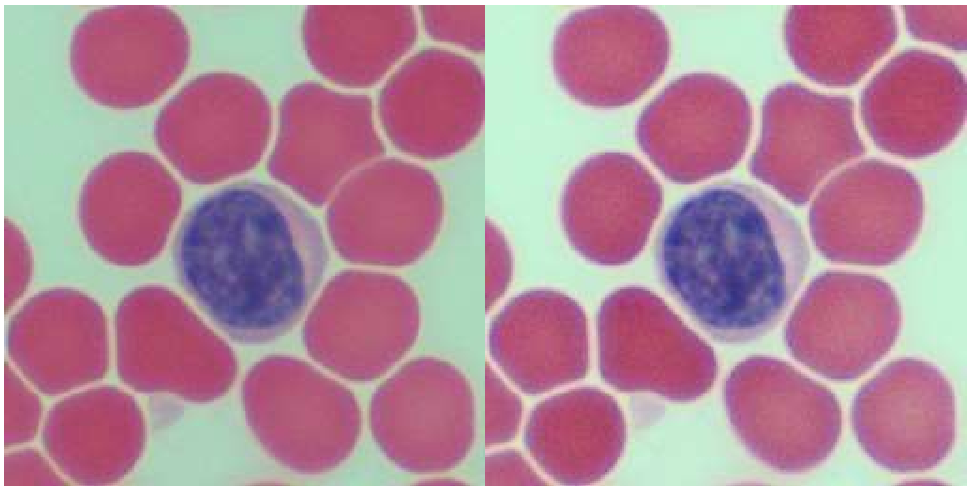
Blood cell Image Enhancement and Detection



GUIDED BY
PROF VISWANATHAN PERUMAL

Team Members:

Vishal Haswani	19BCI0181
S.P. Shreeyas	19BCE2478
Farhan Ansari	20BKT0077
Prikshit Sharma	19BCI0230

SCOPE

VIT UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)

Chapter		Page.No.
1	Abstract & Keywords	3
2	Objectives	3
3	Literature Review	4-5
4	Design -	6-11
4.1.	Proposed architecture	
4.2.	Proposed Image processing	
5	Invention Details	12
5.1.	Objective of the Invention	
5.2.	Detailed Description with functional block of the Invention	
5.3	Image Enhancement, Compression, Feature Extraction and recognition	
5.4.	Summary of Invention	
5.5.	Working Scenario	
5.5.1	Proposed Image Processing System	
5.5.2	Enhancement	
5.5.3	Compression	
5.5.4	Segmentation	
5.5.5	Feature Extraction	
5.5.6	Recognition	
6	Implementation - Snapshot of Prototype	13-23
6.1	Algorithm and Code	
7.	Results	24
8.	Conclusion	25
9	References	25

ABSTRACT

Domain of image processing is progressing a lot and has achieved tremendous milestones. Image processing is helping in many ways for the researchers to achieve their goals especially in security and medical fields. Detection of blood in disaster or remote areas where expert is unavailable is a challenge. In this paper we have proposed a system which will detect blood using image processing techniques. Steps to detect the blood cells from peripheral blood smears using image processing techniques are discussed.

OBJECTIVES

1. Enhancing our images to clearly identify the features for WBCs and RBCs.
2. Generate test images by rotating the images to increase the size of testing dataset.
3. To develop a method for detecting white and red blood cells from a given image of a peripheral blood smear on a microscopic slide.

LITERATURE REVIEW

In clinical laboratory counting of different types of blood cells is important for physicians to diagnose the diseases in particular patient. Manual microscopic inspection of blood cells is time consuming and requires more technical knowledge. Therefore, there is a need to research for an automated blood cell detection system that will help physicians to diagnose diseases in a fast and efficient way. Many researchers have done their research for counting blood cells using different methodologies.

The objective is to study these methodologies and identify future research direction in order to get more accurate results. A complete blood count is used to determine the state of a person's health based on the contents of the blood in particular white blood cells and the red blood cells. The project aims to provide a user-friendly software that allows quick user interaction with a simple tool for counting red and white blood cells from a provided image sample. Blood cell counting is nothing but the complete blood count (CBC) which refers to compilation test for all type of cells including red blood cells (RBC), white blood cells (WBC), platelets, haemoglobin and haematocrit.

Determination of blood type is essential before administering a blood transfusion, including in emergency situation. Currently, these tests are performed manually by technicians, which can lead to human errors. Various systems have been developed to automate these tests, but none is able to perform the analysis in time for emergency situations. One work aims to develop an automatic system to perform these tests in a short period of time, adapting to emergency situations. To do so, it uses the slide test and image processing techniques using the IMAQ Vision from National Instruments. The image captured after the slide test is processed and detects the occurrence of agglutination. Next the classification algorithm determines the blood type in analysis. Finally, all the information is stored in a database. Thus, the system allows determining the blood type in an emergency, eliminating transfusions based on the principle of universal donor and reducing transfusion reactions risks.

Along with the widespread use of the World Wide Web, violent contents have affected our daily lives. Although there are some investigations about violent video detection, few methods touch on the problem of violent and gory image detection. One such project proposes a region-based blood colour detection algorithm. Extracting colour and texture features from the detected bloody region of an image. Extracting features of the whole image according to the global and local method. These features have been fed into a Convolutional Neural Network to carry out the image classification.

These are some methods which apply image processing techniques to the field of medicine and other sciences.

