1. INTRODUCTION

White blood cell, also called leukocyte or white corpuscle, a cellular component of the blood that lacks haemoglobin, has a nucleus, is capable of motility, and defends the body against infection and disease by ingesting foreign materials and cellular debris, by destroying infectious agents and cancer cells, or by producing antibodies. WBC is made in the bone marrow and found in the blood and lymph tissue. White blood cells are part of the body's immune system. They help the body fight infection and other diseases. Blood smear analysis done by a pathologist using microscope is used to observe WBC cells. This analysis help doctor to diagnose a patient but it can cause error. That is why, with the advancement of deep learning, various object detection techniques have become useful for automating the process and reducing human errors in blood smear analysis.

In the last few years, deep learning has increasingly shown the potential to improve healthcare by aiding medical professionals with diagnostic processes and patient interactions. In particular, Convolutional Neural Networks (CNNs), a class of deep learning algorithms, have successfully been applied to classify images of biological features that are often used to help monitor overall health and detect disorders in patients. In this project Convolutional Neural Network based ResNet architecture is used. ResNet is one of the most powerful deep neural networks. The classic ResNet architecture is ResNet50 i.e., ResNet with 50 layers, but here we use a customized ResNet with 24 layers which is similar to ResNet50. ResNet uses Batch Normalization. The Batch Normalization adjusts the input layer to increase the performance of the network. ResNet makes use of the Identity Connection, which helps to protect the network from vanishing gradient problem. Deep Residual Network uses bottleneck residual block design to increase the performance of the network. In ResNet architecture, a "shortcut" or a "skip connection" allows the gradient to be directly backpropagated to earlier layers. Here the ResNet identify the different types of WBC cells in the blood smear image.

The dataset used in this project to train and evaluate the model is blood cell images from Kaggle. The dataset contains 5106 images in total in which 2548 for training, 2487 images for testing and 71 for validation. White blood cells are an important part of our immune system. Different types of white blood cells perform different functions in the body. Overall, white blood cells help to protect us against bacteria, viruses, and parasites. Mainly

there are 5 types of WBC cells. They are Neutrophils, Lymphocytes, Eosinophils, Basophils and Monocytes. But here in this project we are detecting only 4 of them: eosinophil, lymphocyte, monocyte and neutrophil. By detecting the type of WBC, this model helps doctors to diagnose a patient easily. The normal range (total) of white blood cells is between 4,000 and 10,000 cells per microliter (mcL). On its own, a low WBC count doesn't have symptoms. But a low count will often lead to an infection, because not enough white cells are present to fight off the invader. A high white blood count (WBC) can be a symptom of an underlying disorder. Disorders that are related to a high WBC include autoimmune or inflammatory disease, bacterial or viral infection, leukaemia, Hodgkin's disease, or allergic reaction.

2. SUPPORTING LITERATURE

2.1. Literature Review

Paper 1: -Vatathanavaro, Supawit, Suchat Tungjitnob, and Kitsuchart Pasupa. "White blood cell classification: a comparison between VGG-16 and ResNet-50 models." proceeding of the 6th joint symposium on computational intelligence (JSCI6). Vol. 12. 2018.

There are two main methods for WBC classification task. The first one is to use automated blood analysers. Although this method can achieve very high accuracy, its cost and maintenance are incredibly high. The second method is human labour. Due to a similarity between the appearance of various types of WBC, it is very difficult to classify by the human eye. Many researchers applied several algorithms to WBC classification task, i.e., Support Vector Machine, Random Forest, Convolutional Neural Network (CNN). It is found that CNN is the best algorithm. Many deep learning architectures have been introduced to suit with different types of application, such as VGG-16 and ResNet-50. There are studies on deep learning with WBC but, it is difficult to compare these algorithms together due to different experiment settings and datasets. Thus, here we compare both architectures on a well-designed experimental framework on a combination of two different datasets in this work. The optimal number of epochs for the VGG-16 and the ResNet-50 is 44 and 47 is selected to train here. The average validation accuracy from training and test is greater for ResNet50 than VGG16.

Paper 2: - He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Deeper neural networks are more difficult to train. This paper presents a residual learning framework to ease the training of networks that are substantially deeper than those used previously. In deep neural networks adding more layers leads to the model's accuracy saturating, then rapidly decaying, and higher training errors. This paper proposes that instead of hoping every few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping. The identity shortcut can be directly used when the input and output are of the same dimension. when dimensions increase: (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimension; (B). The

projection shortcut is used to match dimensions (done by 1x1 convolutions). The author's hypothesis is that it is easy to optimize the residual mapping function than to optimize the original, unreferenced mapping. In this research the plain and residual net CNN of 18 and 34 layers were trained. In plain CNN the error increases for plain networks when depth is increased from 18 to 34 layer while it reduces in ResNet with an increase in depth and the error was lower compared to plain nets. It led to major improvements in Image recognition, image detection, image localization, and segmentation tasks. It contributed heavily to industrial applications such as medical image analysis, satellite image processing, and robot vision. This architecture has more layers but lesser parameters than previous models hence making it fast.

Paper 3: - Habibzadeh, Mehdi, et al. "Automatic white blood cell classification using pre-trained deep learning models: Resnet and inception." Tenth international conference on machine vision (ICMV 2017). Vol. 10696. SPIE, 2018.

The examination of peripheral thin blood smears plays the main role of hematologic diagnosis. Blood cells are categorized as Red Blood Cells (Erythrocytes), White Blood Cells (Leukocytes), and platelets. The main responsibility of leukocytes is to defend of the body organs using phagocytic activity mechanism to remove cell debris and damage in biological structures. There are five normal WBCs mature types (with typical percentage of occurrence in normal blood): Basophil (≤ 1%); Eosinophil (≤ 5%); Monocyte (3−9%); Lymphocyte (25−35%); and Neutrophil (40−75%). In this research, various Inception and ResNet deep learning classification are presented and the use of these theories is outlined. The best results achieved where by fine tuning all layers and ResNet groups settings. In future work ResNet and Inception combination with powerful distributed TensorFlow to train a huge number of parameters will be addressed. Briefly, the empirical findings in this study provide a better understanding of hierarchical deep feature extraction process. One of the more significant findlings to emerge from this study is that the possibility of extending the use of ResNet to entire field of pathological analysis or other similar medical research.

2.2. Findings and Proposals

From paper 1 we can conclude that ResNet50 is expected to be 88% accurate. The results show that ResNet-50 model can outperform VGG16 model in this task. Network depth is a crucially important factor and the levels of features can be enriched due to the number of stacked layers. If the depth of the VGG-16 model is increased, this can cause gradient vanishing problem that leads to higher training error. On the other hand, the ResNet-50 model is deeper than the VGG16 model, but it has an identity function that can preserve the gradient resulting in a more accurate model. The main contribution of paper 2 is residual learning eases optimization, solved the degradation problem, faster training of deep neural networks, decreases the error rate for deeper networks. It led to major improvements in Image recognition, image detection, image localization, and segmentation tasks. It contributed heavily to industrial applications such as medical image analysis, satellite image processing, and robot vision. This architecture has more layers but lesser parameters than previous models hence making it fast. The core idea of ResNet is that it introduced a so-called "identity shortcut connection" that skips one or more layers. Due to this skip connection, the output of the layer is not the same now. Without using this skip connection, the input gets multiplied by the weights of the layer followed by adding a bias term. The skip connections in ResNet solve the problem of vanishing gradient in deep neural networks by allowing this alternate shortcut path for the gradient to flow through. The third paper shows the implementation of ResNet50 and Inception together for detecting WBC. This paper shows that reconstructing or customizing of ResNet is possible. From the three papers I have read, it is clear that WBC detection is more accurate by using ResNet. The deeper ResNet (ResNet50) is more compatible but it contains 50 layers. ResNet50 can have test accuracy upto 88% for a very large dataset according to paper 1. In the third paper it shows that customization of ResNet can be compatible in future.

3. SYSTEM ANALYSIS

3.1. Analysis of Dataset

3.1.1. About the Dataset

I collect my dataset from

❖ https://www.kaggle.com/code/yvtsanlevy/96-accuracies-blood-cell-recognistion-with-resnet/data?select=blood_cell_images.zip

The dataset used here is called blood cell images. This dataset contains 5106 images of blood cells with accompanying cell type labels (CSV). The classes are Eosinophil, Lymphocyte, Monocyte, and Neutrophil. The dataset contains only these 4 WBC subtypes. White blood cells, also called leukocytes or leucocytes, are the cells of the immune system that are involved in protecting the body against both infectious disease and foreign invaders. All white blood cells are produced and derived from multipotent cells in the bone marrow known as hematopoietic stem cells. All white blood cells have nuclei, which distinguishes them from the other blood cells.

The main types of WBC present here are neutrophils, eosinophils, lymphocytes, and monocytes. Neutrophils are the most abundant white blood cell, constituting 60-70% of the circulating WBC. They defend against bacterial or fungal infection. Neutrophils are the most common cell type seen in the early stages of acute inflammation. The average lifespan of inactivated human neutrophils in the circulation has been reported by different approaches to be between 5 and 135 hours. When adhered to a surface, neutrophil granulocytes have an average diameter of 12–15 micrometers (μm) in peripheral blood smears. In suspension, human neutrophils have an average diameter of 8.85 μm. The nucleus has a characteristic lobed appearance, the separate lobes connected by chromatin. The cytoplasm also contains about 200 granules.

Eosinophils compose about 2-4% of white blood cells in circulating blood. This count fluctuates throughout the day, seasonally, and during menstruation. It rises in response to allergies, parasitic infections, collagen diseases, and disease of the spleen and central nervous system. They are rare in the blood, but numerous in the mucous membranes of the respiratory, digestive, and lower urinary tracts. They primarily deal with parasitic infections. Eosinophils

are also the predominant inflammatory cells in allergic reactions. The most important causes of eosinophilia include allergies such as asthma, hay fever, and hives; and parasitic infections. They secrete chemicals that destroy large parasites, such as hookworms and tapeworms. their nuclei are bi-lobed. The lobes are connected by a thin strand. The cytoplasm is full of granules that assume a characteristic pink-orange colour with eosin staining.

