The transformation process can introduce minor distortions or amplify noise in certain components. Enhancement techniques like **Wavelet Shrinkage Denoising, Total Variation Minimization (TVM), and Unsharp Masking** help mitigate these effects, resulting in cleaner and more informative inputs for the deep learning model.
4. **Improving Classification Accuracy**
    - A well-enhanced image improves the quality of extracted features, leading to **better separation between leukaemia and non-leukaemia cells**. By enhancing contrast, sharpening edges, and reducing uncertainty, the model learns more distinctive and relevant patterns, ultimately boosting classification performance.

Thus, performing enhancement after Neutrosophic Transformation ensures that the most critical features are preserved and refined, leading to improved model robustness and accuracy in leukaemia detection.

4.5 FEATURE EXTRACTION

Feature extraction is a critical step in the pipeline for detecting leukaemia because it takes raw image data and transforms them into useful numerical representations that the classification model can effectively process. Because leukaemia detection involves identifying between malignant and healthy blood cells using shape, texture, and intensity differences, feature extraction improves the accuracy of classification and model stability. Feature extraction is done post-neutrosophic transformation and enhancement when the images are described in terms of truth (T), indeterminacy (I), and falsity (F) values to better depict uncertainty in data.

In this approach, feature extraction is done mainly through deep learning-based techniques. Pre-trained convolutional neural network (CNN) MobileNetV2, is used for extracting high-level spatial features capturing leukaemia cell attributes. The CNN learns hierarchical features from the input images, such as edge information, texture patterns, and shape descriptors, automatically. By the utilization of transfer learning, the pre-trained model

captures meaningful representations without the need for intensive training on new sets, improving computational efficiency remarkably and avoiding overfitting risk.

4.6 CLASSIFICATION AND PREDICTION

Classification and Prediction constitute the final and most crucial phase in the leukaemia detection process, where the features extracted from preprocessed and resized images are computed to determine whether a given blood smear image belongs to a benign or malignant category. This is a critical phase in the automated diagnosis of leukaemia since it allows for early diagnosis, reduces reliance on human inspection, and enhances diagnostic accuracy. Classification is done using deep learning algorithms, namely convolutional neural networks (CNNs), to obtain accurate and effective leukaemia detection.

Once the features are extracted from a pre-trained deep learning model MobileNetV2, the features are passed through fully connected layers for classification. The final layer of the model applies a softmax activation function, which assigns probabilities to each class (benign or malignant). The most probable class is considered the predicted label for the input image. For more specific classification, the model can also predict the leukaemia subtype (Early Pre-B, Pre-B, or Pro-B ALL) through a multi-class classification. The model is trained with a cross-entropy loss function, which punishes misclassifications to provide overall accuracy.

For better classification process, batch normalization and dropout regularization are employed. Batch normalization normalizes the input to each layer to avoid the internal covariate shifts and stabilizes the learning process, and dropout avoids overfitting by randomly dropping out the neurons during training.

The prediction stage is where the model is tested on unseen test data to determine its actual performance in practice. Typical metrics like accuracy, precision, recall, F1-score, and AUC-ROC are employed to determine how accurately the model can distinguish between benign and malignant cells. Recall is especially significant in leukaemia detection because a false negative, i.e., failing to detect a malignant case, would have critical implications.

For clinical deployment, the trained model can be integrated into a diagnostic platform where new blood smear images are fed into the pipeline for computer-aided screening of

leukaemia. The model's confidence scores can also be utilized to aid medical professionals in making informed decisions, offering a robust, reliable, and efficient leukaemia detection system. With the integration of deep learning techniques with careful model optimization, classification and prediction are highly accurate, enabling faster and more accurate leukaemia diagnosis.

Chapter 5

Implementation

5.1 MODULES

5.1.1 General Libraries for Data Handling:

Numpy

Numpy is a high-level, high-performance numerical computation library in Python. Numpy supports n-dimensional arrays and matrices and a vast range of high-level mathematical functions to manipulate them. Numpy is used mainly in deep learning for the numerical computation of data, like manipulation of the array in images and matrix operations during model training. For example, numpy is utilized in the code to manipulate the prediction and to compute the mean of the output of various models in the ensemble technique..

OpenCV(cv2)

OpenCV (cv2) is used for image processing and handling leukaemia blood smear images in this project. It offers image loading and saving, which is convenient for simple writing and reading of images in various formats. It offers data conversion, which transforms images into numerical arrays that can be taken into deep learning models. OpenCV also offers batch processing, which facilitates effective processing of several images, which is crucial in large data sets. With OpenCV, the project is able to provide effective image preprocessing, which is convenient for feature extracting, classifying, and detecting leukaemia when using deep learning.

5.1.2 Libraries for Deep Learning and Model Loading:

TensorFlow/Keras

Tensorflow is an open-source machine learning library developed by Google. Keras, which was initially a separate deep learning library, is now part of TensorFlow and acts as its high-level API. Keras simplifies the process of developing and training deep learning models using easy-to-use interfaces and functions for model construction..

- **Keras.models.load_model** is used to load pre-trained models from saved .h5 files.
- **Keras.preprocessing.image** contains utilities for loading and preprocessing images, which is crucial when preparing images to be used by model predictions.

- **Keras.applications** provides pre-trained models such as ResNet50, VGG19, and MobileNetV2 that can be used in transfer learning, which is a technique where a model trained on a task is used for a different task.

Pre-trained Model (MobileNetV2)

MobileNet model, a light-weight deep learning model, is employed for feature extraction and leukaemia blood smear image classification. MobileNet is computation-optimized and ideally suited for low-processing devices, thus best for medical image classification. The model employs depth wise separable convolutions, with fewer parameters and equivalent accuracy. The pre-trained MobileNet model, trained over large datasets like ImageNet, is employed for transfer learning—enabling it to learn useful features from medical images without being trained from scratch. This process enhances model performance while limiting computational expense and training time. In the project, MobileNet fine-tuning enables the model to be trained for leukaemia detection, with enhanced classification accuracy through learning specific patterns in blood smear images.

Keras Layers

keras.layers has an extensive list of layers required to build deep learning models. Some of these include convolution layers (used for extracting features), dense layers (used for classification), dropout layers (used for regularization), and many more. They are used to build model architectures from scratch or fine-tune pre-trained models.

Keras Optimizers (Adam)

Adam is among the most widely used optimizers in deep learning. It adjusts the learning rate of each parameter by its first and second moments (mean and variance). Thus, it is well-suited for large data and high complexity models since it has less to tune and it is less sensitive to learning rate selection..

Keras.callbacks.EarlyStopping

EarlyStopping is used to halt the training early if the validation loss plateaus. This stops the model from overfitting, saves computational resources, and avoids the model from further training when it is no longer learning useful patterns in the data.

Scikit-learn (sklearn)

Scikit-learn (sklearn) is a popular machine learning library utilized extensively in model assessment, data preprocessing, and performance analysis throughout this

project. It offers extremely useful functions for data splitting, feature scaling, and application of diverse machine learning algorithms

5.1.3. Libraries for Model Evaluation and Metrics:

Scikitlearn(accuracy_score, classification_report, train_test_split)

Scikit-learn is the most popular Python library for machine learning. It includes a collection of utilities for model estimation and data splitting:

- **Accuracy_score** calculates accuracy of a model based on comparison of predicted labels and true labels. It is used to calculate individual model and ensemble techniques' performance.
- **Classification_report** offers an extended report with precision, recall, F1-score, and support per class. It is specifically good at measuring classification models.
- **Train_test_split** is utilized to divide the data into a test set and train set so models are validated against unknown data.

TensorFlow and Keras both natively support intrinsic metrics such as categorical accuracy and loss functions, which can be monitored in real time during training.

5.1.4 Libraries for Visualization and Data Augmentation:

Matplotlib

Matplotlib is a popular Python library for generating static, animated, and interactive visualizations. It is used in the following to plot any kind of graphs like accuracy curves and confusion matrices, which are used for visually evaluating the model's performance.

Keras.preprocessing.image.ImageDataGenerator

The ImageDataGenerator class is employed for real-time data augmentation and preprocessing during training. The class produces batches of tensor image data with real-time data augmentation, including rotation, zoom, flips, and shifts. It assists in artificially increasing the size of the training set without overfitting by adding extra variability to the training data.

5.1.5. Libraries for Handling File I/O:

OS

The os module provides a file system interface. It is used in this project to handle file paths, load images, and perform directory operations such as making and deleting directories.

shutil

shutil is a high-level file operation module that involves moving and copying files. It can be used for filing organization and management, particularly when preprocessing data sets or dealing with training and validation sets.

5.1.6. Libraries for Progress Bar and Time Tracking:

tqdm

tqdm is a progress bar library used to build progress bars, applied to track the progress of operations such as model training or data processing. It is a graphical way of displaying the remaining time, which makes the user experience better when there are long-running tasks.

time

The time module is used to track how long a particular operation, say model training, takes. It is required for monitoring performance and debugging.

TensorFlow for GPU Setup:

Tensorflow is a deep learning library that can execute on both GPU and CPU computation. Models are trained quicker with TensorFlow using GPUs. TensorFlow also manages computational graphs and automatic differentiation during training. This module allows the configuration of TensorFlow for use with GPUs where possible to optimize the training process as much as possible.