DESIGN

Image Processing stages:

Image Acquisition Process

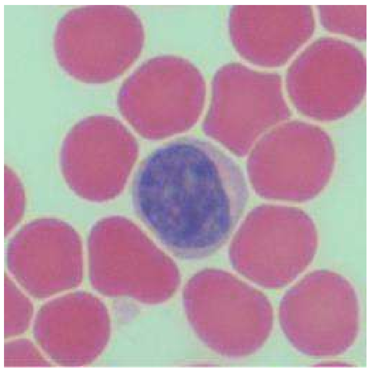
The datasets used are taken from Kaggle. Links to the dataset:

https://www.kaggle.com/draaslan/blood-cell-detection-dataset

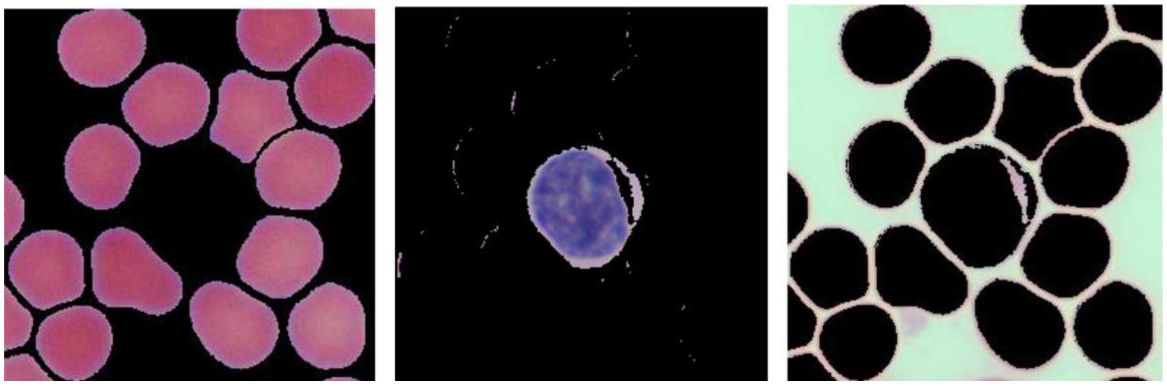
Image Enhancement Process

Lifecycle of the image

1) Raw Image is taken. Example-



2) Now a Min-Max Hybrid contrast stretching is used to enhance each RGB mask separately. The code for this enhancement function is at the end Example-



3) Combining the 3 masks to make a simple image. Example-

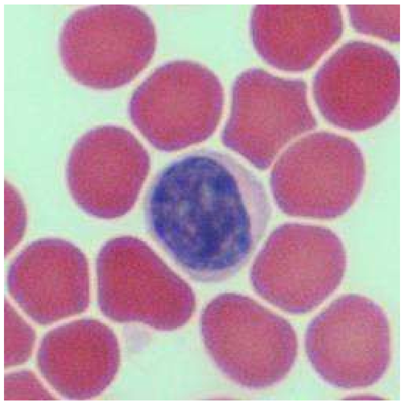
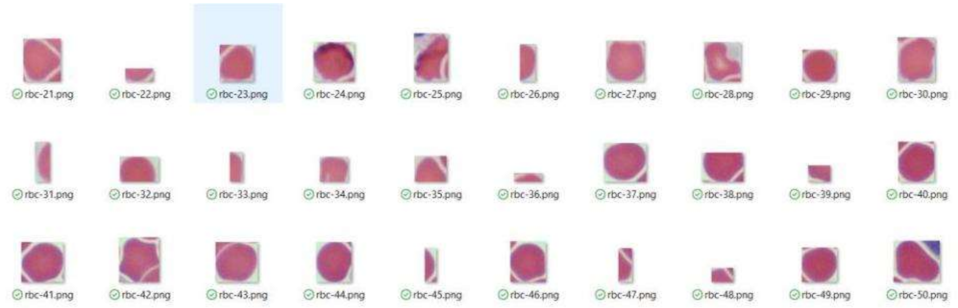
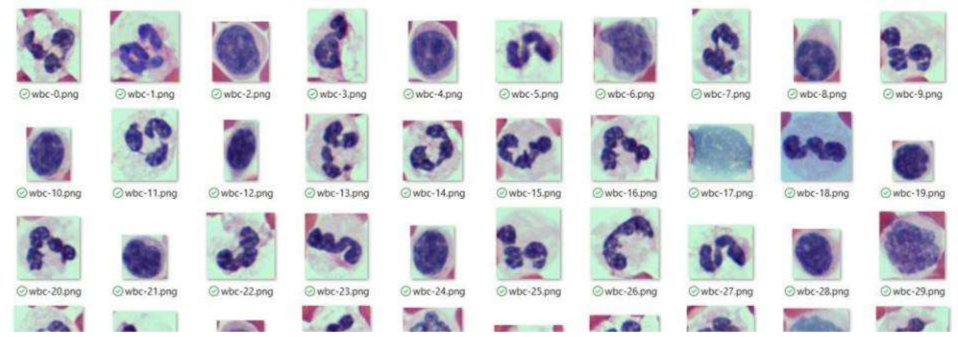


Image Feature Extraction

A python script will be used to cut the segments of the image with the white blood cell and red blood cells, i.e., the boxes shown in the below image will be cut to generate an exclusive RBC or WBC dataset to train the ML Model.



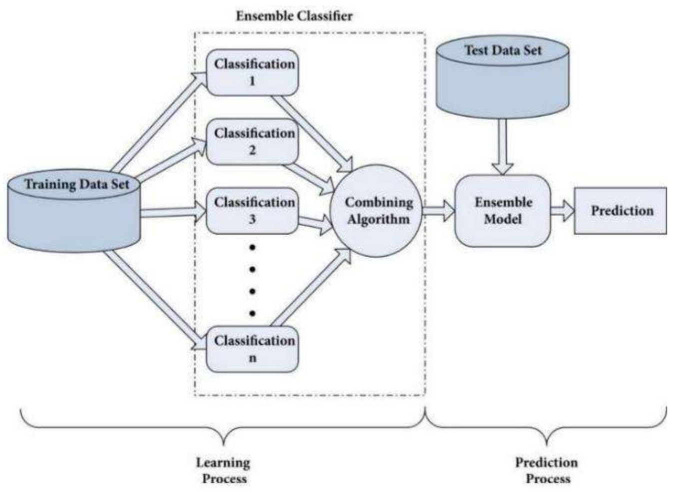
A total of 2236 images for RBCs are available for Model training.



A total of 100 images of WBCs are available for model training.

Image Classification

We will be using Converging Neural Network and Ensemble classifiers to get better accuracy.