Lymphocytes are much more common in the lymphatic system than in blood. Lymphocytes are distinguished by having a deeply staining nucleus that may be eccentric in location, and a relatively small amount of cytoplasm. A general increase in the number of lymphocytes is known as lymphocytosis, whereas a decrease is known as lymphocytopenia. An increase in lymphocyte concentration is usually a sign of a viral infection. A low normal to low absolute lymphocyte concentration is associated with increased rates of infection after surgery or trauma. Monocytes are the largest type of leukocyte in blood. Monocytes are amoeboid in appearance, and have nongranulated cytoplasm. With a diameter of 15–22 μm, monocytes are the largest cell type in peripheral blood. Monocytes are mononuclear cells and the ellipsoidal nucleus is often lobulated/indented, causing a bean-shaped or kidney-shaped appearance. Monocytes compose 2% to 10% of all leukocytes in the human body. The normal total leucocyte count in an adult is 4000 to 11,000 per mm3 of blood.

3.1.2. Explore the Dataset

Blood cell images dataset contains 5106 of smear images with 4 classes eosinophil, lymphocyte, monocyte and neutrophil. There are 2548 images for training, 2487 images for testing and 71 for validation. In train image set 2497 are eosinophil, 2483 are lymphocyte, 2478 are monocyte and 2499 are neutrophil. In test image set 623 images are eosinophil, 620 are lymphocyte, 620 are monocyte and 624 are neutrophil. Lastly in validation set 13 images are eosinophil, 6 are lymphocyte, 4 are monocyte and 48 are neutrophil.

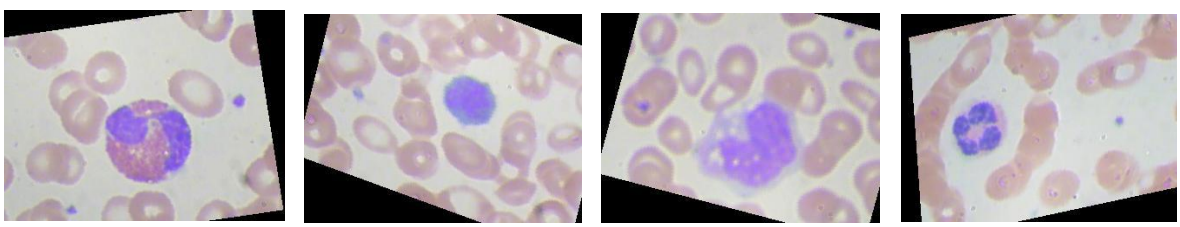

Fig 3.1: Sample images from each class.

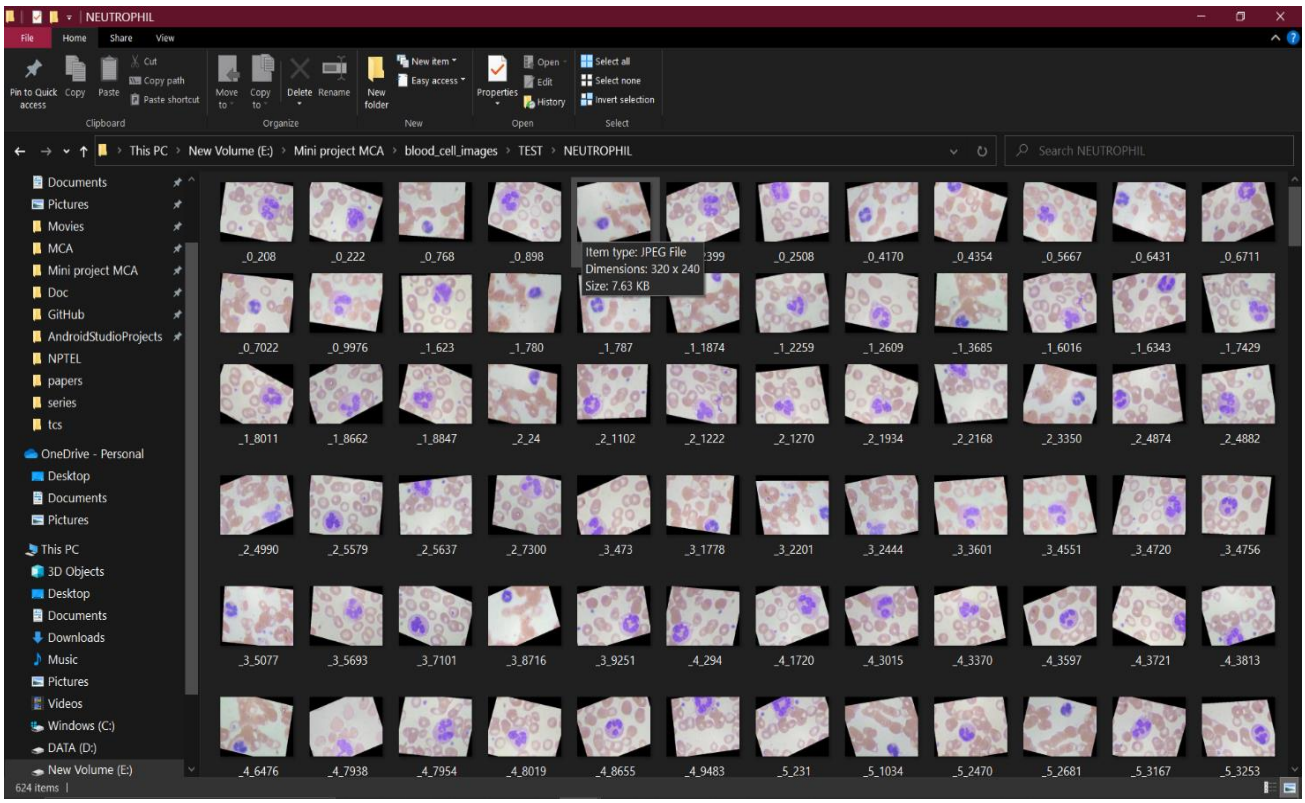

Fig 3.2: Sample from neutrophil folder of Dataset.

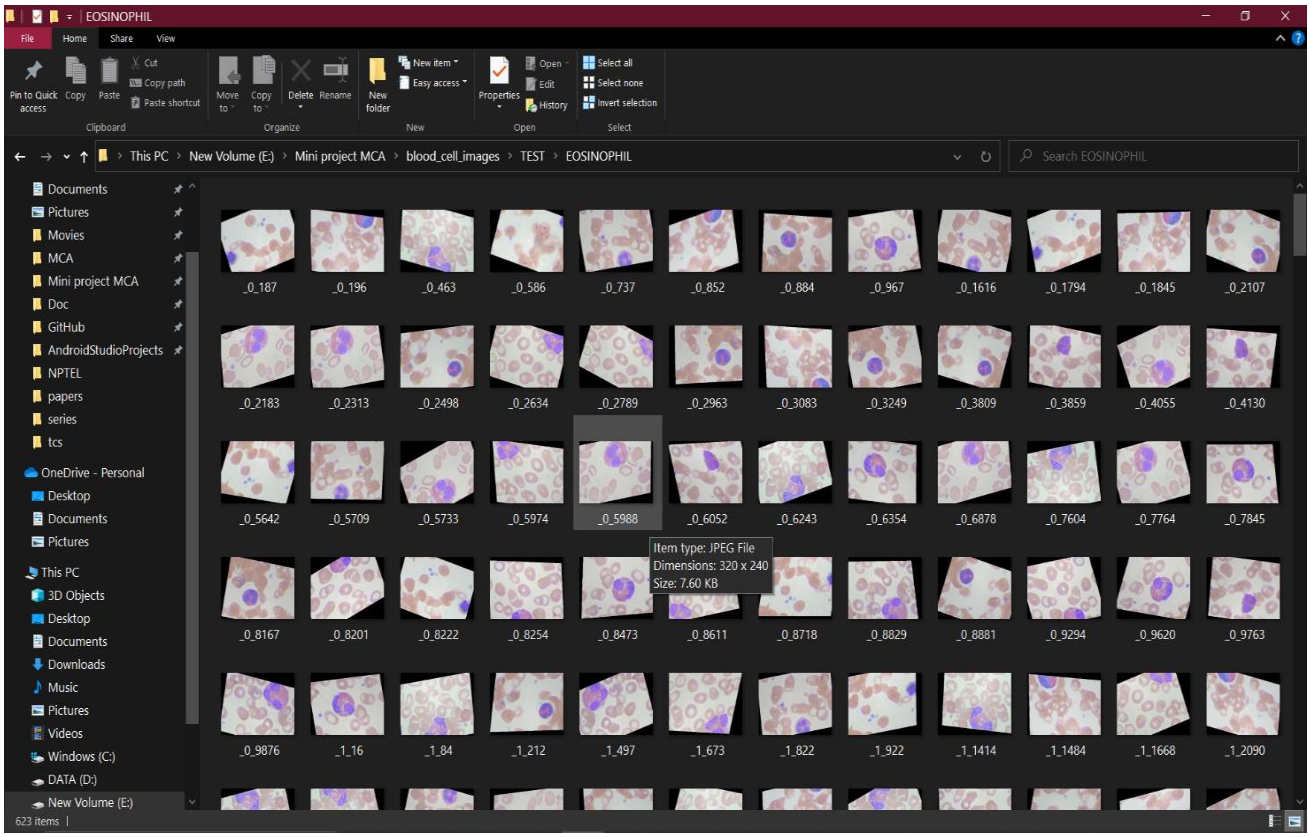

Fig 3.3: Sample from eosinophil folder of Dataset.