Random:

One significant application of the random library is randomizing data split, i.e., randomizing the data before splitting into train and test sets. This is to prevent bias by maintaining the image distribution as uniform for both sets. Also, during data augmentation, certain operations like random rotations, flipping, and shifts can be performed using random values to create different training samples.

5.2 DATA EXTRACTION AND PREPROCESSING

5.2.1 Data Augmentation

Augmentation Techniques Used

1. **Quadrant Rotation** – The images are randomly rotated by 0°, 90°, 180°, or 270°, maintaining their structural integrity while introducing variability. They preserve pixel relationships within the image.
2. **Horizontal and Vertical Flipping** – By flipping the images along different axes, the model learns to recognize leukaemia cells regardless of their orientation.

Augmentation

```
import os
from PIL import Image, ImageOps
from tqdm.notebook import tqdm
import shutil

def augment_images(input_folder, output_folder, target_count):
    """
    Augments images in a folder using horizontal flips, vertical flips, and 90-degree rotations
    until the number of images matches the target count.

    Args:
        input_folder (str): Path to the class folder in the original dataset.
        output_folder (str): Path to the corresponding class folder in the augmented dataset.
        target_count (int): Number of images needed for this class.
    """
    image_files = [f for f in os.listdir(input_folder) if f.endswith('.jpg')]
    current_count = len(image_files)
    print(f"Current count for {input_folder}: {current_count}")

    if current_count >= target_count:
        print(f"No augmentation needed for {input_folder}.")
        return
```

```
augment_count = 0
with tqdm(total=target_count - current_count, desc=f"Augmenting {input_folder}", unit="image") as pbar:
    while current_count < target_count:
        for file_name in image_files:
            if current_count >= target_count:
                break

            img_path = os.path.join(input_folder, file_name)
            img = Image.open(img_path)

            # Apply augmentations
            augmented_images = [
                ImageOps.flip(img),  # Vertical flip
                ImageOps.mirror(img),  # Horizontal flip
                img.rotate(90),  # Rotate 90 degrees
                img.rotate(180),  # Rotate 180 degrees
                img.rotate(270),  # Rotate 270 degrees
            ]

            for aug_img in augmented_images:
                if current_count >= target_count:
                    break

                # Save augmented image
                aug_file_name = f"{file_name.split('.')[0]}_aug{augment_count}.jpg"
                aug_img_path = os.path.join(output_folder, aug_file_name)
                aug_img.save(aug_img_path)
                augment_count += 1
                current_count += 1
```

```
# Create the augmented dataset directory
if os.path.exists(augmented_dataset_path):
    shutil.rmtree(augmented_dataset_path)  # Remove existing augmented dataset directory if necessary
os.makedirs(augmented_dataset_path, exist_ok=True)

# Copy original dataset to augmented dataset directory
for class_name in os.listdir(dataset_path):
    original_class_path = os.path.join(dataset_path, class_name)
    augmented_class_path = os.path.join(augmented_dataset_path, class_name)
    os.makedirs(augmented_class_path, exist_ok=True)

    # Copy original images
    for file_name in os.listdir(original_class_path):
        if file_name.endswith('.jpg'):
            shutil.copy(
                os.path.join(original_class_path, file_name),
                os.path.join(augmented_class_path, file_name)
            )

# Calculate the max number of images across all classes in the original dataset
class_dirs = [os.path.join(dataset_path, class_name) for class_name in os.listdir(dataset_path)]
class_counts = {class_dir: len([f for f in os.listdir(class_dir) if f.endswith('.jpg')]) for class_dir in class_dirs}
max_count = max(class_counts.values())
print(f"Maximum image count among classes: {max_count}")

# Augment classes with fewer images
for class_dir, count in tqdm(class_counts.items(), desc="Processing classes", unit="class"):
    class_name = os.path.basename(class_dir)
    augmented_class_dir = os.path.join(augmented_dataset_path, class_name)
    if count < max_count:
        augment_images(class_dir, augmented_class_dir, max_count)

print("Dataset augmentation completed.")
'''
```

5.2.2 Data Extraction

Here, We directly extract he augmented data for the Neutrosophic Enhancements.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] dataset_path = "/content/drive/MyDrive/leukaemia_augmented"
```

```
import os

if os.path.exists(dataset_path):
    print("Dataset found! Listing categories:")
    print(os.listdir(dataset_path))  # List the class folders
else:
    print("Dataset not found. Please check the path.")
```

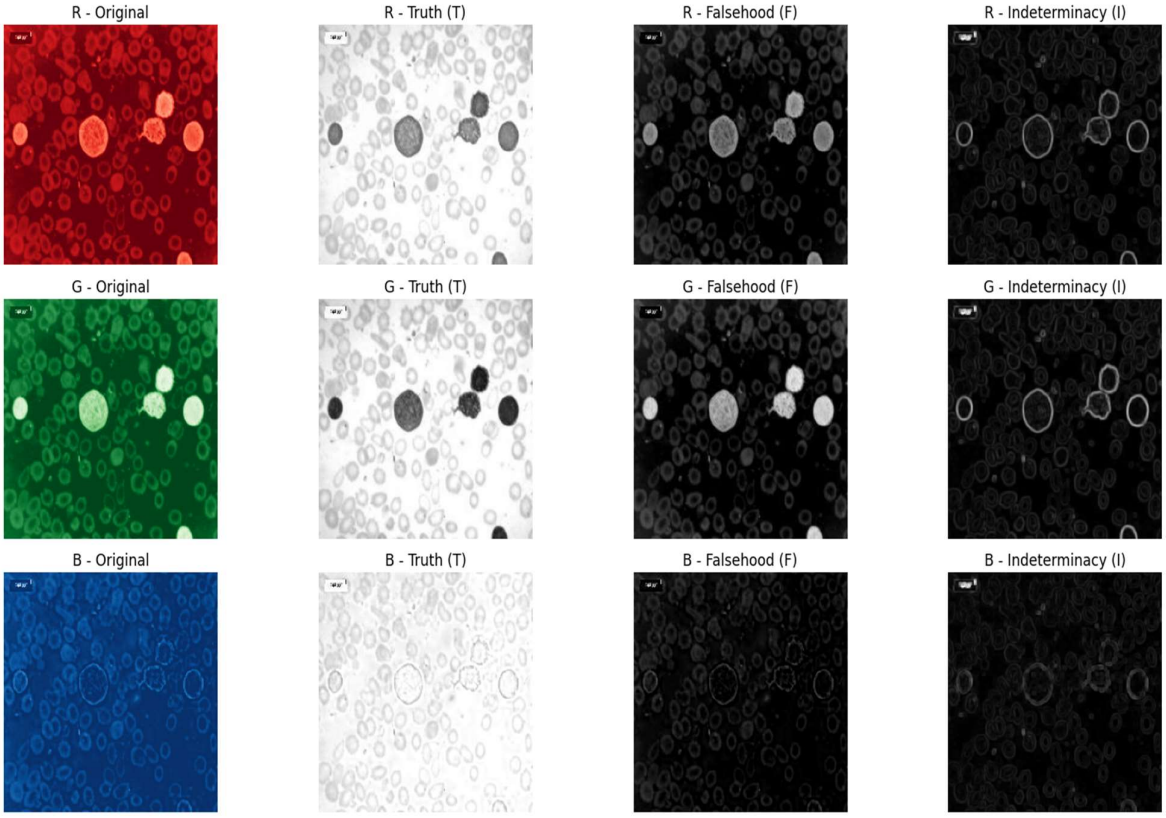
```
Dataset found! Listing categories:
['Benign', 'Early', 'Pro', 'Pre']
```

```python
import os

# Define the dataset path
dataset_pathh = "/content/drive/MyDrive/leukaemia_augmented"

# Verify dataset structure
for root, dirs, files in os.walk(dataset_pathh):
    level = root.replace(dataset_pathh, '').count(os.sep)
    indent = ' ' * 4 * level
    print(f"{indent}{os.path.basename(root)}/")
    sub_indent = ' ' * 4 * (level + 1)
    for f in files[:5]:  # Show only the first 5 files per folder for brevity
        print(f"{sub_indent}{f}")
    if len(files) > 5:
        print(f"{sub_indent}... ({len(files)} files)")
```