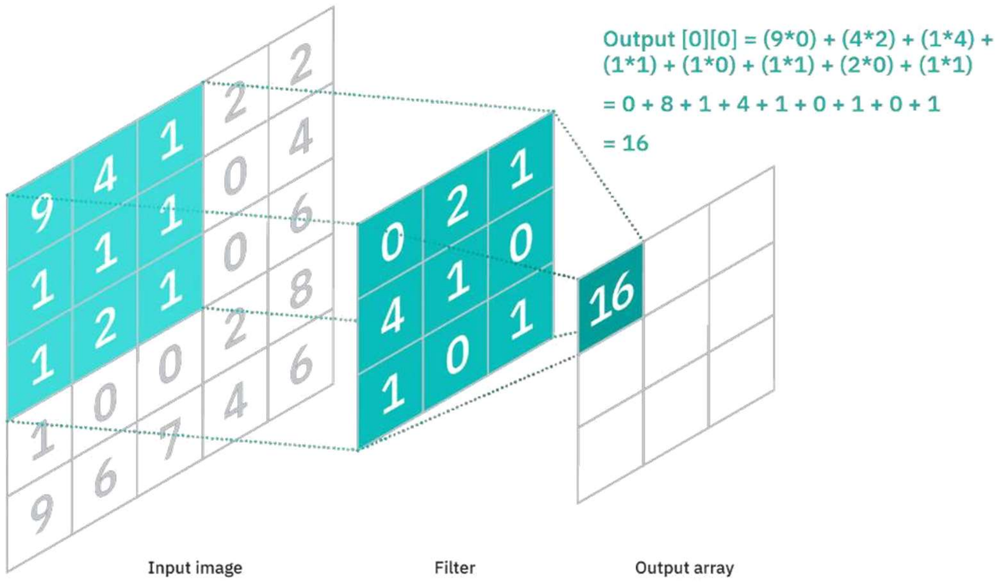
The Object classes in the Image are

0. Red Blood Cells and
1. White Blood Cells.

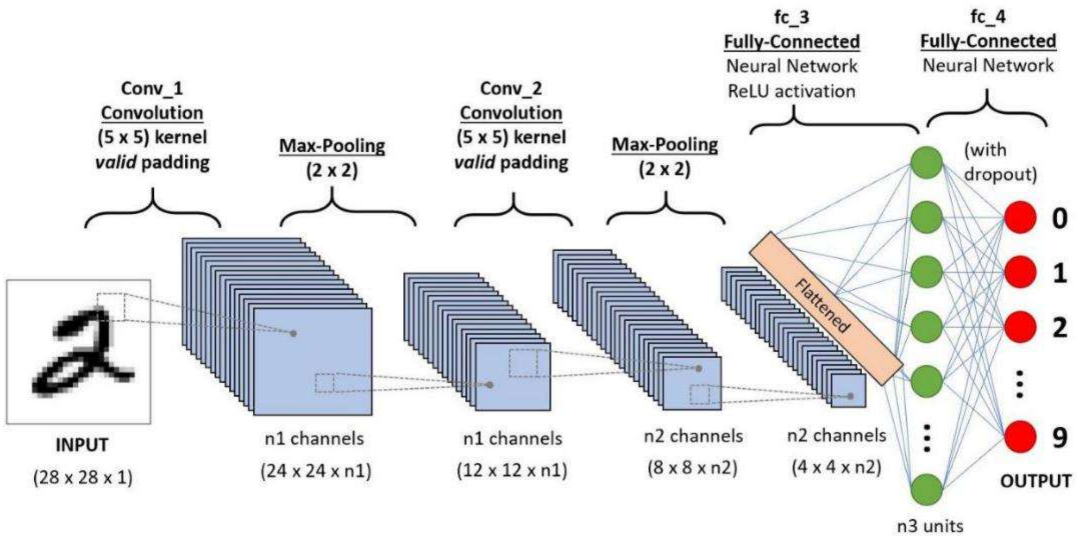
The ML Models chosen:

Convolutional Neural Network
CNN is a class of neural networks conventionally applied in image classification and recognition. This type of network works mainly on two layers: The convolution layer and the pooling layer.

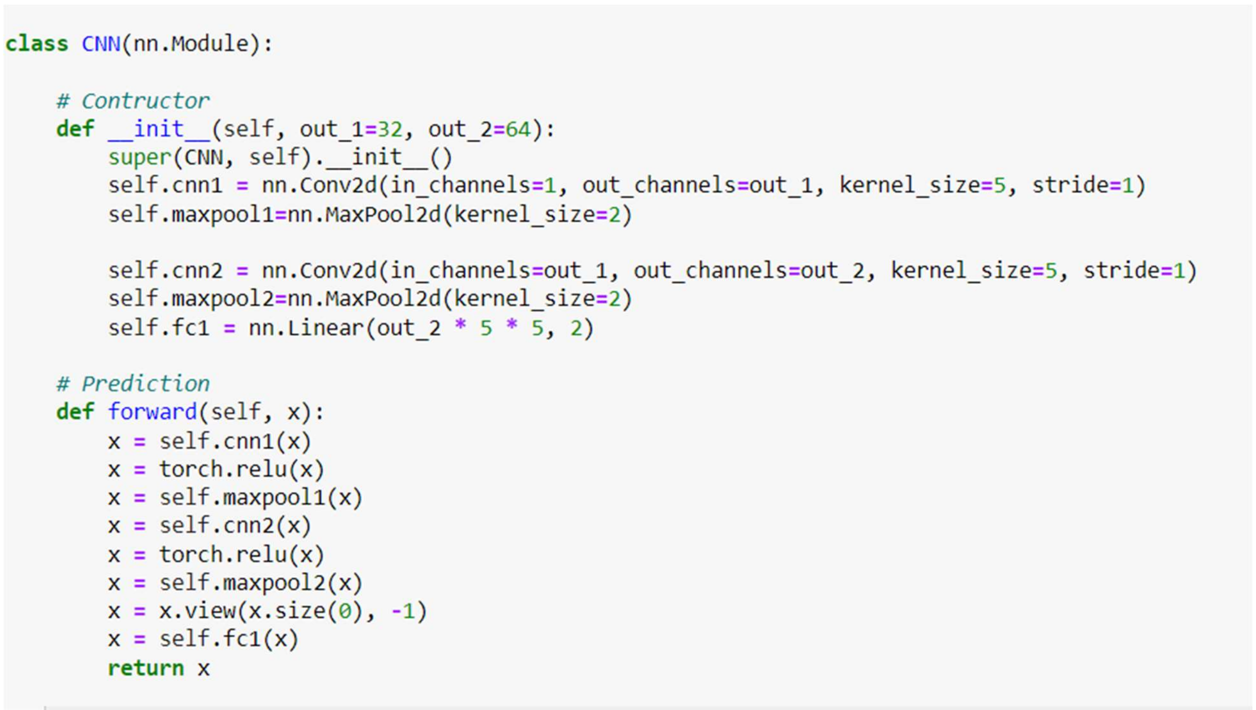
The convolution layer determines the features of the given image (say, a cat). This is done via applying a filter over the entire image which extracts certain numerical values over a 2-D matrix that describe the characteristics of the subject. Such a filter is called a kernel. Increasing the number of filters increases the depth of the model and helps identify more and more complex features. It is basically a dot product of the filter values and pixel values of the image over a specified region (typically, filters are 3x3 matrices). The output is called a "feature map".