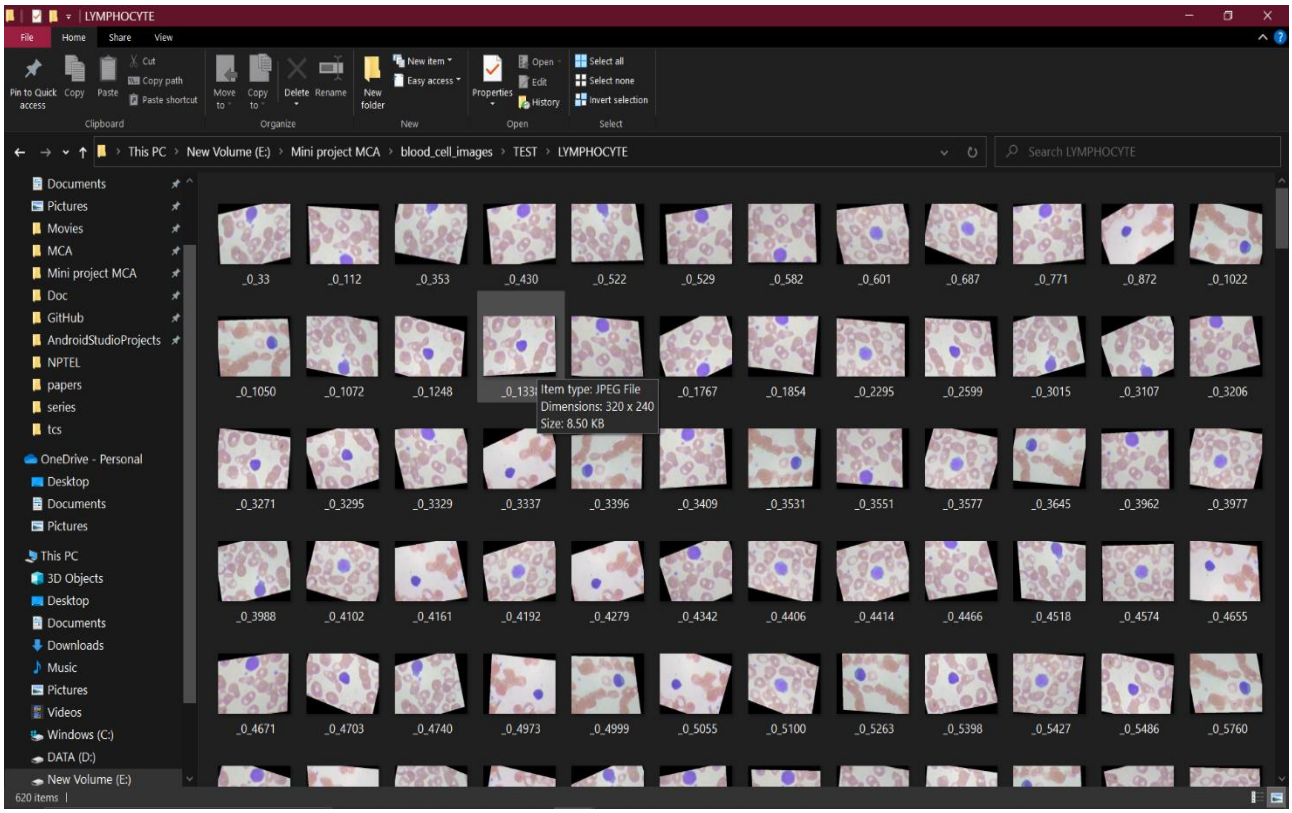

Fig 3.4: Sample from lymphocyte folder of Dataset.

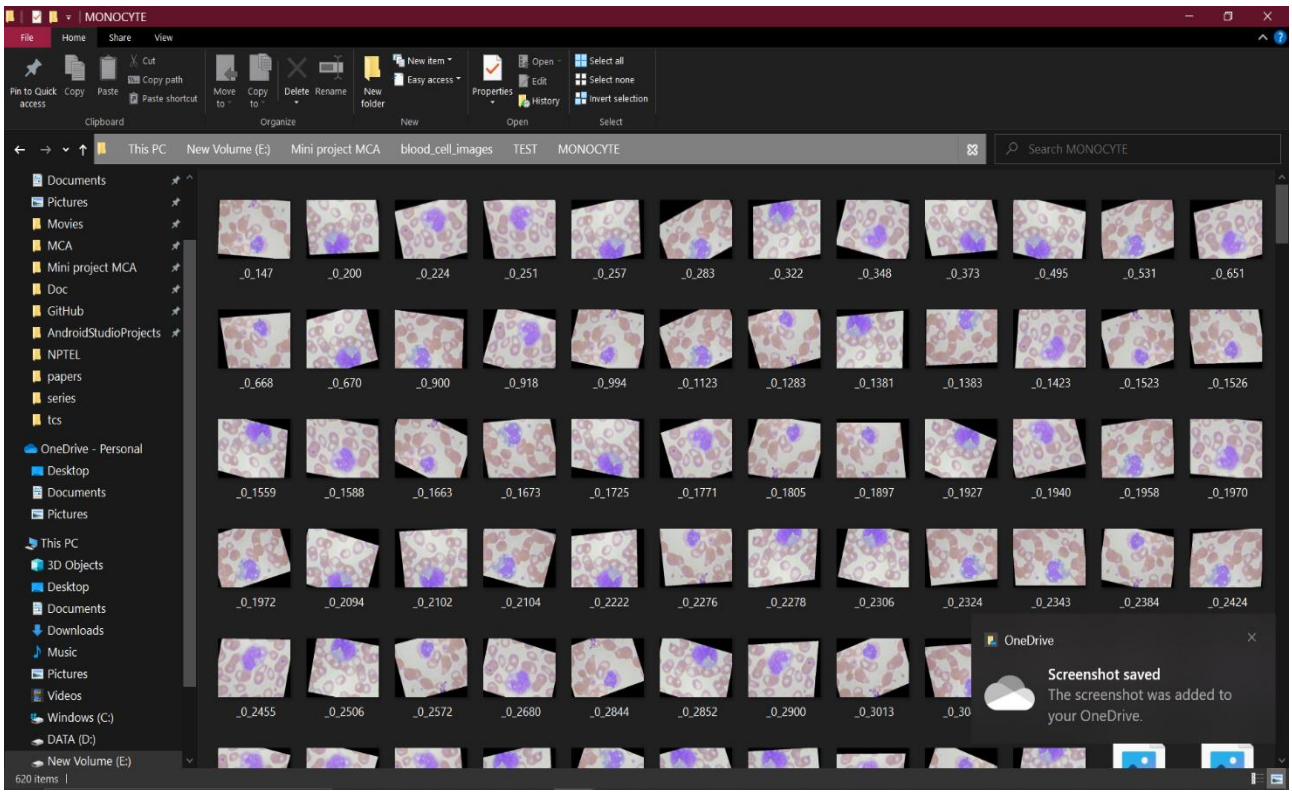

Fig 3.5: Sample from monocyte folder of Dataset.

3.2. Data Pre-processing

3.2.1. Data Cleaning

Images come in different shapes and sizes. They also come through different sources. For example, some images are what we call “natural images”, which means they are taken in colour, in the real world. Taking all these variations into consideration, we need to perform some pre-processing on any image data. RGB is the most popular encoding format, and most “natural images” we encounter are in RGB. Also, among the first step of data pre-processing is to make the images of the same size which is compatible for our architecture. Data generator supports pre-processing — it normalizes the images. The images are pre-processed during the time of training. Pre-processing of images involve applying augmentations such as rotation, width shift, height shift, shear, zoom, horizontal flip, and vertical flip. Augmentations are applied randomly to the frames in the dataset. Augmentation is usually done with data generators, i.e., the augmented data is generated batch-wise. By pre-processing we can suppress undesired distortions and enhance some features which are necessary for the particular application we are working for. The code below sets up a custom data generator:

```
train_gen = ImageDataGenerator(
  rotation_range=20,
  width_shift_range=0.1,
  height_shift_range=0.1,
  shear_range=0.1,
  zoom_range=0.2,
  horizontal_flip=True,
  vertical_flip=True,
  preprocessing_function=preprocess_input2
)
```

Fig 3.6: Dataset Pre-Processing.

3.2.2. Analysis of Feature Variables

Feature list is the attributes that describes the images and files in the dataset. The dataset is of size 129 MB. It contains 15 folders. These images are all of “jpeg” format with size approximately 8.35 KB for each image. The images in dataset are of size 240 x 320 x 3. Total 5106 images are there. About 420 images are pure images and others are augmented images.

3.2.3. Analysis of Class Variables

White blood cells are components of the blood that protect the body against disease and foreign invaders. Several types of white blood cell serve different functions. Most people will produce around 100 billion white blood cells every day. There are normally between 4,500 and 11,000 white blood cells in every microliter of blood, although this can vary according to sex, age, and race. Classes are sometimes called as targets/labels or categories. Class of the dataset is the category to which the input will be classified to. That means the final result of an application. In my work, I have 4 classes: - Eosinophil, Lymphocyte, Monocyte, and Neutrophil which are different types of WBC. Lymphocytes are vital for producing antibodies that help the body defend itself against bacteria, viruses, and other threats. Neutrophils are powerful white blood cells that destroy bacteria and fungi. Eosinophils are responsible for destroying parasites and cancer cells, and they are part of an allergic response. Monocytes are responsible for attacking and breaking down germs or bacteria that enter the body. White blood cells are colourless but can appear as a very light purple to pink colour. These have a round shape with a distinct centre membrane (nucleus). Neutrophils will appear spherical in shape with a dark stained nucleus that is segmented (2 to 5 lobes). A closer look will also reveal fine granules (neutrophilic granules) and thin threads connecting the nucleus lobes (chromatin threads). Compared to neutrophils that may have 2 to 5 lobed nuclei, eosinophils only have a bi-lobed (two lobes) nucleus that is shaped like a horse-shoe. They will also appear spherical in shape with fine granules referred to as acidophilic refractive granules. While they are smaller compared to other leukocytes, lymphocytes have a large round nucleus that takes up much of the cell volume. Lymphocytes have very little to no cytoplasm. Compared to lymphocytes (agranular leukocytes) monocytes are larger in size with a nucleus that is bean or kidney shaped. These cells also have more cytoplasm compared to lymphocytes.