```
leukaemia_augmented/
    Benign/
        WBC-Benign-004.jpg
        WBC-Benign-001.jpg
        WBC-Benign-003.jpg
        WBC-Benign-002.jpg
        WBC-Benign-018.jpg
        ... (985 files)
    Early/
        WBC-Malignant-Early-001.jpg
        WBC-Malignant-Early-047.jpg
        WBC-Malignant-Early-015.jpg
        WBC-Malignant-Early-004.jpg
        WBC-Malignant-Early-051.jpg
        ... (985 files)
    Pro/
        WBC-Malignant-Pro-002.jpg
        WBC-Malignant-Pro-001.jpg
        WBC-Malignant-Pro-003.jpg
        WBC-Malignant-Pro-004.jpg
        WBC-Malignant-Pro-005.jpg
        ... (985 files)
    Pre/
        WBC-Malignant-Pre-004.jpg
        WBC-Malignant-Pre-003.jpg
        WBC-Malignant-Pre-002.jpg
        WBC-Malignant-Pre-001.jpg
        WBC-Malignant-Pre-024.jpg
        ... (985 files)
```

5.3 NEUTROSOPHIC ENHANCEMENTS

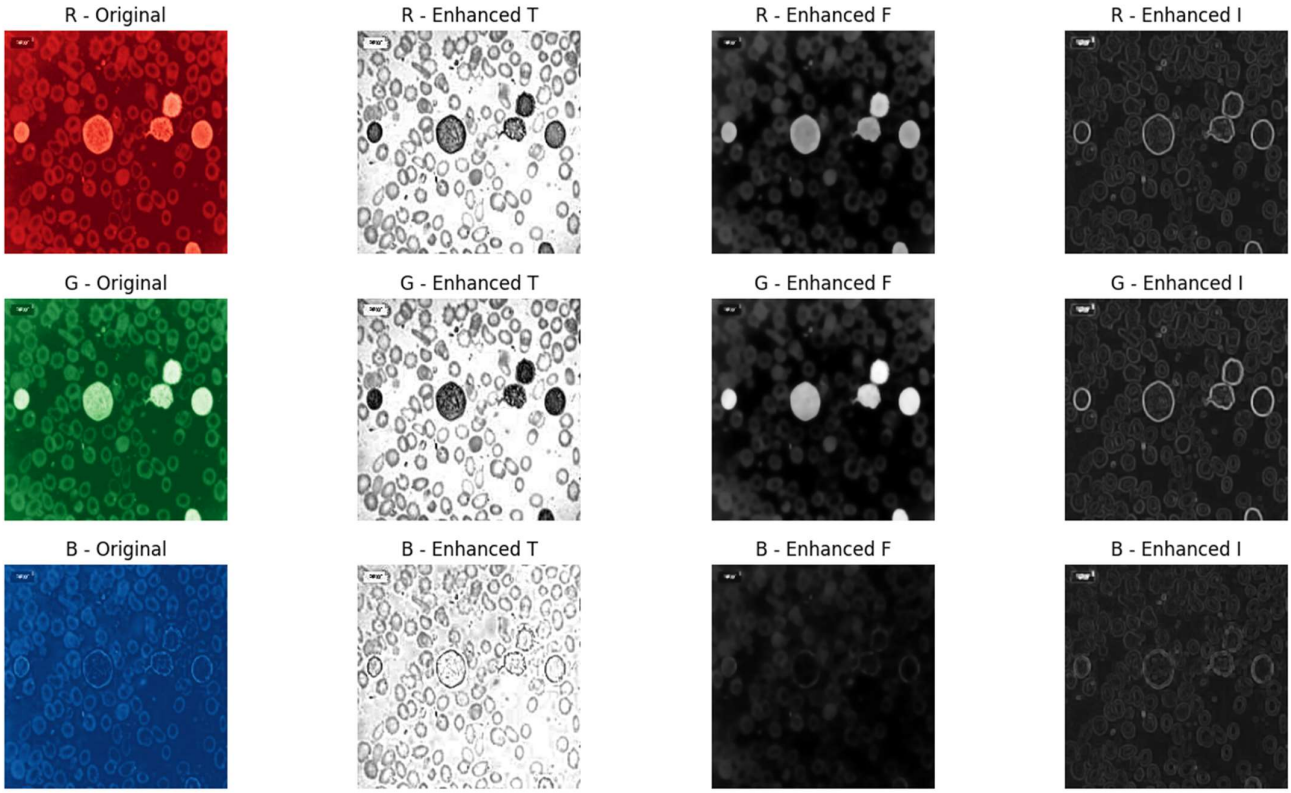
5.3.1 Applying Neutrosophic Tranformation to each of the channels of RGB



5.3.2 Further Enhancing the Neutrosophic Enhancement

- Enhancing **True(T)** part using CLAHE and further sharpening it.
- Reducing noise in **Falsehood(F)** using TVM.
- Suppressing uncertainty in **Indeterminacy(I)** using Wavelet Shrinkage Denoising and further enhancing it with Multi-Scale Wavelet Sharpening.

```
import cv2
import numpy as np
from skimage.restoration import denoise_tv_chambolle, denoise_tv_bregman, denoise_wavelet
from skimage.filters import unsharp_mask
import pywt

def wavelet_sharpen(image, wavelet='haar', level=2, alpha=1.2):
    coeffs = pywt.wavedec2(image, wavelet, level=level)
    coeffs = list(coeffs)
    for i in range(1, len(coeffs)):
        coeffs[i] = tuple(alpha * c for c in coeffs[i])
    return pywt.waverec2(coeffs, wavelet)

def enhance_neutrosophic(T, F, I):
    # Enhance T using CLAHE
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    T_enhanced = clahe.apply((T * 255).astype(np.uint8)) / 255.0

    # Further sharpen T
    T_enhanced = unsharp_mask(T_enhanced, radius=1.5, amount=1.5)

    # Reduce noise in F using TVM
    F_enhanced = denoise_tv_chambolle(F, weight=0.1, channel_axis=None)

    # Suppress uncertainty in I using Wavelet Shrinkage Denoising
    I_enhanced = denoise_wavelet(I, method='BayesShrink', mode='soft', channel_axis=None, rescale_sigma=True)

    # Further enhance I with Multi-Scale Wavelet Sharpening
    I_enhanced = wavelet_sharpen(I_enhanced, wavelet='db2', level=2, alpha=1.3)

    return T_enhanced, F_enhanced, I_enhanced

# Apply enhancement to each channel's Neutrosophic components
T_R_enh, F_R_enh, I_R_enh = enhance_neutrosophic(T_R, F_R, I_R)
T_G_enh, F_G_enh, I_G_enh = enhance_neutrosophic(T_G, F_G, I_G)
T_B_enh, F_B_enh, I_B_enh = enhance_neutrosophic(T_B, F_B, I_B)

# Display the enhanced components
fig, ax = plt.subplots(3, 4, figsize=(16, 9))
components = ["Original", "Enhanced T", "Enhanced F", "Enhanced I"]
channels = [(R, T_R_enh, F_R_enh, I_R_enh), (G, T_G_enh, F_G_enh, I_G_enh), (B, T_B_enh, F_B_enh, I_B_enh)]
colors = ["Reds", "Greens", "Blues"]

for i in range(3):
    for j in range(4):
        ax[i, j].imshow(channels[i][j], cmap=colors[i] if j == 0 else "gray")
        ax[i, j].set_title(f"{['R', 'G', 'B'][i]} - {components[j]}")
        ax[i, j].axis("off")