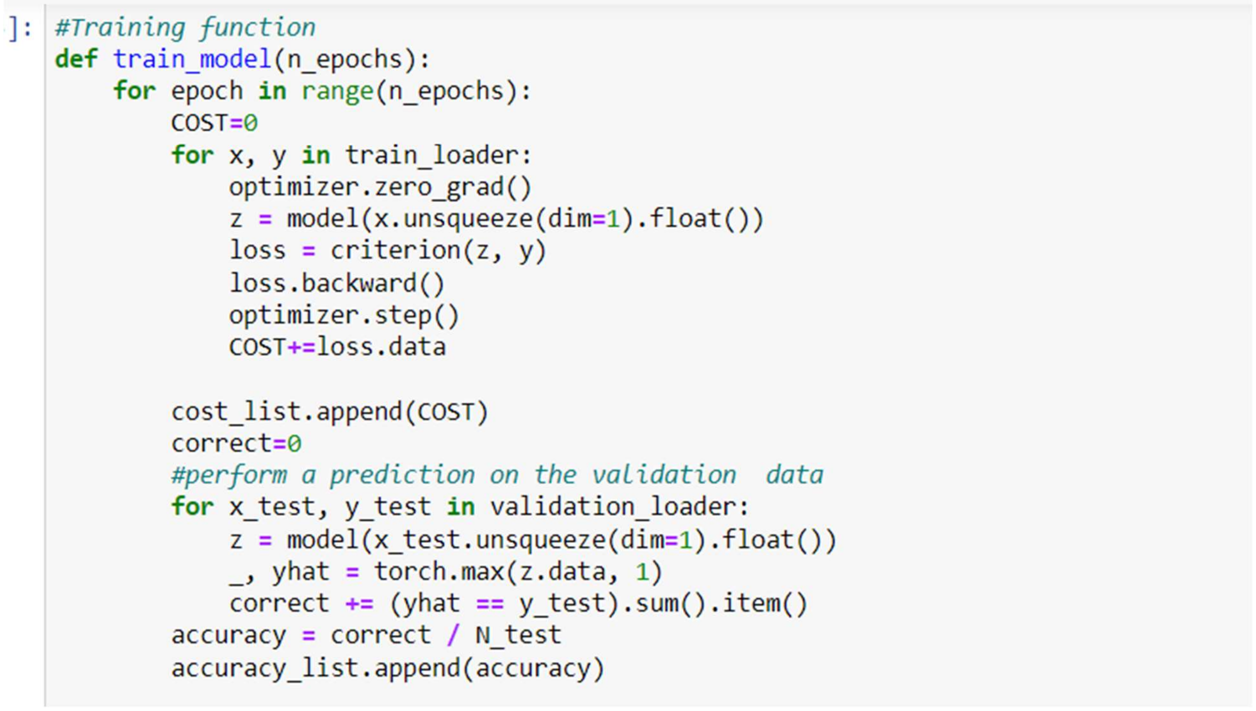
The pooling layer basically reduces the number of input features to bring out the more prominent ones, making computation easier. It is applied to the feature map. In our model, we prescribe using max pooling which extracts the maximum pixel value from the feature map. It carries forward the maximum amplitude of information onto the next layer of the network.



Pytorch snippets of CNN and its training:



```
class CNN(nn.Module):

    # Contructor
    def __init__(self, out_1=32, out_2=64):
        super(CNN, self).__init__()
        self.cnn1 = nn.Conv2d(in_channels=1, out_channels=out_1, kernel_size=5, stride=1)
        self.maxpool1=nn.MaxPool2d(kernel_size=2)

        self.cnn2 = nn.Conv2d(in_channels=out_1, out_channels=out_2, kernel_size=5, stride=1)
        self.maxpool2=nn.MaxPool2d(kernel_size=2)
        self.fc1 = nn.Linear(out_2 * 5 * 5, 2)

    # Prediction
    def forward(self, x):
        x = self.cnn1(x)
        x = torch.relu(x)
        x = self.maxpool1(x)
        x = self.cnn2(x)
        x = torch.relu(x)
        x = self.maxpool2(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        return x
```



```
]:  #Training function
    def train_model(n_epochs):
        for epoch in range(n_epochs):
            COST=0
            for x, y in train_loader:
                optimizer.zero_grad()
                z = model(x.unsqueeze(dim=1).float())
                loss = criterion(z, y)
                loss.backward()
                optimizer.step()
                COST+=loss.data

            cost_list.append(COST)
            correct=0
            #perform a prediction on the validation  data
            for x_test, y_test in validation_loader:
                z = model(x_test.unsqueeze(dim=1).float())
                _, yhat = torch.max(z.data, 1)
                correct += (yhat == y_test).sum().item()
            accuracy = correct / N_test
            accuracy_list.append(accuracy)
```

INVENTION DETAILS

The image enhancement techniques used are uniquely applied in our model covering the entire range of RGB filters while fitting contrast stretching methods to our sample images of RBCs and WBCs. The approach used was not to be found in any of the research materials we referred to for this project.