3.3. Data Visualization

Data visualization is the graphical representation of information and data in a pictorial or graphical format (Example: charts, graphs, and maps). Data visualization tools provide an accessible way to see and understand trends, patterns in data, and outliers. Data visualization tools and technologies are essential to analysing massive amounts of information and making data-driven decisions. The concept of using pictures is to understand data that has been used

for centuries. General types of data visualization are Charts, Tables, Graphs, Maps, and Dashboards.

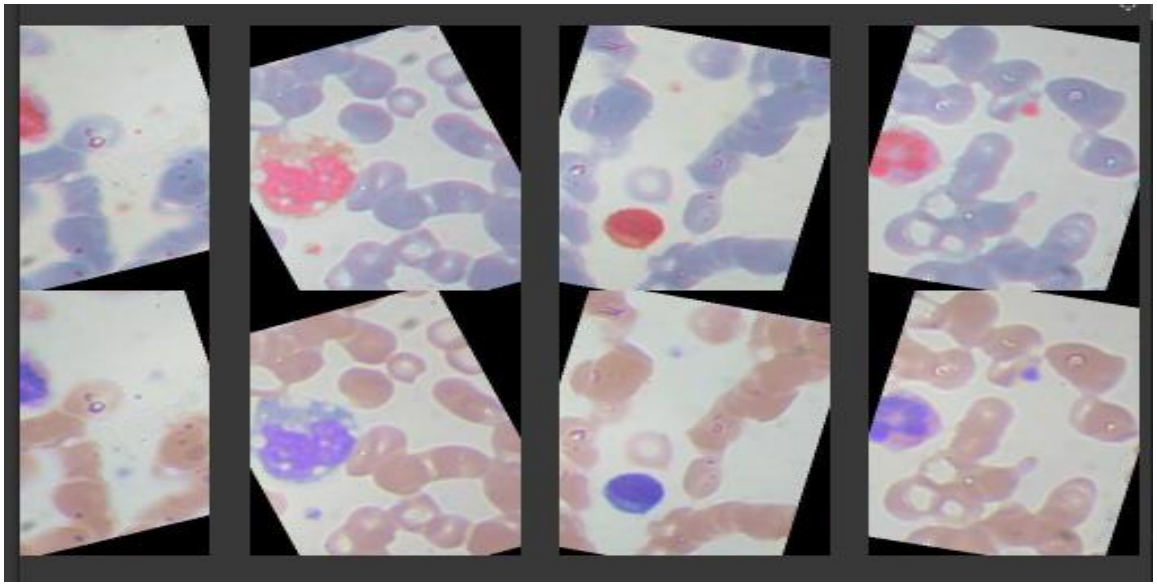

Fig 3.7: Data visualization.

3.4. Analysis of Architecture

3.4.1. Detailed study of Architecture

ResNet is one of the most powerful deep neural networks which has achieved fantabulous performance results in the ILSVRC 2015 classification challenge. ResNet has achieved excellent generalization performance on other recognition tasks and won the first place on ImageNet detection, ImageNet localization, COCO detection and COCO segmentation in ILSVRC and COCO 2015 competitions. There are many variants of ResNet architecture i.e. same concept but with a different number of layers. We have ResNet-18, ResNet-34, ResNet- 50, ResNet-101, ResNet-110, ResNet-152, ResNet-164, ResNet-1202 etc. The name ResNet followed by a two or more-digit number simply implies the ResNet architecture with a certain number of neural network layers.

Deep Residual Network is almost similar to the networks which have convolution, pooling, activation and fully-connected layers stacked one over the other. The only construction to the simple network to make it a residual network is the identity connection between the layers.

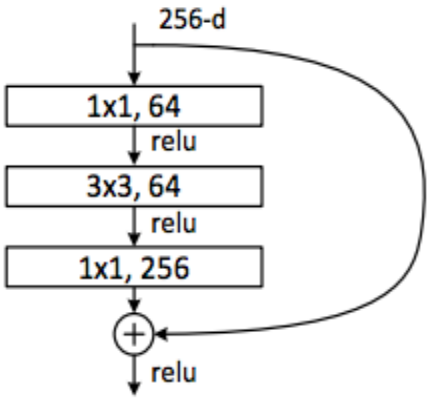

Fig 3.8: A residual block of deep residual network.

The architecture of this project can be called as ResNet24 according to above said because it has 24 layers as shown in the diagram below. Our model handles with 4 classes. The network can take the input image having height and width as 224 x 224. It has convolution layer, convolutional block and identity block.

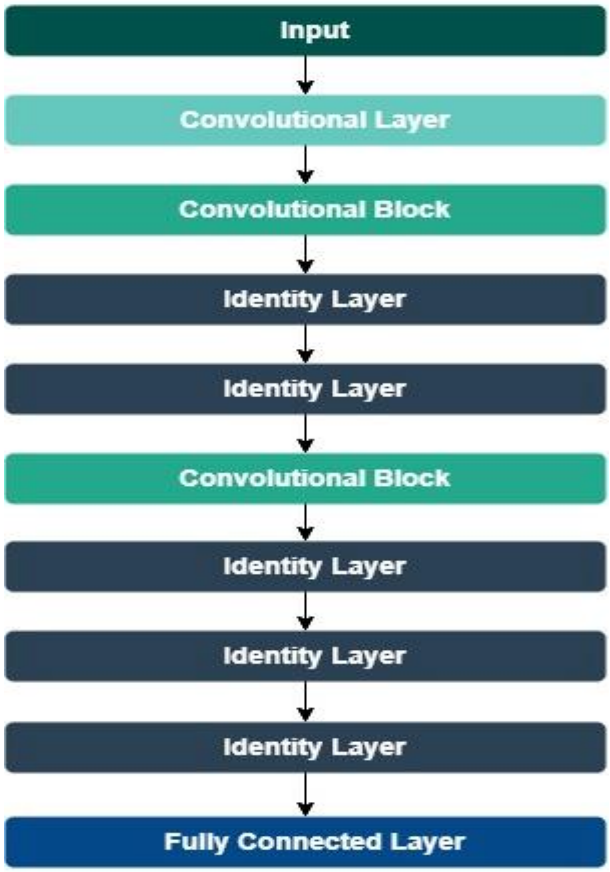

Fig 3.9: Architecture diagram

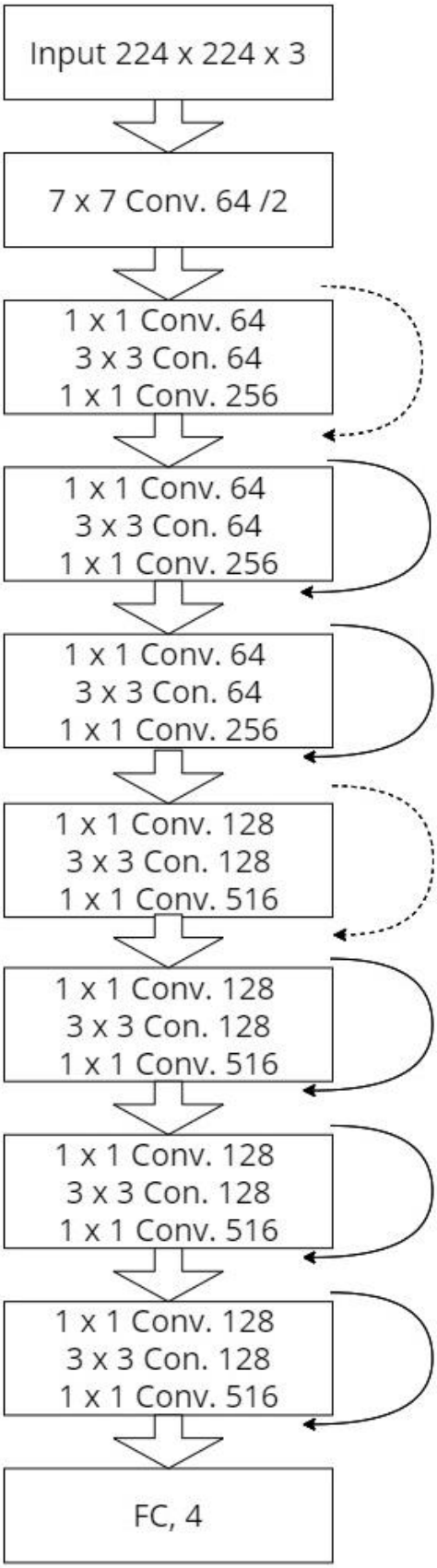

Fig 3.10: Block diagram

Like every ResNet architecture this also performs the initial convolution and max-pooling using 7×7 and 3×3 kernel sizes respectively. For deeper networks like ResNet50, ResNet152, etc, bottleneck design is used. For each residual function F, 3 layers are stacked one over the other. Here we use the same as that of deeper networks. The three layers are 1×1, 3×3, 1×1 convolutions. The 1×1 convolution layers are responsible for reducing and then restoring the dimensions. The 3×3 layer is left as a bottleneck with smaller input/output dimensions. Also, a convolutional block with 3 convolutional layers where first 2 layers are followed by batch normalization and activation function (ReLU). And the last layer is followed by an add function to add the inputted value of the block along with the block updated value and an activation function (ReLU). Here, 1 convolutional block is used after the first convolutional layer along with 2 identity blocks. Then another convolutional block along with 3 identity blocks. Finally, a fully connected layer.