plt.show()
```



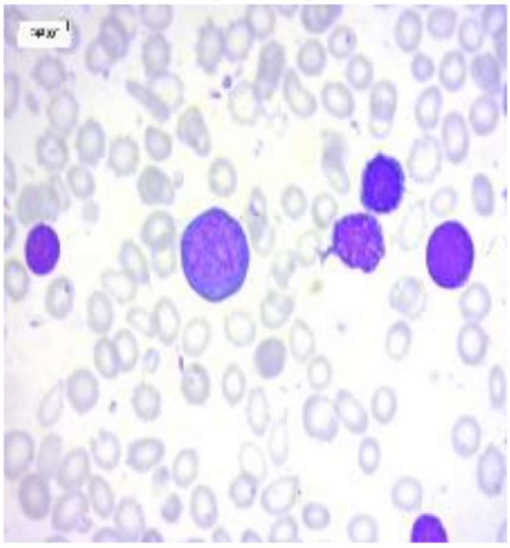
5.3.3 **Reconstructing a single image (Example Reconstruction)**

Since the improvement process enhances each component individually in the Red, Green, and Blue (RGB) channels, the need to combine them back to a single representation of the image arises. Reconstruction guarantees that the important features derived from all the components are maintained while color homogeneity and structural consistency are guaranteed.

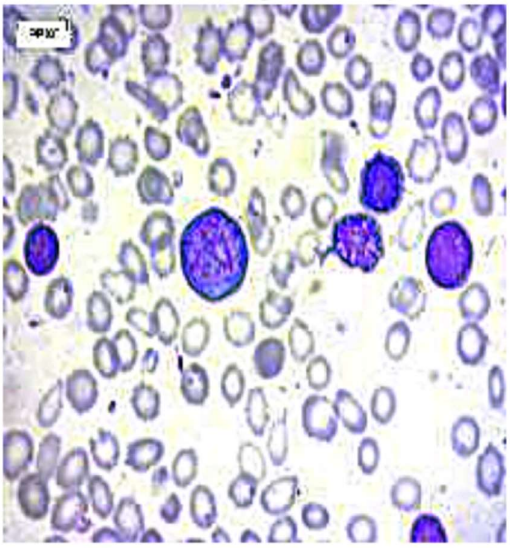
Reconstruction proceeds by summing the enhanced T, F, and I components for each of the individual color channels (R, G and B) in such a way that their enhanced counterparts increase the image as a whole. After contrast enhancement (T), noise removal (F), and uncertainty enhancement (I) of the channels, the enhanced channels are linearly summed to reconstruct a better version of the original image. Blending is performed to preserve the strengths of the Neutrosophic transformation and enhancement process, including better visibility of features, sharper cell boundaries, and fewer distortions.

1. **Enhancement of True(T) component of the colour channel**

Original Image

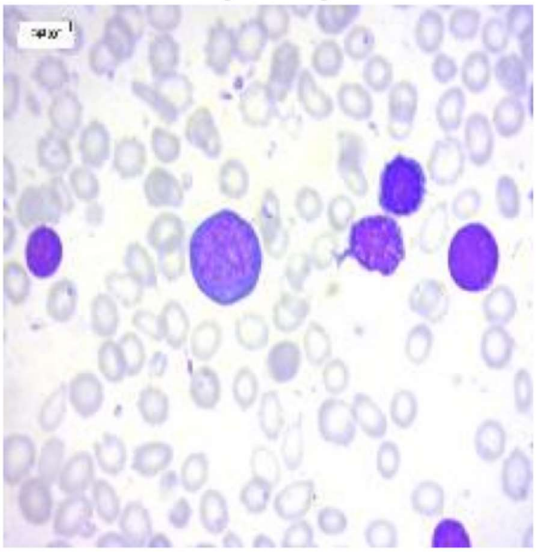


Reconstructed Image (Neutrosophic)

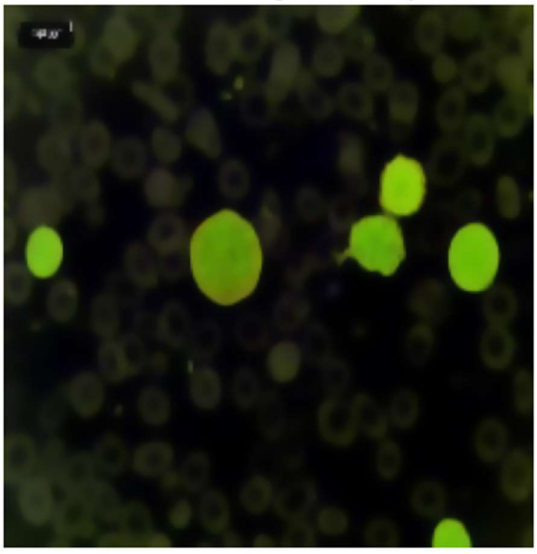


2. **Enhancement of Falsehood(F) component of the colour channel**

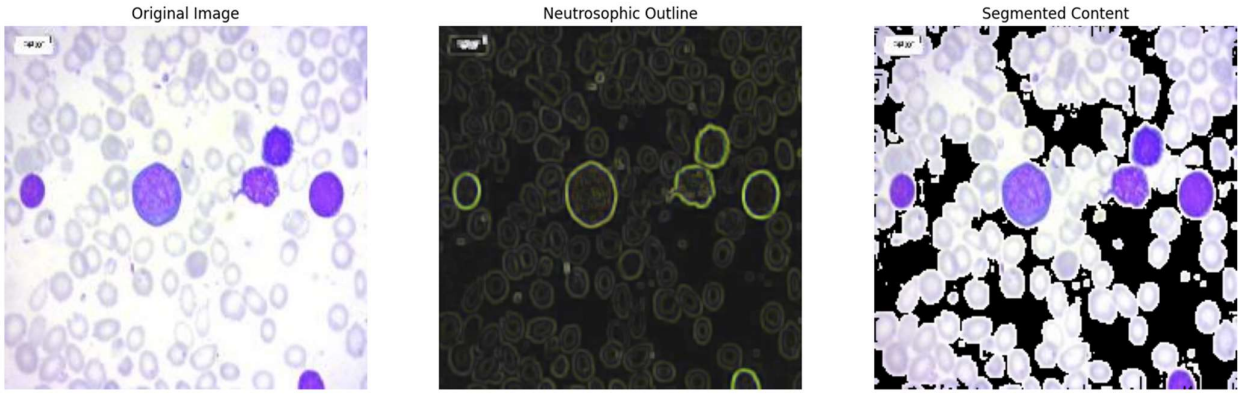
Original Image



Reconstructed Image (Neutrosophic)



3. Enhancement of Indeterminacy(I) component of the colour channel



After the improved Red, Green, and Blue components are rebuilt individually, these are then layered on top of each other to create the final improved image. This is indeed the combination of all processed data into one high-quality output, a refined input for deep learning models.

4. Final Reconstruction of the Image:

```
def reconstruct_channel(T, F, I):
    """
    Reconstruct a single-channel image using enhanced T, F, and I.
    Ensures pixel values remain in the valid range [0, 1].
    """
    reconstructed = T - F + I
    reconstructed = np.clip(reconstructed, 0, 1)  # Ensure values remain valid
    return reconstructed

# Reconstruct R, G, B channels
R_reconstructed = reconstruct_channel(T_R_enh, F_R_enh, I_R_enh)
G_reconstructed = reconstruct_channel(T_G_enh, F_G_enh, I_G_enh)
B_reconstructed = reconstruct_channel(T_B_enh, F_B_enh, I_B_enh)

# Merge reconstructed channels into final image
final_image = np.stack([R_reconstructed, G_reconstructed, B_reconstructed], axis=-1)

# Display original vs reconstructed image
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax[0].imshow(image)
ax[0].set_title("Original Image")
ax[0].axis("off")

ax[1].imshow(final_image)
ax[1].set_title("Reconstructed Image (Neutrosophic)")
ax[1].axis("off")

plt.show()
```

5.3.4 Enhancement on Complete Dataset

```
# Define the output directory
neutrosophic_dataset_path = "/content/neutrosophic_dataset"
os.makedirs(neutrosophic_dataset_path, exist_ok=True)

# Function to process and save images
def process_and_save_images(input_path, output_path):
    categories = os.listdir(input_path)

    for category in tqdm(categories, desc="Processing Categories"):
        category_path = os.path.join(input_path, category)
        output_category_path = os.path.join(output_path, category)
        os.makedirs(output_category_path, exist_ok=True)

        for image_name in tqdm(os.listdir(category_path), desc=f"Processing {category}", leave=False):
            # Load image
            image_path = os.path.join(category_path, image_name)
            image = cv2.imread(image_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

            # Extract channels
            R, G, B = cv2.split(image)

            # Apply neutrosophic processing
            T_R, F_R, I_R = neutrosophic_transform(R)
            T_G, F_G, I_G = neutrosophic_transform(G)
            T_B, F_B, I_B = neutrosophic_transform(B)

            T_R_enh, F_R_enh, I_R_enh = enhance_neutrosophic(T_R, F_R, I_R)
            T_G_enh, F_G_enh, I_G_enh = enhance_neutrosophic(T_G, F_G, I_G)
            T_B_enh, F_B_enh, I_B_enh = enhance_neutrosophic(T_B, F_B, I_B)

            R_reconstructed = reconstruct_channel(T_R_enh, F_R_enh, I_R_enh)
            G_reconstructed = reconstruct_channel(T_G_enh, F_G_enh, I_G_enh)
            B_reconstructed = reconstruct_channel(T_B_enh, F_B_enh, I_B_enh)

            final_image = np.stack([R_reconstructed, G_reconstructed, B_reconstructed], axis=-1)
            final_image = (final_image * 255).astype(np.uint8)  # Convert to uint8

            # Save image
            save_path = os.path.join(output_category_path, image_name)
            cv2.imwrite(save_path, cv2.cvtColor(final_image, cv2.COLOR_RGB2BGR))

# Process dataset
process_and_save_images(dataset_path, neutrosophic_dataset_path)
print("All neutrosophic images have been saved!")
```

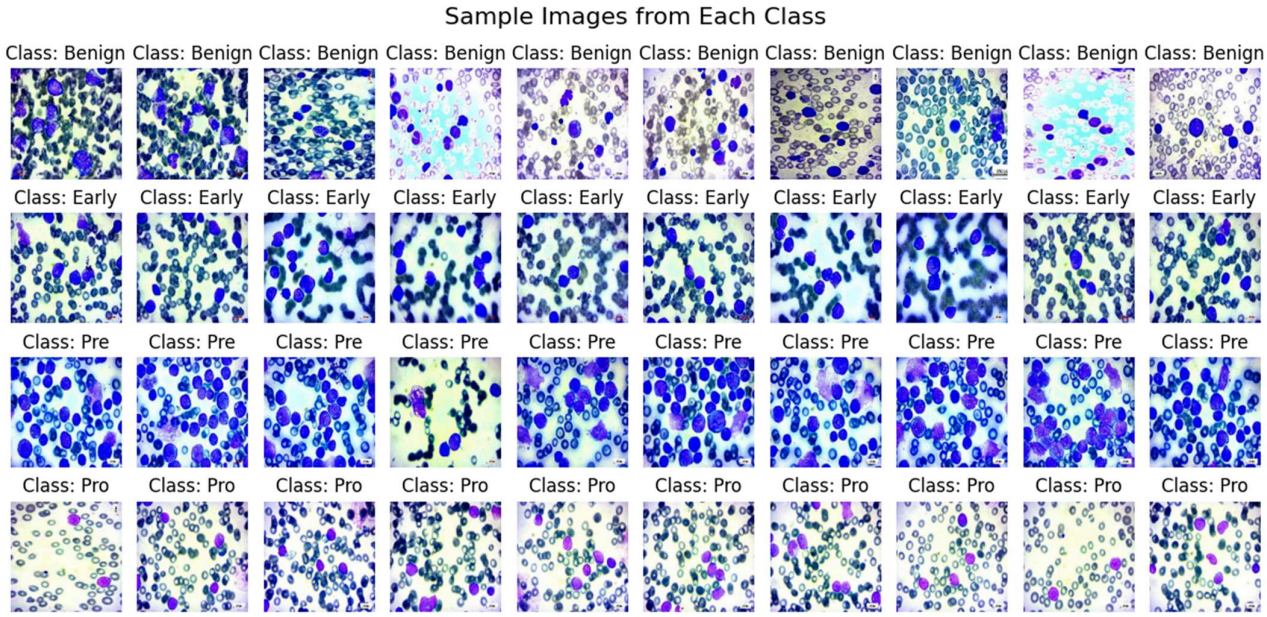
```
Processing Categories: 100% 4/4 [06:46<00:00, 101.30s/it]
```

```
All neutrosophic images have been saved!
```

5.3.5 Extracting the Neutrosophic data for feature extractions using Transfer Learning Models.