Comparing the results

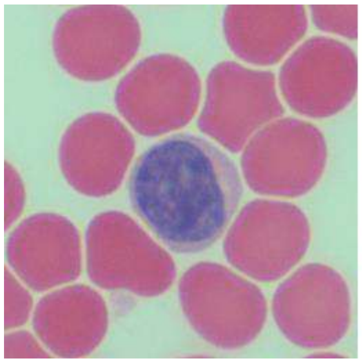
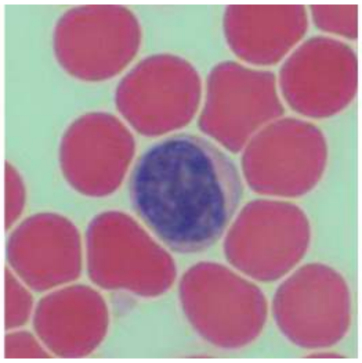
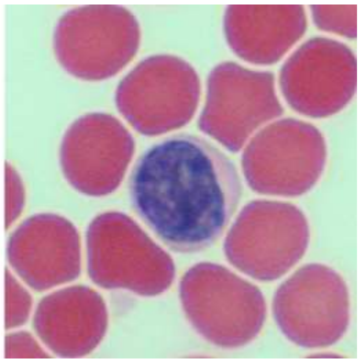




Fig-1 Fig-2 Fig-3

Fig-1: The raw data image
Fig-2: The Image simply enhanced without the segmentation method.
Fig-3: The image enhanced using the above-mentioned method.

Result: The Fig-3 has a better colour diversity and can be seen in the RBCs and WBC

For the Image Classification, these enhanced images were passed to a Convolutional Neural Network created using PyTorch in Python. More details on that can be found in the implementation section.

IMPLEMENTATION – SNAPSHOT OF PROTOTYPE

IMAGE ENHANCEMENT IMPLEMENTATION:

```
% The images in the dataset are labled from 1 to 120
% but the dataset only has 100 images
% the try catch part of the code filters out the missing images
% giving the set of available image numbers.
project_loc = "C:\Users\raj20\OneDrive\Desktop\IP\";
available_image_indexes = [];
for i = 1:120
    try
        imread(project_loc + "Dataset\images\image-"+ i +".png");
        available_image_indexes(end+1) = i;
    catch
        display("Image-" + i + " Not Found");

    end
end


% Extracting the RBC from the images to
% form a binary and coloured mask
for i = available_image_indexes
    img = imread(project_loc + "Dataset\images\image-"+ i +".png");
    [BinaryMask, RGBMask] = RBCMask(img);
    imwrite(BinaryMask, project_loc + "Processed Image\RBC Binary Mask\image-"+
i +".png");
    imwrite(RGBMask, project_loc + "Processed Image\RBC RGB Mask\image-"+ i
+".png");
end
% If we manually check the color mask images
% we can see that 5 images in total have very bad image masks


% Getting the mask for the remaining 5 images
for i = [31 79 80 81 86]
    img = imread(project_loc + "Dataset\images\image-"+ i +".png");
    [BinaryMask, RGBMask] = RBCMask2(img);
    imwrite(BinaryMask, project_loc + "Processed Image\RBC Binary Mask\image-"+
i +".png");
    imwrite(RGBMask, project_loc + "Processed Image\RBC RGB Mask\image-"+ i
+".png");
end
% Now the 5 bad masks have been over written
% by the new better ones


% Generating binary masks and RBG Masks for the WBCs
```

```
 for i = available_image_indexes
     try
         img = imread(project_loc + "Dataset\images\image-"+ i +".png");
         [WBCBinaryMask, RGBMask] = WBCMask(img);
         imwrite(WBCBinaryMask, project_loc + "Processed Image\WBC Binary
Mask\image-"+ i +".png");
         imwrite(RGBMask, project_loc + "Processed Image\WBC RGB Mask\image-"+ i
+".png");
     catch
         display("Problem at i = " + i);
     end
 end
 % again Those 5 Images are distorted
 % making the required WBC masks for the 5 different images

 for i = [31 79 80 81 86]
     try
         img = imread(project_loc + "Dataset\images\image-"+ i +".png");
         [WBC_RBCBinaryMask, RGBMask] = WBCMask2(img);
         imwrite(WBCBinaryMask, project_loc + "Processed Image\WBC Binary
Mask\image-"+ i +".png");
         imwrite(RGBMask, project_loc + "Processed Image\WBC RGB Mask\image-"+ i
+".png");
     catch
         display("Problem at i = " + i);
     end
 end
 % Now the 5 bad masks have been over written by the new better ones


 % Enhancing the RBC RGB Masks
 for i = available_image_indexes
     try
         img = imread(project_loc + "Processed Image\RBC RGB Mask\image-"+ i
+".png");
         img = EnhanceRGB(img, 2/3, 1.15);
         imwrite(img, project_loc + "RGB Enhanced Masks\RBC\image-"+ i +".png");
     catch
         display("Problem at i = " + i);
     end
 end


 % Enhancing the WBC RGB Masks
 for i = available_image_indexes
     try
         img = imread(project_loc + "Processed Image\WBC RGB Mask\image-"+ i
+".png");
         img = EnhanceRGB(img, 5/8, 1.15);
         imwrite(img, project_loc + "RGB Enhanced Masks\WBC\image-"+ i +".png");
```

```
        catch
            display("Problem at i = " + i);
        end
    end


    % Extracting and enhancing the background
    for i = available_image_indexes
        try

            img = imread(project_loc + "Dataset\images\image-"+ i +".png");
            RBCMask = imread(project_loc + "RGB Enhanced Masks\RBC\image-"+ i
+".png");
            WBCMask = imread(project_loc + "RGB Enhanced Masks\WBC\image-"+ i
+".png");
            RBC_WBCMask = logical(RBCMask) | logical(WBCMask);
            BackgroundMask = logical(img) - logical(RBC_WBCMask);
            img(~BackgroundMask) = 0;
            img = EnhanceRGB(img, 0.01, 1.1);
            imwrite(img, project_loc + "Processed Image\Background\image-"+ i
+".png");
        catch
            display("Problem at i = " + i);
        end
    end


    % Combining 3 masks
    for i = available_image_indexes
        RBCMask = imread(project_loc + "RGB Enhanced Masks\RBC\image-"+ i +".png");
        WBCMask = imread(project_loc + "RGB Enhanced Masks\WBC\image-"+ i +".png");
        BgMask = imread(project_loc + "Processed Image\Background\image-"+ i
+".png");
        for l3 = 1:3
            for l2 =  1:256
                for l1 = 1:256
                    if BgMask(l1, l2, l3) == 0
                        if WBCMask(l1, l2, l3) == 0
                            BgMask(l1, l2, l3) = RBCMask(l1, l2, l3);
                        else
                            BgMask(l1, l2, l3) = WBCMask(l1, l2, l3);
                        end
                    end
                end
            end
        end
    imwrite(BgMask, project_loc + "RGB Enhanced Masks\Final\image-"+ i
+".p
 ng"
 );
 end


 % Code for hiding the
 distortionsdim = 3;
 filter = ones(dim)/(dim^2);
 for i = available_image_indexes
     img = imread(project_loc + "RGB Enhanced Masks\Final\image-"+ i
     +".png");

     BgMaskAVG = imfilter(BgMask, filter, "symmetric");
     imwrite(img, project_loc + "Final Dataset\Images\image-"+ i +".png");
 end


 % Generating a without segmentation
 % dataset for comparing the
 resultfor i =
 available_image_indexes
     img = imread(project_loc + "Dataset\images\image-"+ i
     +".png");img = EnhanceRGB(img, 0.4, 0.98);
     imwrite(img, project_loc + "Final Dataset without Segmentation\image-"+
     i
+".p
 ng"
 );
 end
```