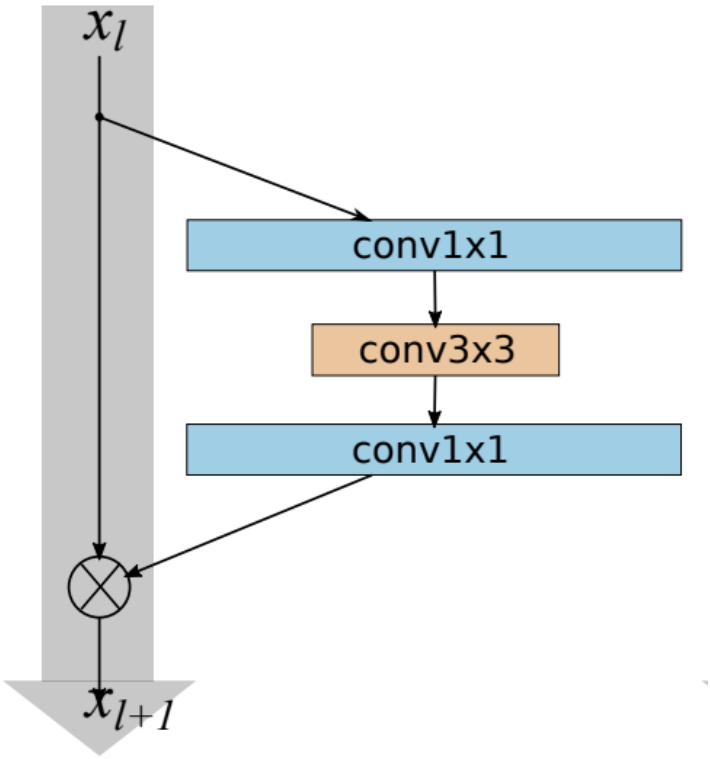

Fig 3.11: Bottleneck

Key Features of ResNet

- A residual network is formed by stacking several residual blocks together.

- ResNet uses Batch Normalization at its core. The Batch Normalization adjusts the input layer to increase the performance of the network. The problem of covariate shift is mitigated.
- ResNet makes use of the Skip Connection, which helps to protect the network from vanishing gradient problem.
- Deep Residual Network uses bottleneck residual block design to increase the performance of the network.

Skip Connection

In ResNet architecture, a "shortcut" or a "skip connection" alleviate the issue of vanishing gradient by setting up an alternate shortcut for the gradient to pass through. In addition, they enable the model to learn an identity function. This ensures that the higher layers of the model do not perform any worse than the lower layers. In short, the residual blocks make it considerably easier for the layers to learn identity functions. As a result, ResNet improves the efficiency of deep neural networks with more neural layers while minimizing the percentage of errors. In other words, the skip connections add the outputs from previous layers to the outputs of stacked layers, making it possible to train much deeper networks than previously possible.

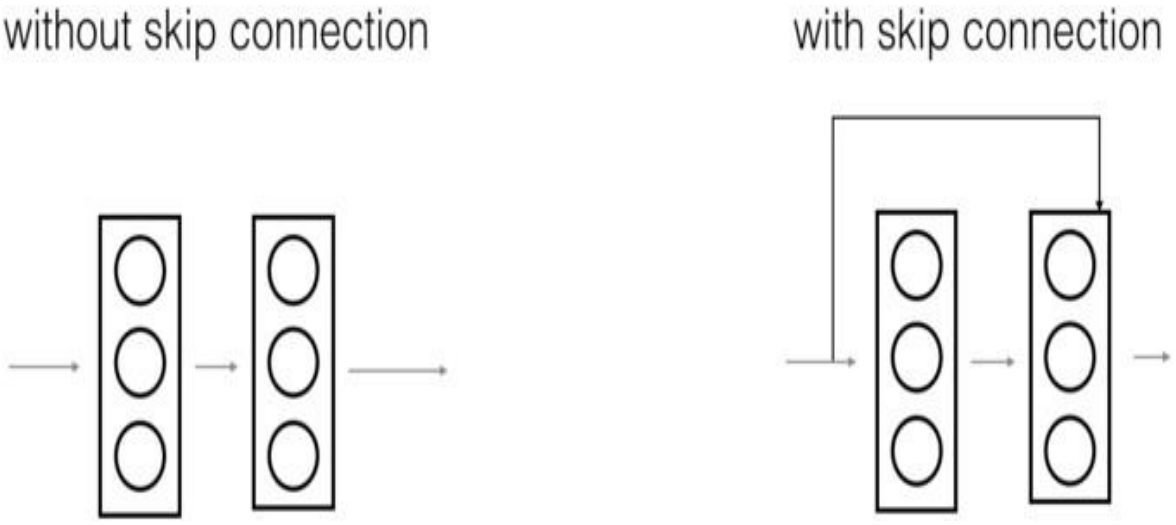

Fig 3.12: Skip Connection

The image on the left shows the “main path” through the network. The image on the right adds a shortcut to the main path. By stacking these ResNet blocks on top of each other, you can form a very deep network.

3.4.2. Diagrams and Details of Each Layer

1. Convolutional Layer

This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size 7 x 7. By sliding (stride size = 2) the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (7 x 7). The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

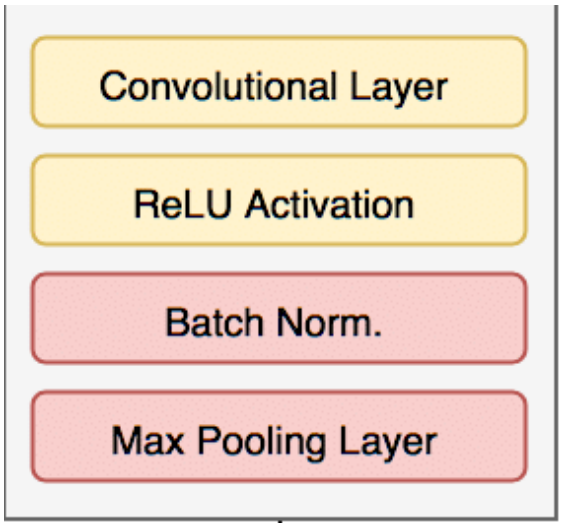

Fig 3.13: Convolutional Layer

2. Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations.

In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

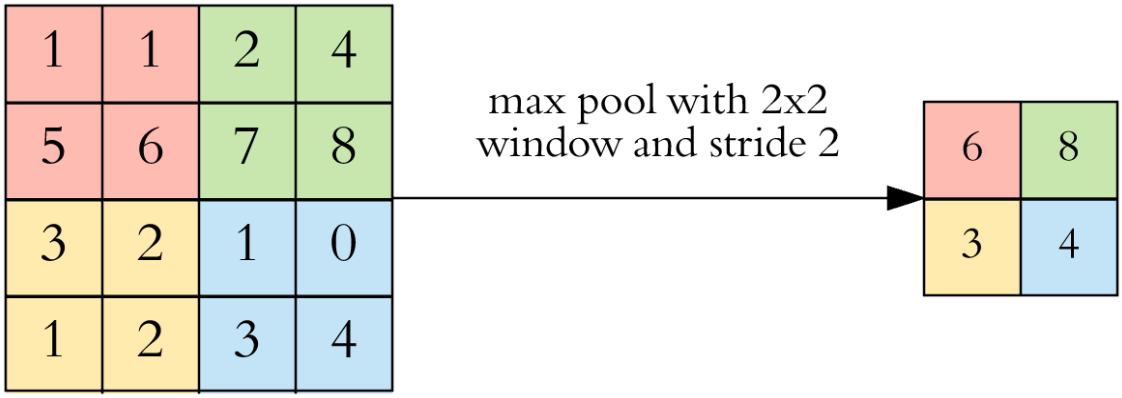

Fig 3.14: Process in max pooling layer

3. Convolutional Block

We can use this type of block when the input and output dimensions don't match up. The difference with the identity block is that there is a CONV layer in the shortcut path. In our architecture we have 3 layers along with our shortcut CONV layer.

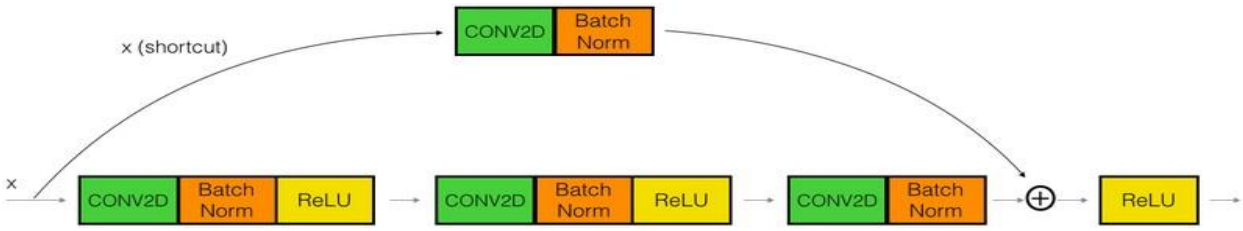

Fig 3.15: Convolution Block

4. Identity Block

The identity block is also called the residual block. It is the standard block used in ResNets and corresponds to the case where the input activation has the same dimension as the output activation. Here in our architecture each identity block contains 3 convolutional layers each. For each first two convolutional layers followed by batch normalization and activation function (ReLU). The last convolutional layer passes the value to a add function to add the inputted value and updated value along with an activation function.

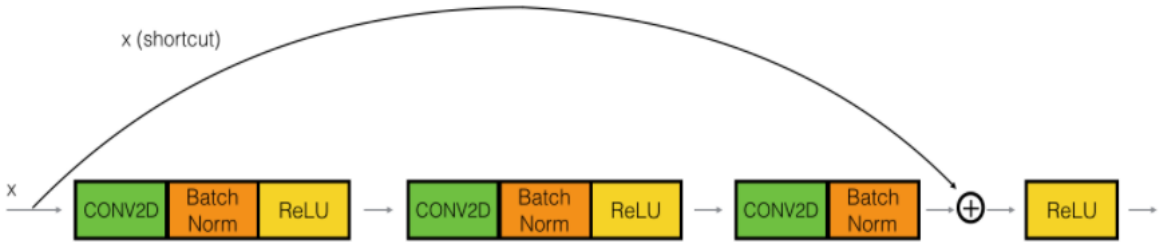

Fig 3.16: Identity block

5. Fully Connected Layer

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and from the last few layers of a CNN Architecture. In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few mathematical functions operations. In this stage, the classification process begins to take place.