```python
import os
import matplotlib.pyplot as plt

# Define paths to each class
base_dir = "/content/neutrosophic_dataset"
class_dirs = {
    'Benign': os.path.join(base_dir, 'Benign'),
    'Early': os.path.join(base_dir, 'Early'),
    'Pre': os.path.join(base_dir, 'Pre'),
    'Pro': os.path.join(base_dir, 'Pro')
}

# Count the number of images in each class
for class_name, path in class_dirs.items():
    num_images = len(os.listdir(path))
    print(f"Class '{class_name}' has {num_images} images.")

# Display a few images from each class
def display_sample_images(class_dirs, num_images=10):
    """Display sample images from each class."""
    fig, axes = plt.subplots(len(class_dirs), num_images, figsize=(12, 6))
    fig.suptitle("Sample Images from Each Class", fontsize=16)

    for i, (class_name, path) in enumerate(class_dirs.items()):
        images = os.listdir(path)[:num_images]
        for j, img_name in enumerate(images):
            img = plt.imread(os.path.join(path, img_name))
            axes[i, j].imshow(img)
            axes[i, j].set_title(f"Class: {class_name}")
            axes[i, j].axis('off')

    plt.tight_layout()
    plt.show()

# Display images
display_sample_images(class_dirs)
```

5.3.6 Sample Images after the Neutrosophic Transformation & Enhancement



5.4 MODEL

```
from tensorflow.keras.applications import ResNet50, DenseNet121, EfficientNetB3, MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import AUC, Precision, Recall

# Define input layer
input_tensor = Input(shape=(224, 224, 3))

# Initialize the ResNet model with pre-trained weights
base_model = MobileNetV2(weights='imagenet', include_top=False, input_tensor=input_tensor)

# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False

# Create the model
x = GlobalAveragePooling2D()(base_model.output)  # Change from Flatten to GlobalAveragePooling2D
x = Dense(128, activation='relu')(x)
output = Dense(len(class_names), activation='softmax')(x)  # Output layer for multi-class classification

# Define the complete model
model = Model(inputs=input_tensor, outputs=output)

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Display the model summary
model.summary()
```

5.5 TRAINING

1. **Data Splitting:**
   The data is separated into training, validation, and test sets. The split retains a portion of data for testing without exposure during training. The data was split into training(80%), validation(10%) and test(10%)

2. **Loss Function:**
   The Categorical Cross-entropy was used to handle class imbalance by emphasizing overlap between predicted and ground truth.

3. **Optimizer:**
   The Adam optimizer was used with an initial learning rate of 0.0010, chosen for its adaptive learning rate adjustments. Learning rate adjustments can be added to improve convergence.

4. **Batch Processing:**
   A batch size of 32 was used, balancing memory usage and computational efficiency.

5. **Early Stopping:**
   This is for loss validation or accuracy. In case there is no improvement after a specified number of epochs (patience level), training is halted early. This avoids overfitting and saves training time.

```
# Set training parameters
epochs = 50  # You can adjust the number of epochs
batch_size = 32

# Define EarlyStopping callback
early_stopping = EarlyStopping(
    monitor='val_accuracy',  # metric to monitor
    patience= epochs + 1,          # number of epochs to wait for improvement
    restore_best_weights=True  # restores weights from the best epoch
)

# Train the model with early stopping
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=epochs,
    batch_size=batch_size,
    callbacks=[early_stopping],  # add the callback here
    verbose=1
)
```

```python
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

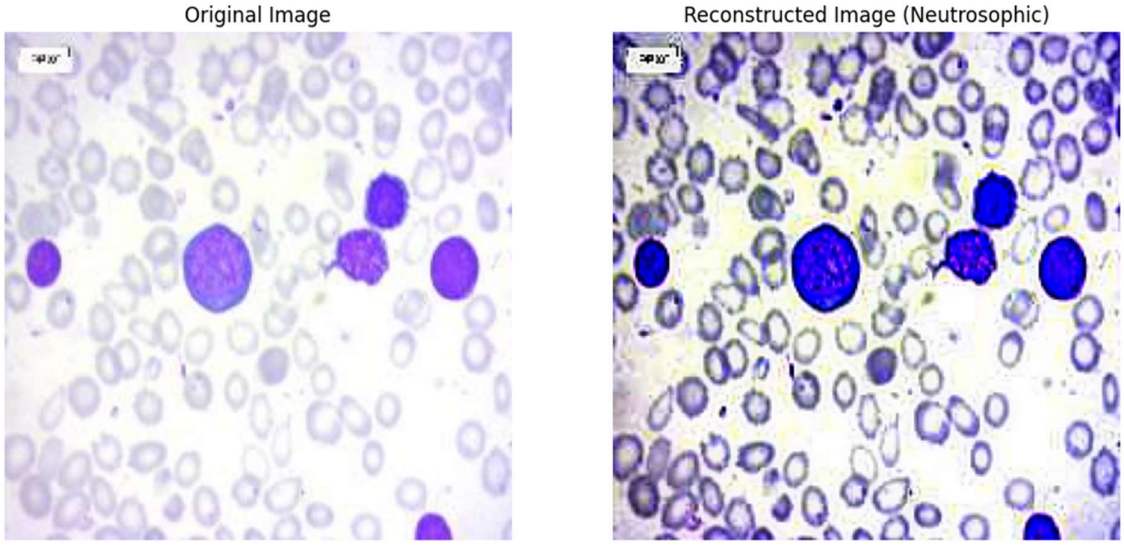
5.6 PREDICTION

```python
# Number of images to display
num_images_to_display = 10
random_indices = random.sample(range(X_test.shape[0]), num_images_to_display)

# Prepare the figure for plotting
plt.figure(figsize=(15, 8))

for i, idx in enumerate(random_indices):
    # Get the image and its true label
    img = X_test[idx]
    true_label = label_encoder.inverse_transform([np.argmax(y_test[idx])])[0]

    # Make a prediction
    img_expanded = np.expand_dims(img, axis=0)  # Add batch dimension
    predictions = model.predict(img_expanded)
    predicted_label = label_encoder.inverse_transform([np.argmax(predictions)])[0]

    # Display the image and prediction
    plt.subplot(1, num_images_to_display, i + 1)
    plt.imshow(img)
    plt.title(f"True: {true_label}\nPredicted: {predicted_label}")
    plt.axis('off')

plt.tight_layout()
plt.show()
```

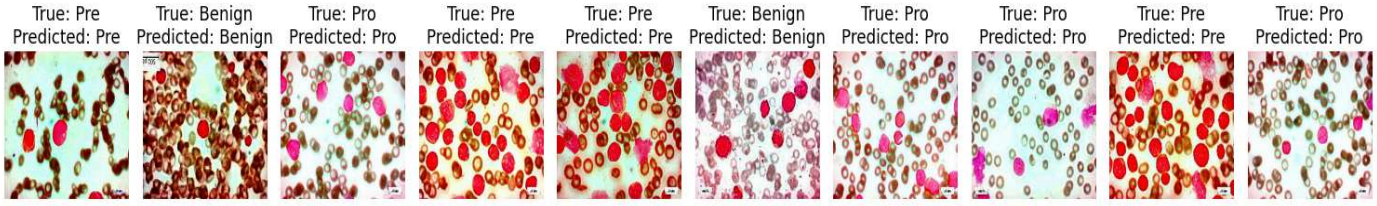
Chapter 6

Results

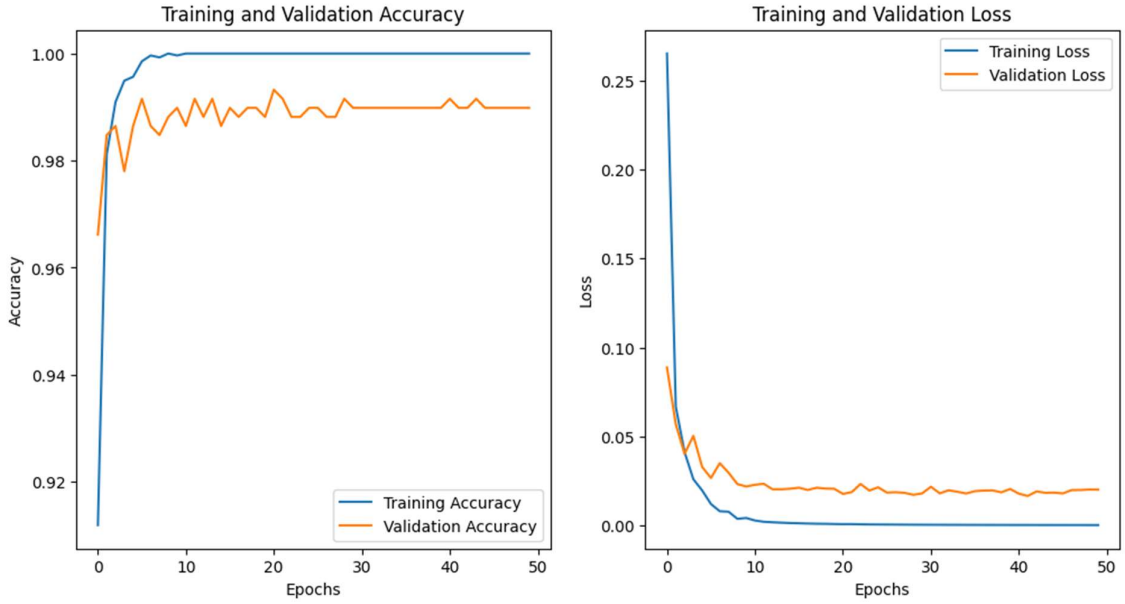
6.1 IMAGE RECONSTRUCTION AFTER ENHANCEMENTS