Comparing the results

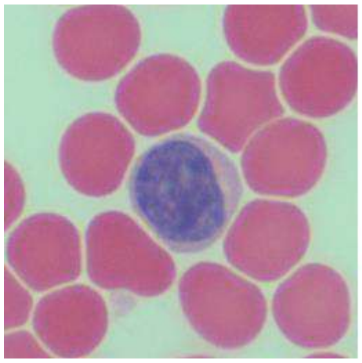
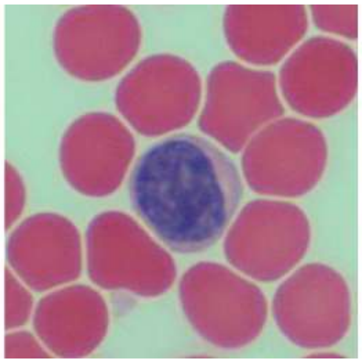
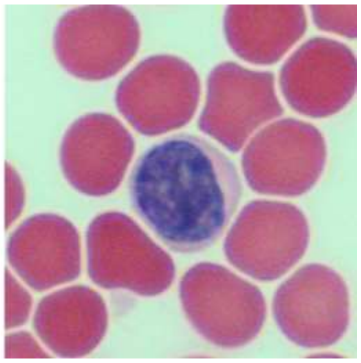




Fig-1 Fig-2 Fig-3

Fig-1: The raw data image
Fig-2: The Image simply enhanced without the segmentation method.
Fig-3: The image enhanced using the above-mentioned method.

<u>Result:</u> The Fig-3 has a better colour diversity and can be seen in the RBCs and WBC

IMAGE CLASSIFICATION USING CNN:

```
import torch
import torch.nn as nn
import matplotlib.pylab as plt
import numpy as np
import os
import cv2
import random
import pickle

#Creating the dataset
DATADIR = "C:\\Users\\Farhan\\Pictures\\Images 2"
CATEGORIES = ['RBC', 'WBC']
IMAGE_SIZE = 32
data = []

class my_set():
        def create_train():
                for category in CATEGORIES:
                    path = os.path.join(DATADIR, category)
                    class_num = CATEGORIES.index(category)
                    for img in os.listdir(path):
                        try:
                          img_array = cv2.imread(os.path.join(path, img),
cv2.IMREAD_GRAYSCALE)
                          new_array = cv2.resize(img_array, (IMAGE_SIZE,
IMAGE_SIZE))
                          data.append([new_array, class_num])
                        except:
                            pass

        create_train()
        random.shuffle(data)

        x = []
        y = []

        for features, label in data:
            x.append(features)
            y.append(label)

        x = np.array(x).reshape(-1, IMAGE_SIZE, IMAGE_SIZE, 1)

        pickle_out = open("x.pickle", "wb")
        pickle.dump(x, pickle_out)
        pickle_out.close()

        pickle_out = open("y.pickle", "wb")
        pickle.dump(y, pickle_out)
        pickle_out.close()

print("Length of dataset:", len(data))
```

```
Length of dataset: 2336
```