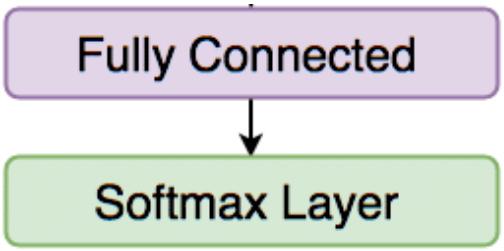

Fig 3.17: FC Layer

a. Activation Functions

Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network.It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred an for a multi-class classification, generally softmax us used. Before learning each layer, there are two parameters which are important in the working of

convolutional neural network layers, stride and padding. Here we use ReLU activation function for layers and for FC layer we use softmax.

b. Stride

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and soon. Stride is a component of convolutional neural networks, or neural networks tuned for the compression of images and video data. Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time. The size of the filter affects the encoded output volume, so stride is often set to a whole integer, rather than a fraction or decimal.

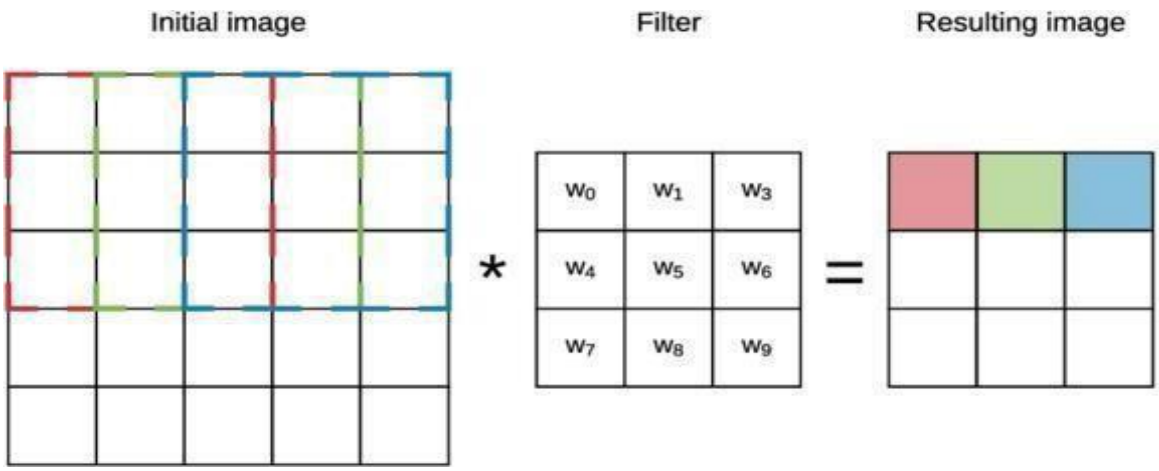

Fig 3.18: Convolution layer working with a stride of 1.

Imagine a convolutional neural network is taking an image and analysing the content. If the filter size is 3x3 pixels, the contained nine pixels will be converted down to 1 pixel in the output layer. Naturally, as the stride, or movement, is increased, the resulting output will be smaller. Stride is a parameter that works in conjunction with padding, the feature that adds blank, or empty pixels to the frame of the image to allow for a minimized reduction of size in the output layer. Roughly, it is a way of increasing the size of an image, to counter act the fact that stride reduces the size. Padding and stride are the foundational parameters of any convolutional neural network.

c. Padding

Sometimes filter does not perfectly fit the input image. we have two options:

- Pad the picture with zeros (zero-padding) so that it fits.
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

Padding is a term relevant to convolutional neural networks as it refers to the amount of pixels added to an image when it is being processed by the kernel of a CNN. For example, if the padding in a CNN is set to zero, then every pixel value that is added will be of value zero. If, however, the zero padding is set to one, there will be a one-pixel border added to the image with a pixel value of zero.

Padding works by extending the area of which a convolutional neural network processes an image. The kernel is the neural networks filter which moves across the image, scanning each pixel and converting the data into a smaller, or sometimes larger, format. In order to assist the kernel with processing the image, padding is added to the frame of the image to allow for more space for the kernel to cover the image. Adding padding to an image processed by a CNN allows for more accurate analysis of images.

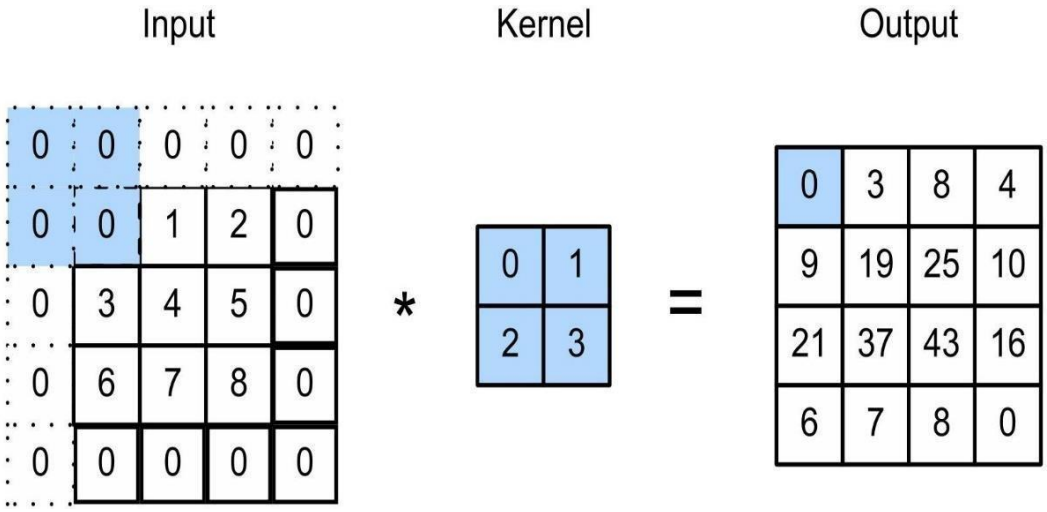

Fig 3.19: padding

The key building block in a convolutional neural network is the convolutional layer. We can visualize a convolutional layer as many small square templates, called convolutional kernels, which slide over the image and look for patterns. Where that part of the image

matches the kernel's pattern, the kernel returns a large positive value, and when there is no match, the kernel returns zero or a smaller value.

Convolution layers used trainable kernels or filters to perform convolution operations, sometimes including an optional trainable bias for each kernel. These convolution operations involved moving the kernels over the input in steps called strides. Generally, the larger the stride was, the more spaces the kernels skipped between each convolution. This led to less overall convolutions and more miniature output size. For each placement of a given kernel, a multiplication operation was performed between the input section and the kernel, with the bias summed to the result. This produced a feature map containing the convolved result. Thefeature maps were typically passed through an activation function to provide input for the subsequent layer.

Size of the feature map = [(input_size − kernel_size + 2 × padding) / stride] + 1.

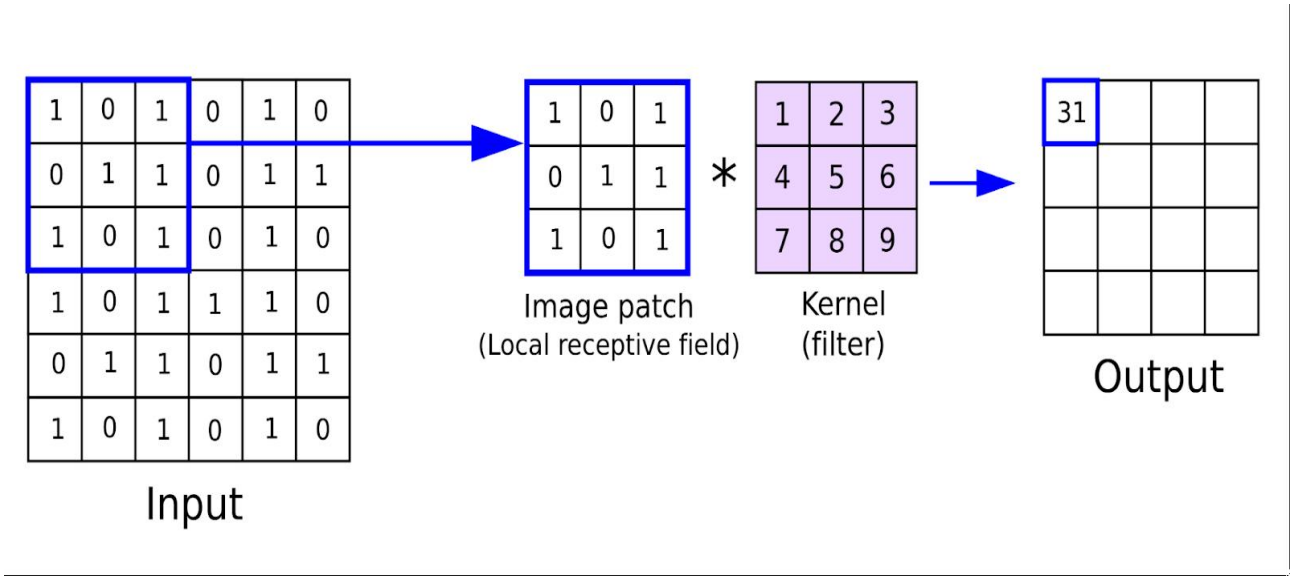

Fig 3.20: Working of convolutional layer.

d. ReLU

The rectified linear activation function or ReLU for short is a linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

$$f(x) = max(0, x)$$

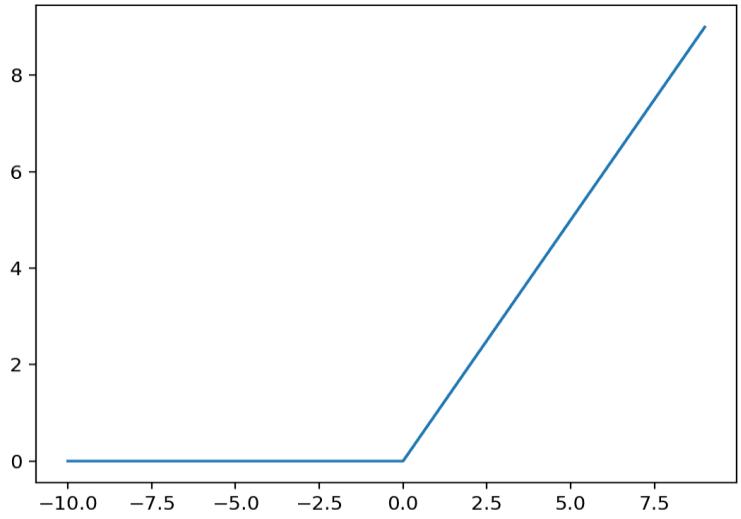

Fig 3.21: ReLU Graph.

e. Softmax

Softmax is often used as the activation for the last layer of a classification network because the result could be interpreted as a probability distribution.