6.2 PREDICTION



6.3 TRAINING AND VALIDATION PERFORMANCE



6.4 TESTING PERFORMANCE

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9907 - loss: 0.0243
Test Loss: 0.030882379040122032
Test Accuracy: 0.9864636063575745
```

6.5 PERFORMANCE METRICS

Performance metrics are essential for measuring the effectiveness of machine learning models, particularly in classification problems such as the problem employed in this research. They inform one about the performance of the model and provide feedback on areas that need improvement. In this section, we present the important performance metrics employed for measuring the accuracy and confidence of the image classification system.

1. **Accuracy:** Accuracy is the most widely used measure in classification problems, and it denotes the number of correct predictions divided by the total number of predictions. Although accuracy provides an overall sense of model performance, it is not always enough, especially in cases of imbalanced datasets.

2. **Precision:** Precision refers to how many of the positive labels predicted are correct. It is valuable when the false positive cost is high. It is particularly beneficial in tasks where the model's false positive rate must be low.

3. **Recall (Sensitivity):** Recall, or sensitivity, indicates the number of actual positive instances correctly predicted by the model. It is appropriate when false negatives must be reduced, such as in medical diagnosis or detecting fraud.

4. **F1-Score:** The F1-score is the harmonic mean between precision and recall, giving an average of the two. It is a useful measure when there is an imbalance in the class distribution since it considers both false positives and false negatives.

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 7s 167ms/step
              precision    recall  f1-score   support

      Benign       0.98      0.98      0.98       133
       Early       0.98      0.98      0.98       151
         Pre       0.99      0.99      0.99       156
         Pro       1.00      0.99      1.00       151

    accuracy                           0.99       591
   macro avg       0.99      0.99      0.99       591
weighted avg       0.99      0.99      0.99       591
```

Chapter 7

Discussions

The findings of this research show that the model was able to learn to predict leukaemia cases with great accuracy. The training phase showed a gradual increase in accuracy, while the loss reduced progressively, indicating successful learning. The validation accuracy trailed the training accuracy but showed occasional oscillations, which could be indicative of the occurrence of overfitting. This is a phenomenon where the model becomes over-tuned in the training set and loses the capacity to generalize to new samples.

7.1 EVALUATION OF RESULTS

The performance of the model was evaluated based on important metrics including accuracy, precision, recall, and F1-score. The high accuracy value obtained shows that the model successfully differentiates leukaemia-positive from leukaemia-negative cases. Precision and recall values also demonstrate the model's capability to reduce false positives and false negatives, which is significant in medical diagnostics. A high precision value indicates that the model possesses a strong ability to accurately classify leukaemia cases without misclassifying healthy samples, while a high recall value indicates that the majority of actual leukaemia cases were identified. The F1-score, which is a balance between precision and recall, validated the overall effectiveness of the model.

Yet, some of the misclassifications highlight areas where additional optimization is needed. The analysis of the confusion matrix showed particular instances where the model performed poorly, probably because of unclear features in some samples of leukaemia. There were images with low contrast or noisy content, which created challenges for the model to pick up on useful features. A closer examination of these cases can help in improving the decision-making process of the model, perhaps by the use of sophisticated preprocessing methods. Furthermore, the addition of more inclusive and representative samples of data could make the model more resilient and help in the classification of different subtypes of leukaemia.

7.2 CHALLENGES ENCOUNTERED

In the development and training stage, some of the challenges encountered included managing class imbalance in the dataset. Many leukaemia datasets present an unbalanced distribution of positive and negative cases, and such a situation could lead to class biasing the model towards the majority class. To mitigate this, class weighting and oversampling of minority cases were implemented.

Another difficulty was generalization to actual situations, since medical images can be quite different in quality and origin. Variations in imaging conditions, including lighting, resolution, and staining methods, created challenges in obtaining a model that works well across various datasets. Variations in cell morphology and staining characteristics across various medical institutions also introduced variability that the model needed to accommodate. Improving data augmentation techniques, e.g., rotation and flipping, may assist in making the model more resilient against these variations.

Computational resource constraints also came into play during training. Deep learning models, especially convolutional neural networks, are computationally intensive and memory-intensive. Training the model on high-resolution medical images made the computation even heavier, and the use of efficient methods like batch normalization and optimized learning rates became necessary to keep the training efficient. Using cloud-based GPU resources or distributed training methods may also improve training efficiency in future research.

7.3 COMPARISON WITH EXISTING METHODS

In comparison to conventional leukaemia detection techniques, the new deep learning method has several advantages. Traditional techniques, including pathologist manual microscopic inspection, are time-consuming and subject to inter-observer variability. Human analysis is based on subjective judgment, which can result in inconsistent diagnosis. Moreover, conventional statistical techniques and machine learning classifiers like Support Vector Machines (SVM) and Random Forests demand heavy feature engineering, wherein the domain experts have to manually extract useful features from the images. This is time-consuming and does not always represent complex patterns in the data.

Deep learning architectures, especially CNN-based models, are best at learning hierarchical features from images automatically, resulting in better performance. In contrast to conventional approaches, CNNs are capable of learning spatial hierarchies of features, enhancing classification accuracy without requiring manual feature selection. The application of the proposed model showed enhanced classification accuracy compared to these conventional approaches, showing the potential of AI-based diagnostics in leukaemia detection. Moreover, the ability of deep learning models to continuously improve with larger datasets makes them a promising alternative to conventional techniques.

Nonetheless, deep learning techniques also come with challenges when compared to conventional methods. One major limitation is that they require enormous quantities of labeled data, as medical image annotation is a labor-intensive task involving experts. Moreover, deep learning models are not interpretable, with it being hard to know how decisions are arrived at. Combining domain knowledge and explainability tools, like Grad-CAM visualizations, would potentially increase the reliability and acceptability of the model within clinical practice. Next steps could include hybrid methods that merge deep learning with conventional feature-based techniques in an effort to benefit from both.

7.4 POTENTIAL IMPROVEMENTS

For reducing overfitting, methods including data augmentation, dropout, and L2 regularization can be tried. Data augmentation adds randomness in the training data, causing the model to learn more generalized features instead of memorizing the patterns. Dropout randomly shuts down some neurons while training, compelling the model to learn more general representations. L2 regularization prevents overcomplicated models by charging more for large weight values, resulting in better generalization. In addition, applying cross-validation methods may serve to further elucidate the model's strength across varying data splits.

Early stopping was used effectively to avoid unnecessary training after the optimal point. Validation loss was tracked to stop training once the model ceased to improve, minimizing overfitting. The method ensures that the resulting model is balanced between learning and generalization, avoiding deterioration in real-world performance. Yet, additional fine-tuning of the stopping threshold would be helpful to maximize model performance.

Even though the model achieved robust classification performance, misclassifications still occurred. Examining these instances would reveal important insights into the weaknesses of the present method. Certain misclassifications can be traced back to noisy or unclear images in the database, indicating that some data preprocessing methods like noise removal and contrast stretching would lead to higher precision. The performance of the model can also be furthered by incorporating feature extraction methods focusing on salient leukaemia-related features in medical images.

7.5 SUMMARY

The result evaluation showed high accuracy in classification, with high precision, recall, and F1-score, confirming the effectiveness of the model in detecting leukaemia. Minor misclassifications also suggested further optimization, especially in processing uncertain or low-contrast images. The confusion matrix indicated points of improvement in separating borderline cases.

There were a few challenges faced, which included class imbalance in the data, inconsistencies in image quality, and computational constraints. Methods such as class weighting and data augmentation assisted in lessening these effects, but improvements such as synthetic data generation can further optimize performance.

Compared to standard approaches such as manual microscopic screening and machine learning classifiers, deep learning performed better in accuracy and feature extraction compared to them. In contrast to conventional methods with manual feature engineering, CNNs learned complex patterns automatically, enhancing diagnosis. While deep learning algorithms require large quantities of labeled datasets and are less interpretable, they pose hurdles to adoption within clinical practice.

Potential enhancements are deep data augmentation, dropout regularization, and ensemble learning for enhancing generalization. Furthermore, incorporating explainability tools such as Grad-CAM would further increase model transparency and clinical trust. Validation using heterogeneous datasets and hybrid methods combining deep learning with conventional methods for stable leukaemia detection is what future work should prioritize.

Chapter 8

Conclusion

The conclusion summarizes the major findings of the research, providing insight into the effectiveness of the Neutrosophic method applied for image classification. It also presents possible directions for future work to improve the existing methodology and solve problems that were faced in the study.