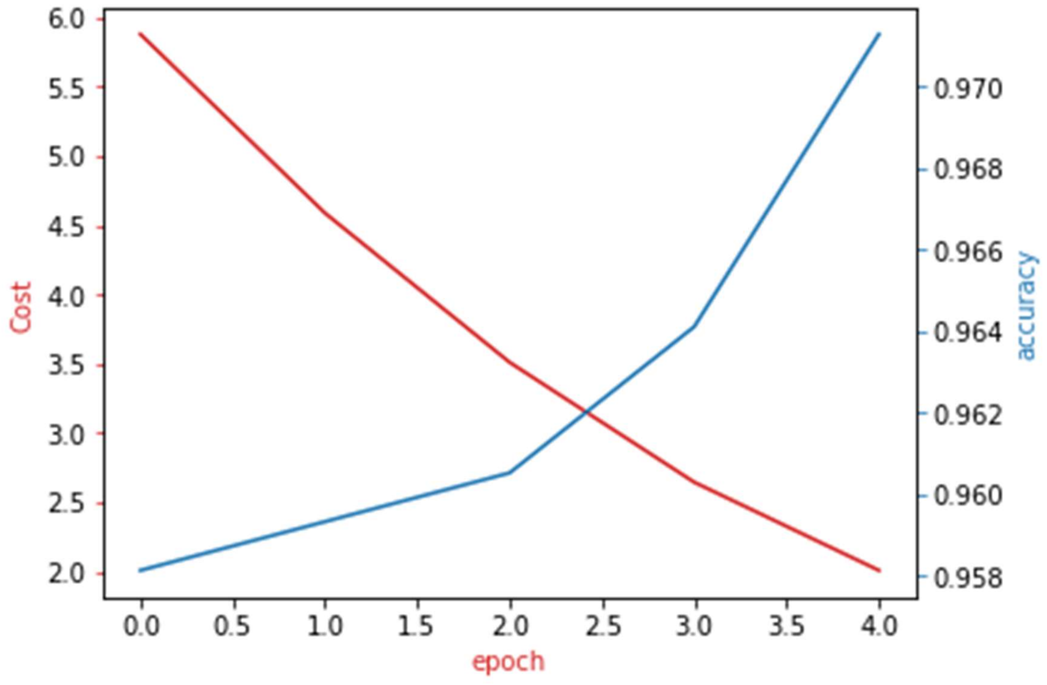
```
#Function to plot the image and model class (CNN)
def show_data(data_sample):
  plt.imshow(data_sample[0].reshape(32, 32), cmap='gray')
  plt.title('y = '+ str(data_sample[1]))


class CNN(nn.Module):

  # Contructor
  def __init__(self, out_1=32, out_2=64):
    super(CNN, self).__init__()
    self.cnn1 = nn.Conv2d(in_channels=1, out_channels=out_1,
kernel_size=5, stride=1)
    self.maxpool1=nn.MaxPool2d(kernel_size=2)

    self.cnn2 = nn.Conv2d(in_channels=out_1, out_channels=out_2,
kernel_size=5, stride=1)
    self.maxpool2=nn.MaxPool2d(kernel_size=2)
    self.fc1 = nn.Linear(out_2 * 5 * 5, 2)

  # Prediction
  def forward(self, x):
    x = self.cnn1(x)
```

```
        x = torch.relu(x)
        x = self.maxpool1(x)
        x = self.cnn2(x)
        x = torch.relu(x)
        x = self.maxpool2(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        return x

#Creating model instance and other parameters for training
model = CNN()
criterion = nn.CrossEntropyLoss()
learning_rate = 0.00001
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate,
weight_decay=1e-5)
train_set, val_set =  torch.utils.data.random_split(data, [1500, 836])
train_loader = torch.utils.data.DataLoader(dataset = train_set, batch_size
= 100)
validation_loader = torch.utils.data.DataLoader(dataset = val_set,
batch_size=1)
n_epochs = 5
cost_list = []
accuracy_list = []
N_test = len(val_set)
COST = 0
print("Length of training set: ", len(train_set))
print("Length of validation set: ", len(val_set))
```

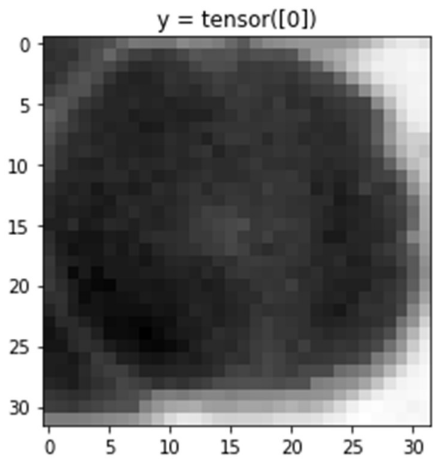
```
Length of training set:  1500
Length of validation set:  836
```

```
#Training function
def train_model(n_epochs):
    for epoch in range(n_epochs):
        COST=0
        for x, y in train_loader:
            optimizer.zero_grad()
            z = model(x.unsqueeze(dim=1).float())
            loss = criterion(z, y)
            loss.backward()
            optimizer.step()
            COST+=loss.data

        cost_list.append(COST)
        correct=0
        #perform a prediction on the validation  data
        for x_test, y_test in validation_loader:
            z = model(x_test.unsqueeze(dim=1).float())
            _, yhat = torch.max(z.data, 1)
            correct += (yhat == y_test).sum().item()
        accuracy = correct / N_test
        accuracy_list.append(accuracy)

    fig, ax1 = plt.subplots()
    color = 'tab:red'
    ax1.plot(cost_list, color=color)
    ax1.set_xlabel('epoch', color=color)
    ax1.set_ylabel('Cost', color=color)
    ax1.tick_params(axis='y', color=color)

    ax2 = ax1.twinx()
    color = 'tab:blue'
    ax2.set_ylabel('accuracy', color=color)
    ax2.set_xlabel('epoch', color=color)
    ax2.plot(accuracy_list, color=color)
    ax2.tick_params(axis='y', color=color)
    fig.tight_layout()
    print("Validation Accuracy: ", accuracy*100, "%")

train_model(n_epochs)
```

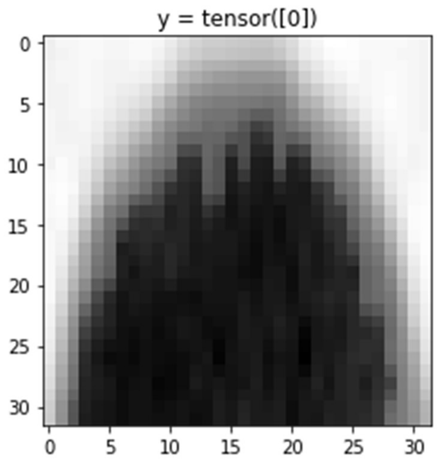
```
Validation Accuracy:  97.1291866028708 %
```