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

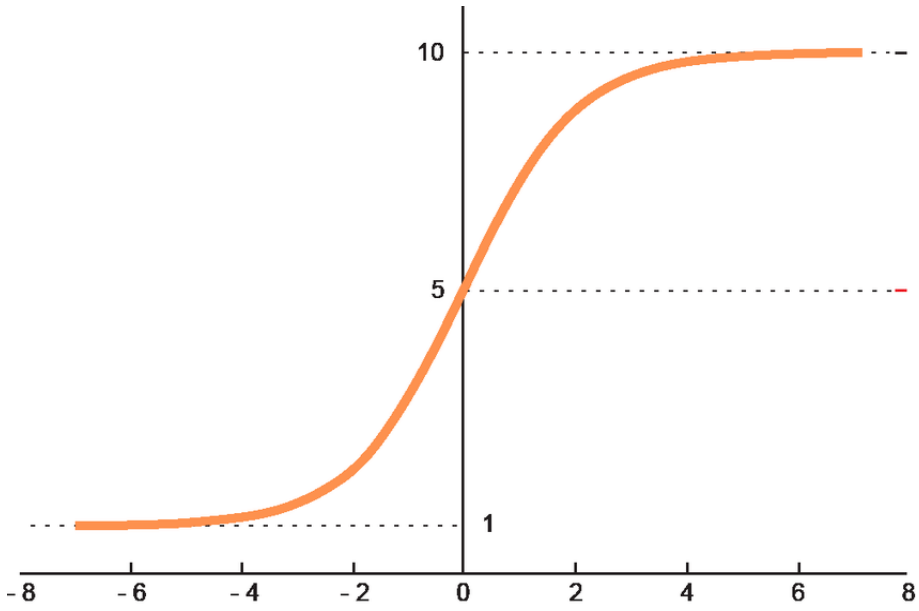

Fig 3.22: Softmax Graph.

3.4.3. Dimension Table

LAYER	KERNEL SIZE	FILTERS	STRIDE	OUTPUT SIZE
Input	-	-	-	224*224
CONV1	7*7	64	2	112*112
MAX_POOL	3*3		2	56*56
CONV2_X	1*1 3*3 1*1	64 64 256	-	56*56
CONV3_X	1*1 3*3 *2 1*1	64 64 *2 256	-	56*56
CONV4_X	1*1 3*3 1*1	128 128 512	-	28*28
CONV5_X	1*1 3*3 *3 1*1	128 128 *3 512	-	28*28
FC	-	-	-	1*1

Table 3.1: Dimension Table of custom ResNet

3.5. Project Pipeline

Project pipeline explains the project flow. The below figure explains the project flow during the developing stage of the project. Here first we gathered and analysed the dataset. The necessary data pre-processing has done like resizing it into architecture suitable data. The training data and validation data is used for training and also to keep trach of prediction. The model with highest accuracy is selected. Its saved hyperparameters are used to build the model and tested again. Then its UI is built and deployed i.e.; users can input the images and let the model predict the wbc type.

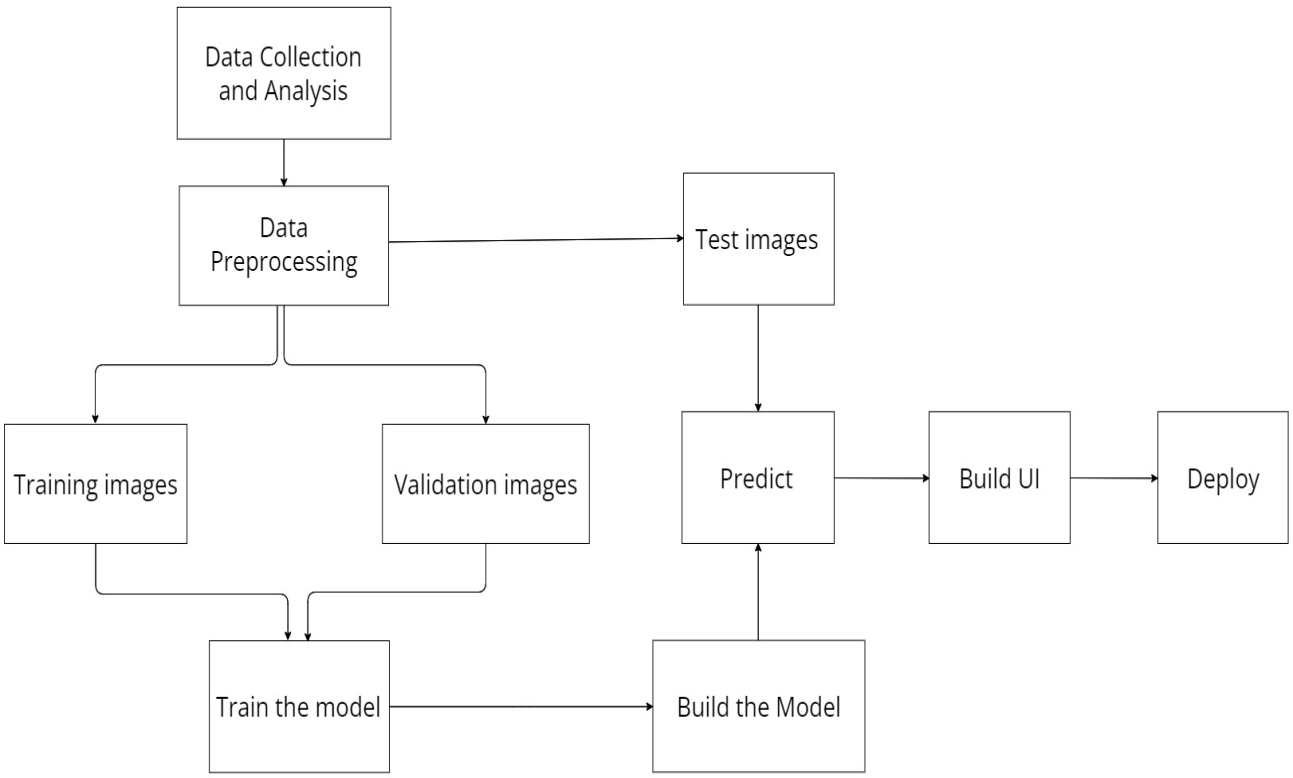

Fig 3.23: Project pipeline

3.6. Feasibility Analysis

A feasibility study aims to objectively and rationally uncover the strengths and weaknesses of an existing system or proposed system, opportunities and threats present in the natural environment, the resources required to carry through, and ultimately the prospects for success.

Evaluated the feasibility of the system in terms of the following categories:

- Technical Feasibility
- Economic Feasibility
- Operational Feasibility

3.6.1. Technical Feasibility

Proposed system is technically feasible since all the required tools are easily available. Technical issues involved are the necessary technology existence, technical guarantees of accuracy, reliability, ease of access, data security, and aspects of future expansion. The application is technically feasible because all the technical resources required for the development and working of the application is easily available and reliable. The project is implemented in Python. Since Python supports a various libraries and packages that make the project development easier, the project was technically feasible. The codes are written in Google Colab, therefore all the libraries will be available, no need to install or import each of those. These requirements are easily available, reliable, and will make the system more time saving and require less manpower.

3.6.2. Economic Feasibility

In our proposed system ” White Blood Cell Subtype Identification Using ResNet Architecture”, the development cost of the application is optimum. The system requires only a computer for working. The code is working on Google Colab and Jupyter notebook, The Colab can consumes an amount of internet. The development of the system will not need a huge amount of money. It will be economically feasible. But in Jupyter Notebook it needs high memory and time. But it doesn’t need internet.

3.6.3. Operational Feasibility

Operational feasibility assesses the extent to which the required system performs a series of steps to solve business problems and user requirements. Operational feasibility is mainly concerned with issues like whether the system will be used if it is developed and implemented. The developed system is completely driven and user friendly. Since the code is written on Google Colab, no need for worrying about importing or installing the libraries required. There is no need of skill for a new user to open this application and use it. The interface contain only a file upload option and a submit button. Users also need to be aware of the application initially. Then they can use it easily. So it is feasible. But sometimes it have a GPU issues. If we use jupyter notebook, then we must need a GPU in our system to get a faster user input and output.

3.7. System Environment

System environment specifies the hardware and software configuration of the new system. Regardless of how the requirement phase proceeds, it ultimately ends with the software requirement specification. A good SRS contains all the system requirements to a level of detail sufficient to enable designers to design a system that satisfies those requirements. The system specified in the SRS will assist the potential users to determine if the system meets their needs or how the system must be modified to meet their needs.

3.7.1. Software Environment

Various software used for the development of this application are the following:

- Python

Python is a high-level programming language that lets developers work quickly and integrate systems more efficiently. This model is developed by using many of the Python libraries and packages such as:

- Numpy:

NumPy is a Python library used for working with arrays. NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. In this application, its used for handling arrays.

- Matplotlib:

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. In this application, its used for plotting the graph.

- Tensorflow :

TensorFlow is an open-source library developed by Google primarily for deep learning applications. In this application, its used for creating and handling the model.

- Keras:

Keras is a powerful and easy-to-use free open-source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code. In this application, its used for creating and handling the model.

- Scikit-learn:

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. In this application it is used to plot confusion matrix.

- OS:

The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. In this application, its used for saving the model.

- Google Colab

Colab is a free Jupyter notebook environment that runs entirely in the cloud. We can write and execute code in Python. Colab supports many machine learning libraries which can be easily loaded in the colab notebook.