8.1 SUMMARY OF FINDINGS

In this research, the use of deep learning in the detection of leukaemia through medical imaging data was examined. From the results, it is clear that the suggested model had high accuracy, precision, recall, and F1-score, validating its capability to efficiently classify leukaemia cases. The consistent reduction of training loss and the close convergence between training and validation performance reveal effective learning and generalization. However, small oscillations in validation accuracy indicate the occurrence of overfitting, which might be overcome using additional optimization methods like data augmentation and regularization.

While the results were encouraging, there were several challenges faced, including class imbalance, heterogeneity of medical image quality, and computational constraints. Mitigating these by class weighting, better preprocessing, and effective training techniques improved model performance. Yet, further improvements, including synthetic data generation and sophisticated feature extraction methods, can be achieved in terms of classification accuracy.

In comparison with conventional leukaemia detection techniques, the deep learning method has considerable automation and precision benefits. Contrary to labor-intensive manual microscopic screening, which involves variability and takes a lot of time, convolutional neural networks learn complex patterns automatically, which results in more uniform and consistent predictions. Despite this, the issues of having large amounts of labelled data and model interpretability need to be overcome in order to make the method clinically usable in practice.

Ultimately, this research makes the case for the promise of deep learning leukaemia detection as an attractive solution towards enhanced early detection and treatment preparation. With refinement and validation to come, artificial intelligence-based medical diagnostics may come to be used as a very significant factor enhancing the accuracy and effectiveness of leukaemia classification in actual clinical practice.

8.2 FUTURE WORKS

Future work will explore alternative architectures, e.g., transformer models or ensemble learning approaches, to further enhance classification performance. Transformer models, with their highly acclaimed effectiveness on computer vision tasks, might help to preserve fine spatial relationships between medical images. Ensemble learning, where an ensemble of models is combined, potentially would make the output more trustworthy by leveraging the strengths of varied architectures. Another area for improvement is the inclusion of domain adaptation techniques, which would allow the model to generalize better across different medical imaging datasets.

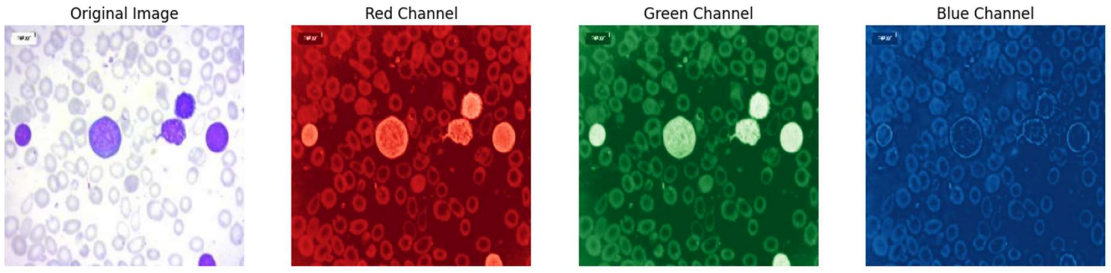
Furthermore, real-world deployment requires verification on real-world data to test the robustness of the model and its practicability in a clinical setting. Reference to medical practitioners would ensure that the model reflects expert judgments, reducing misclassifications in real-world settings. Explainability techniques, such as Grad-CAM, can further be employed to provide visualizations of the model's decisions, fostering trust in AI-powered leukaemia diagnostics.

In short, while the model has achieved encouraging performance in leukaemia classification, further enhancements are required to enhance its generalizability and resilience. With the addition of state-of-the-art techniques such as data augmentation, ensemble learning, and explainability methods, the model can be optimized to generate more accurate and clinically relevant predictions. Findings from this work contribute to continued research on the application of AI-based medical diagnostic methods, e.g., the contribution of deep learning towards enhancing leukaemia early detection and treatment.

Appendices

Appendix 1: Loading image in R, G, B Channels

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
import random

# Path to dataset
dataset_path = "/content/drive/MyDrive/leukaemia_augmented"

# Get a random image from the dataset
category = random.choice(os.listdir(dataset_path))  # Pick a random class
image_path = os.path.join(dataset_path, category, random.choice(os.listdir(os.path.join(dataset_path, category))))

# Load image in RGB format
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Extract R, G, B channels
R, G, B = cv2.split(image)

# Display images
fig, ax = plt.subplots(1, 4, figsize=(16, 4))
ax[0].imshow(image)
ax[0].set_title("Original Image")
ax[1].imshow(R, cmap="Reds")
ax[1].set_title("Red Channel")
ax[2].imshow(G, cmap="Greens")
ax[2].set_title("Green Channel")
ax[3].imshow(B, cmap="Blues")
ax[3].set_title("Blue Channel")

for a in ax:
    a.axis("off")

plt.show()
```



Appendix 2: Source Code for Neutrosophic Transformation

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import uniform_filter

def neutrosophic_transform(channel, window_size=3):
    """
    Convert an image channel to the Neutrosophic domain (T, F, I) with sharper I.

    Args:
        channel (np.array): 2D NumPy array (Single-channel image)
        window_size (int): Window size for local mean and variance.

    Returns:
        T, F, I components as NumPy arrays
    """
    # Normalize pixel values to [0,1]
    channel = channel.astype(np.float32) / 255.0

    # Compute local mean using a box filter
    local_mean = uniform_filter(channel, size=window_size)

    # Compute local variance
    local_mean_sq = uniform_filter(channel**2, size=window_size)
    local_std = np.sqrt(local_mean_sq - local_mean**2 + 1e-8)  # Numerical stability

    # Compute Neutrosophic Components
    T = np.clip(channel, 0, 1)  # Truth: Original pixel intensity
    I = local_std  # Indeterminacy: Local standard deviation
    F = 1 - T  # Falsehood: Complement of Truth

    return T, F, I

# Apply Neutrosophic transform to each channel
T_R, F_R, I_R = neutrosophic_transform(R)
T_G, F_G, I_G = neutrosophic_transform(G)
T_B, F_B, I_B = neutrosophic_transform(B)

# Display results
fig, ax = plt.subplots(3, 4, figsize=(16, 9))
components = ["Original", "Truth (T)", "Falsehood (F)", "Indeterminacy (I)"]
channels = [(R, T_R, F_R, I_R), (G, T_G, F_G, I_G), (B, T_B, F_B, I_B)]
colors = ["Reds", "Greens", "Blues"]

for i in range(3):
    for j in range(4):
        ax[i, j].imshow(channels[i][j], cmap=colors[i] if j == 0 else "gray")
        ax[i, j].set_title(f"{['R', 'G', 'B'][i]} - {components[j]}")
        ax[i, j].axis("off")

plt.tight_layout()  # Prevent title overlap
plt.show()
```

Further Enhancement

6 Further Enhancing the Neutrosophic Enhancement

6.3 Enhancing **True(T)** part using CLAHE and further sharpening it.

6.4 Reducing noise in **Falsehood(F)** using TVM.

6.5 Suppressing uncertainty in **Indeterminacy(I)** using Wavelet Shrinkage Denoising and further enhancing it with Multi-Scale Wavelet Sharpening

```python
import cv2
import numpy as np
from skimage.restoration import denoise_tv_chambolle, denoise_tv_bregman, denoise_wavelet
from skimage.filters import unsharp_mask
import pywt

def wavelet_sharpen(image, wavelet='haar', level=2, alpha=1.2):
    coeffs = pywt.wavedec2(image, wavelet, level=level)
    coeffs = list(coeffs)
    for i in range(1, len(coeffs)):
        coeffs[i] = tuple(alpha * c for c in coeffs[i])
    return pywt.waverec2(coeffs, wavelet)

def enhance_neutrosophic(T, F, I):
    # Enhance T using CLAHE
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    T_enhanced = clahe.apply((T * 255).astype(np.uint8)) / 255.0

    # Further sharpen T
    T_enhanced = unsharp_mask(T_enhanced, radius=1.5, amount=1.5)

    # Reduce noise in F using TVM
    F_enhanced = denoise_tv_chambolle(F, weight=0.1, channel_axis=None)

    # Suppress uncertainty in I using Wavelet Shrinkage Denoising
    I_enhanced = denoise_wavelet(I, method='BayesShrink', mode='soft', channel_axis=None, rescale_sigma=True)

    # Further enhance I with Multi-Scale Wavelet Sharpening
    I_enhanced = wavelet_sharpen(I_enhanced, wavelet='db2', level=2, alpha=1.3)

    return T_enhanced, F_enhanced, I_enhanced

# Apply enhancement to each channel's Neutrosophic components
T_R_enh, F_R_enh, I_R_enh = enhance_neutrosophic(T_R, F_R, I_R)
T_G_enh, F_G_enh, I_G_enh = enhance_neutrosophic(T_G, F_G, I_G)
T_B_enh, F_B_enh, I_B_enh = enhance_neutrosophic(T_B, F_B, I_B)

# Display the enhanced components
fig, ax = plt.subplots(3, 4, figsize=(16, 9))
components = ["Original", "Enhanced T", "Enhanced F", "Enhanced I"]
channels = [(R, T_R_enh, F_R_enh, I_R_enh), (G, T_G_enh, F_G_enh, I_G_enh), (B, T_B_enh, F_B_enh, I_B_enh)]
colors = ["Reds", "Greens", "Blues"]

for i in range(3):
    for j in range(4):
        ax[i, j].imshow(channels[i][j], cmap=colors[i] if j == 0 else "gray")
        ax[i, j].set_title(f"{['R', 'G', 'B'][i]} - {components[j]}")
        ax[i, j].axis("off")

plt.show()
```

Appendix 3: Reconstructing Neutrosophic transformations for T, F and I parts separately.