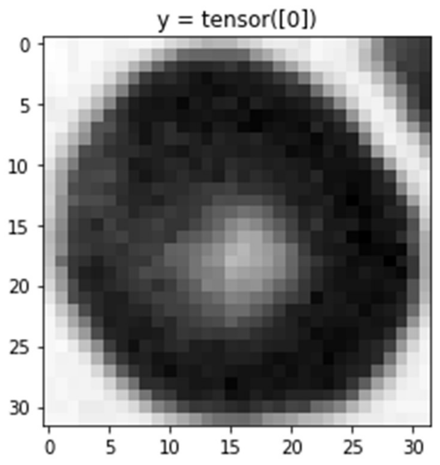
```
#Plotting sample pictures with the outputs
correct=0
count=0
for x_test, y_test in validation_loader:
        z = model(x_test.unsqueeze(dim=1).float())
        _, yhat = torch.max(z.data, 1)
        correct += (yhat == y_test).sum().item()
        if count < 5:
            show_data((x_test, y_test))
            plt.show()
            print("yhat:", yhat)
            count+=1
```



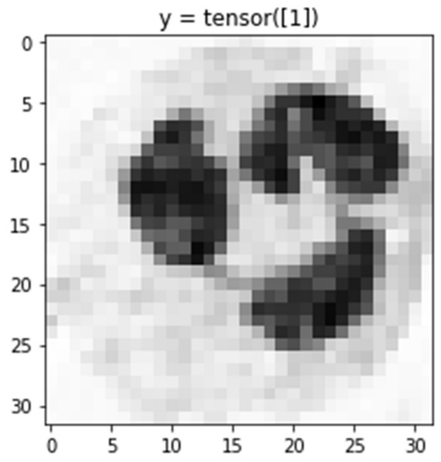
```
yhat: tensor([0])
```



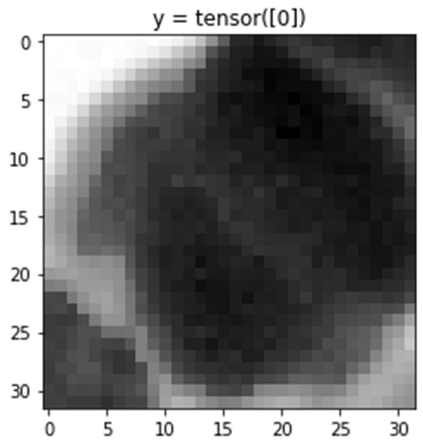
```
yhat: tensor([0])
```



```
yhat: tensor([0])
```



```
yhat: tensor([1])
```



```
yhat: tensor([0])
```

RESULTS

We successfully applied image processing techniques to our respective dataset and carried out the classification part using various architectures and methods, mainly using Python and MATLAB.

Image Enhancement was achieved through our proposed methods and inventions:

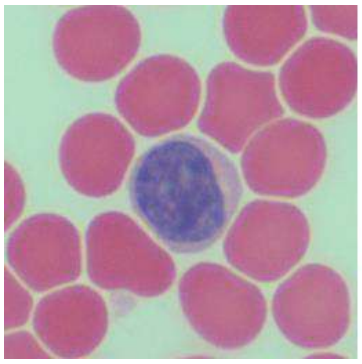
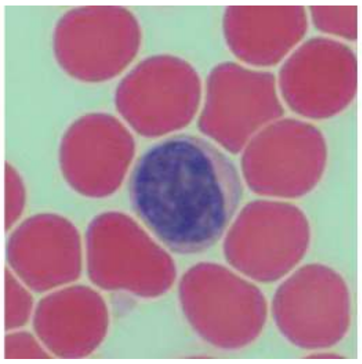
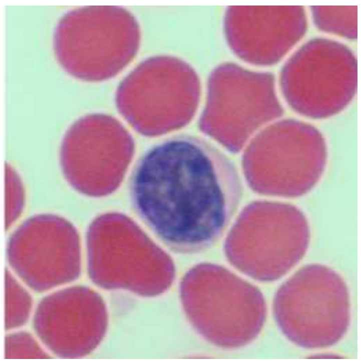




Fig-1 Fig-2 Fig-3

Fig-1: The raw data image

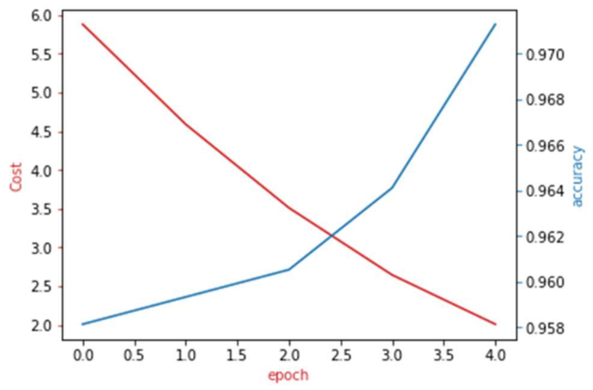
Fig-2: The Image simply enhanced without the segmentation method.

Fig-3: The image enhanced using the above-mentioned method.

Result: The Fig-3 has a better colour diversity and can be seen in the RBCs and WBC

The classification model yielded a satisfactory accuracy after training:

```
Validation Accuracy:  97.1291866028708 %
```



CONCLUSION

The methods used in our project have proven to be effective in not only classifying the images but also enhancing them to further accelerate the process of said classification. The new and innovative approach to classification has been carried out with success and marvel.

REFERENCES

https://www.ijert.org/research/automatic-blood-cell-analysis-by-using-digital-image-processing-a-preliminary-study-IJERTV2IS90964.pdf

https://www.intechopen.com/chapters/58322

https://www.kaggle.com/kushal1996/detecting-malaria-cnn