1. Enhancement of T Component:

```
def reconstruct_channel(T):
    """
    Reconstruct a single-channel image using enhanced T, F, and I.
    Ensures pixel values remain in the valid range [0, 1].
    """
    reconstructed = T
    reconstructed = np.clip(reconstructed, 0, 1)  # Ensure values remain valid
    return reconstructed

# Reconstruct R, G, B channels
R_reconstructed = reconstruct_channel(T_R_enh)
G_reconstructed = reconstruct_channel(T_G_enh)
B_reconstructed = reconstruct_channel(T_B_enh)

# Merge reconstructed channels into final image
final_image = np.stack([R_reconstructed, G_reconstructed, B_reconstructed], axis=-1)

# Display original vs reconstructed image
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax[0].imshow(image)
ax[0].set_title("Original Image")
ax[0].axis("off")

ax[1].imshow(final_image)
ax[1].set_title("Reconstructed Image (Neutrosophic)")
ax[1].axis("off")

plt.show()
```

2. Enhancement of F Component:

```
def reconstruct_channel(F):
    """
    Reconstruct a single-channel image using enhanced T, F, and I.
    Ensures pixel values remain in the valid range [0, 1].
    """
    reconstructed = F
    reconstructed = np.clip(reconstructed, 0, 1)  # Ensure values remain valid
    return reconstructed

# Reconstruct R, G, B channels
R_reconstructed = reconstruct_channel(F_R_enh)
G_reconstructed = reconstruct_channel(F_G_enh)
B_reconstructed = reconstruct_channel(F_B_enh)

# Merge reconstructed channels into final image
final_image = np.stack([R_reconstructed, G_reconstructed, B_reconstructed], axis=-1)

# Display original vs reconstructed image
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax[0].imshow(image)
ax[0].set_title("Original Image")
ax[0].axis("off")

ax[1].imshow(final_image)
ax[1].set_title("Reconstructed Image (Neutrosophic)")
ax[1].axis("off")

plt.show()
```

3. Enhancement of I Component:

```python
import numpy as np
import matplotlib.pyplot as plt

def reconstruct_channel(I):
    """
    Reconstruct a single-channel image using the Indeterminacy (I) component.
    Ensures pixel values remain in the valid range [0, 1] and enhances visibility.
    """
    # Normalize I to [0, 1] using min-max scaling
    I = I - I.min()  # Shift minimum to 0
    I = I / (I.max() + 1e-8)  # Normalize to [0,1] (avoid division by zero)

    return np.clip(I, 0, 1)

# Reconstruct R, G, B channels using normalized I components
R_reconstructed = reconstruct_channel(I_R_enh)
G_reconstructed = reconstruct_channel(I_G_enh)
B_reconstructed = reconstruct_channel(I_B_enh)

# Merge reconstructed channels into final image
final_image = np.stack([R_reconstructed, G_reconstructed, B_reconstructed], axis=-1)
'''
# Display original vs reconstructed image
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax[0].imshow(image)
ax[0].set_title("Original Image")
ax[0].axis("off")

ax[1].imshow(final_image)
ax[1].set_title("Reconstructed Image (Neutrosophic - Enhanced)")
ax[1].axis("off")

plt.show()
'''
# Convert the Neutrosophic outline to grayscale
gray_outline = cv2.cvtColor((final_image * 255).astype(np.uint8), cv2.COLOR_RGB2GRAY)

# Apply threshold to extract edges
_, binary_mask = cv2.threshold(gray_outline, 30, 255, cv2.THRESH_BINARY)

# Find contours from the binary mask
contours, _ = cv2.findContours(binary_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Create a mask for segmentation
mask = np.zeros_like(binary_mask)

# Draw filled contours on the mask
cv2.drawContours(mask, contours, -1, (255), thickness=cv2.FILLED)

# Apply mask to original image to extract segmented region
segmented_image = cv2.bitwise_and(image, image, mask=mask)

# Display original, outline, and segmented images
fig, ax = plt.subplots(1, 3, figsize=(18, 6))

ax[0].imshow(image)
ax[0].set_title("Original Image")
ax[0].axis("off")

ax[1].imshow(final_image)
ax[1].set_title("Neutrosophic Outline")
ax[1].axis("off")

ax[2].imshow(segmented_image)
ax[2].set_title("Segmented Content")
ax[2].axis("off")
```

Appendix 4: Preprocessing Neutrosophic enhanced images before feeding it to the Model

```python
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from tqdm.notebook import tqdm
import os

# Define the target size for resizing
target_size = (224, 224)

# Function to load and preprocess images
def load_images(class_names, class_paths):
    images = []
    labels = []

    for class_name in class_names:
        class_path = class_paths[class_name]
        # Use tqdm to show a progress bar
        for image_file in tqdm(os.listdir(class_path), desc=f"Processing {class_name} images"):
            # Load the image
            img_path = os.path.join(class_path, image_file)
            img = cv2.imread(img_path)

            if img is not None:
                # Resize the image
                img = cv2.resize(img, target_size)
                # Normalize pixel values to [0, 1] and convert to float32
                img = img.astype(np.float32) / 255.0

                # Append the image and label
                images.append(img)
                labels.append(class_name)

    # Convert lists to numpy arrays
    images = np.array(images, dtype=np.float32)
    labels = np.array(labels)

    return images, labels

# Load and preprocess images
images, labels = load_images(class_names, class_paths)

# Display the shape of the images and labels
print("Images shape:", images.shape)
print("Labels shape:", labels.shape)
```

```
Processing Early images: 100% 985/985 [00:04<00:00, 240.39it/s]
Processing Pro images: 100% 985/985 [00:01<00:00, 695.19it/s]
Processing Pre images: 100% 985/985 [00:01<00:00, 648.15it/s]
Processing Benign images: 100% 985/985 [00:02<00:00, 522.67it/s]
Images shape: (3940, 224, 224, 3)
Labels shape: (3940,)
```

Performing One-hot encoding

```
[ ]  from sklearn.preprocessing import LabelEncoder
     from tensorflow.keras.utils import to_categorical

     # Encode the labels
     label_encoder = LabelEncoder()
     labels_encoded = label_encoder.fit_transform(labels)

     # Convert to one-hot encoding
     labels_one_hot = to_categorical(labels_encoded)

     # Display the shape of the one-hot encoded labels
     print("One-hot encoded labels shape:", labels_one_hot.shape)
```

```
One-hot encoded labels shape: (3940, 4)
```

Source code for Dataset splits

```
# Split the dataset into training (80%), validation (10%), and test (10%) sets
X_train, X_temp, y_train, y_temp = train_test_split(images, labels_one_hot, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Display the shapes of the splits
print("Training set shape:", X_train.shape, y_train.shape)
print("Validation set shape:", X_val.shape, y_val.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (2758, 224, 224, 3) (2758, 4)
Validation set shape: (591, 224, 224, 3) (591, 4)
Test set shape: (591, 224, 224, 3) (591, 4)
```

Appendix 5: Model Summary

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 224, 224, 3)	0	-
Conv1 (Conv2D)	(None, 112, 112, 32)	864	input_layer[0][0]
bn_Conv1 (BatchNormalization)	(None, 112, 112, 32)	128	Conv1[0][0]
Conv1_relu (ReLU)	(None, 112, 112, 32)	0	bn_Conv1[0][0]
expanded_conv_depthwise (DepthwiseConv2D)	(None, 112, 112, 32)	288	Conv1_relu[0][0]
expanded_conv_depthwise_… (BatchNormalization)	(None, 112, 112, 32)	128	expanded_conv_depthwi…
expanded_conv_depthwise_… (ReLU)	(None, 112, 112, 32)	0	expanded_conv_depthwi…
expanded_conv_project (Conv2D)	(None, 112, 112, 16)	512	expanded_conv_depthwi…
expanded_conv_project_BN (BatchNormalization)	(None, 112, 112, 16)	64	expanded_conv_project…
block_1_expand (Conv2D)	(None, 112, 112, 96)	1,536	expanded_conv_project…
block_1_expand_BN (BatchNormalization)	(None, 112, 112, 96)	384	block_1_expand[0][0]
block_16_project (Conv2D)	(None, 7, 7, 320)	307,200	block_16_depthwise_re…
block_16_project_BN (BatchNormalization)	(None, 7, 7, 320)	1,280	block_16_project[0][0]
Conv_1 (Conv2D)	(None, 7, 7, 1280)	409,600	block_16_project_BN[0…
Conv_1_bn (BatchNormalization)	(None, 7, 7, 1280)	5,120	Conv_1[0][0]
out_relu (ReLU)	(None, 7, 7, 1280)	0	Conv_1_bn[0][0]
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0	out_relu[0][0]
dense (Dense)	(None, 128)	163,968	global_average_poolin…
dense_1 (Dense)	(None, 4)	516	dense[0][0]

Total params: 2,422,468 (9.24 MB)
Trainable params: 164,484 (642.52 KB)
Non-trainable params: 2,257,984 (8.61 MB)

Appendix 6: Source Code for printing the performance metrics values

```
from sklearn.metrics import classification_report

# Make predictions on the test set
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Convert one-hot encoded labels back to class labels
y_true = np.argmax(y_test, axis=1)

# Generate the classification report
print(classification_report(y_true, y_pred, target_names=label_encoder.classes_))
```