- Jupyter Notebook

The Jupyter Notebook is the original web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience. he Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet. In addition to displaying/editing/running notebook documents, the Jupyter Notebook App has a "Dashboard" (Notebook Dashboard), a "control panel" showing local files and allowing to open notebook documents or shutting down their kernels.

- Visual Studio Code

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs, such as Visual Studio IDE.

- HTML and CSS

Hyper Text Markup Language is used for creating web pages.HTML describes the structure of the web page. Here, the user interface of my project is done using HTML. Cascading Style Sheet is used with HTML to style the web pages.

- Github

Git is an open-source version control system that was started by Linus Torvalds. Git is similar to other version control systems Subversion, CVS, and Mercurial to name a few. Version control systems keep these revisions straight, storing the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute. Git is the preferred version control system of most developers, since it has multiple advantages over the other systems available. It stores file changes more efficiently and ensures file integrity better.The social networking aspect of GitHub is probably its most powerful feature, allowing projects to grow more than just about

any of the other features offered. Project revisions can be discussed publicly, so a mass of experts can contribute knowledge and collaborate to advance a project forward.

3.7.2. Hardware Environment

Selection of hardware configuration is very important task related to the software development. The hardware configuration of project done system is:

Processor : 2 GHz or faster (dual-core or quad-core will be much faster)

Memory : 8 GB RAM or greater

Disk space : 40 GB or greater good internet connectivity

4. SYSTEM DESIGN

4.1. Model Building

Model Building involves Model Planning, Model Training and Model Testing.

4.1.1. Model Planning

The architecture is a custom deeper Resnet with 24 layers. It handles with 4 classes Eosinophil, Lymphocyte, Monocyte and Neutrophil. The blood cell dataset has total 5106 images which is divided into 3 folders (Train, Test and Test_simple). Train folder has 2548 images for model training. Test has 2487 images for testing is to check whether the application is able to correctly predict the output. Test_simple has 71 images for validation. The images are pre-processed using ImageDataGenerator. The model is similar to Resnet50 but only first 23 layers and a fully connected layer. It contains one convolutional layer, 2 convolutional block and 5 identity blocks following each convolutional block.

4.1.2. Training

Training data is the initial dataset we use to teach a deep learning application to recognize pattern. Training dataset is used to train our model, in order to get correct predictions by the model. The model is similar to Resnet50 but only first 23 layers and a fully connected layer of ResNet50. It contains one convolutional layer, 2 convolutional block and 5 identity blocks following each convolutional block. A convolutional block contains 3 convolutional layers. An identity block also contains 3 convolutional layers. Each convolution is followed by batch normalization and relu activation function. The final dense layer is followed by a softmax activation function. This customized resnet model is developed using Tensorflow and Keras libraries with 24 Conv layers. For the development identity blocks and convolutional blocks are defined as a function. To improve generalization, the convolutional layers use ReLU activation functions that introduce non-linearities and Batch Normalization to regularize their output and prevent the vanishing gradient effect. Since I got 96% accuracy for training data and 90% accuracy for validation data, no changes to the architecture were made. I had set 20 epochs with an early stopping feature so it ran 19 epochs and achieved training accuracy of 96%. The model summary is shown below:

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 224, 224, 3)]	0	[]
zero_padding2d (ZeroPadding2D)	(None, 230, 230, 3)	0	['input_1[0][0]']
conv2d (Conv2D)	(None, 112, 112, 64)	9472	['zero_padding2d[0][0]']
batch_normalization (BatchNormalization)	(None, 112, 112, 64)	256	['conv2d[0][0]']
activation (Activation)	(None, 112, 112, 64)	0	['batch_normalization[0][0]']
zero_padding2d_1 (ZeroPadding2D)	(None, 114, 114, 64)	0	['activation[0][0]']
max_pooling2d (MaxPooling2D)	(None, 56, 56, 64)	0	['zero_padding2d_1[0][0]']

Fig 4.1: Model summary.

conv2d_1 (Conv2D)	(None, 56, 56, 64)	4160	['max_pooling2d[0][0]']
batch_normalization_1 (BatchNormalization)	(None, 56, 56, 64)	256	['conv2d_1[0][0]']
activation_1 (Activation)	(None, 56, 56, 64)	0	['batch_normalization_1[0][0]']
conv2d_2 (Conv2D)	(None, 56, 56, 64)	36928	['activation_1[0][0]']
batch_normalization_2 (BatchNormalization)	(None, 56, 56, 64)	256	['conv2d_2[0][0]']
activation_2 (Activation)	(None, 56, 56, 64)	0	['batch_normalization_2[0][0]']
conv2d_3 (Conv2D)	(None, 56, 56, 256)	16640	['activation_2[0][0]']
conv2d_4 (Conv2D)	(None, 56, 56, 256)	16640	['max_pooling2d[0][0]']
batch_normalization_3 (BatchNormalization)	(None, 56, 56, 256)	1024	['conv2d_3[0][0]']
batch_normalization_4 (BatchNo	(None, 56, 56, 256)	1024	['conv2d_4[0][0]']

Fig 4.2: Model summary.

```
add (Add)                      (None, 56, 56, 256)  0          ['batch_normalization_3[0][0]',
                                                                'batch_normalization_4[0][0]']

activation_3 (Activation)      (None, 56, 56, 256)  0          ['add[0][0]']

conv2d_5 (Conv2D)              (None, 56, 56, 64)   16448      ['activation_3[0][0]']

batch_normalization_5 (BatchNo  (None, 56, 56, 64)  256        ['conv2d_5[0][0]']
rmalization)

activation_4 (Activation)      (None, 56, 56, 64)   0          ['batch_normalization_5[0][0]']

conv2d_6 (Conv2D)              (None, 56, 56, 64)   36928      ['activation_4[0][0]']

batch_normalization_6 (BatchNo  (None, 56, 56, 64)  256        ['conv2d_6[0][0]']
rmalization)

activation_5 (Activation)      (None, 56, 56, 64)   0          ['batch_normalization_6[0][0]']

conv2d_7 (Conv2D)              (None, 56, 56, 256)  16640      ['activation_5[0][0]']
```

Fig 4.3: Model summary.

```
batch_normalization_7 (BatchNo  (None, 56, 56, 256)  1024       ['conv2d_7[0][0]']
rmalization)

add_1 (Add)                    (None, 56, 56, 256)  0          ['batch_normalization_7[0][0]',
                                                                'activation_3[0][0]']

activation_6 (Activation)      (None, 56, 56, 256)  0          ['add_1[0][0]']

conv2d_8 (Conv2D)              (None, 56, 56, 64)   16448      ['activation_6[0][0]']

batch_normalization_8 (BatchNo  (None, 56, 56, 64)  256        ['conv2d_8[0][0]']
rmalization)

activation_7 (Activation)      (None, 56, 56, 64)   0          ['batch_normalization_8[0][0]']

conv2d_9 (Conv2D)              (None, 56, 56, 64)   36928      ['activation_7[0][0]']

batch_normalization_9 (BatchNo  (None, 56, 56, 64)  256        ['conv2d_9[0][0]']
rmalization)

activation_8 (Activation)      (None, 56, 56, 64)   0          ['batch_normalization_9[0][0]']
```

Fig 4.4: Model summary.

```
conv2d_10 (Conv2D)             (None, 56, 56, 256)  16640      ['activation_8[0][0]']

batch_normalization_10 (BatchN  (None, 56, 56, 256)  1024      ['conv2d_10[0][0]']
```

Fig 4.5: Model summary.

```
add_2 (Add)                 (None, 56, 56, 256)  0          ['batch_normalization_10[0][0]',
                                                            'activation_6[0][0]']

activation_9 (Activation)   (None, 56, 56, 256)  0          ['add_2[0][0]']

conv2d_11 (Conv2D)          (None, 28, 28, 128)  32896      ['activation_9[0][0]']

batch_normalization_11 (BatchN  (None, 28, 28, 128)  512    ['conv2d_11[0][0]']
ormalization)

activation_10 (Activation)  (None, 28, 28, 128)  0          ['batch_normalization_11[0][0]']

conv2d_12 (Conv2D)          (None, 28, 28, 128)  147584     ['activation_10[0][0]']

batch_normalization_12 (BatchN  (None, 28, 28, 128)  512   ['conv2d_12[0][0]']
ormalization)

activation_11 (Activation)  (None, 28, 28, 128)  0          ['batch_normalization_12[0][0]']

conv2d_13 (Conv2D)          (None, 28, 28, 512)  66048      ['activation_11[0][0]']

conv2d_14 (Conv2D)          (None, 28, 28, 512)  131584     ['activation_9[0][0]']
```

Fig 4.6: Model summary.

```
batch_normalization_13 (BatchN  (None, 28, 28, 512)  2048      ['conv2d_13[0][0]']
ormalization)

batch_normalization_14 (BatchN  (None, 28, 28, 512)  2048      ['conv2d_14[0][0]']
ormalization)

add_3 (Add)                 (None, 28, 28, 512)  0          ['batch_normalization_13[0][0]'
                                                            'batch_normalization_14[0][0]'

activation_12 (Activation)  (None, 28, 28, 512)  0          ['add_3[0][0]']

conv2d_15 (Conv2D)          (None, 28, 28, 128)  65664      ['activation_12[0][0]']

batch_normalization_15 (BatchN  (None, 28, 28, 128)  512   ['conv2d_15[0][0]']
ormalization)

activation_13 (Activation)  (None, 28, 28, 128)  0          ['batch_normalization_15[

conv2d_16 (Conv2D)          (None, 28, 28, 128)  147584     ['activation_13[0][0]']

batch_normalization_16 (BatchN  (None, 28, 28, 128)  512   ['conv2d_16[0][0]']
ormalization)
```

Fig 4.7: Model summary.

```
activation_14 (Activation)     (None, 28, 28, 128)  0          ['batch_normalization_16[0][0]']

conv2d_17 (Conv2D)             (None, 28, 28, 512)  66048      ['activation_14[0][0]']

batch_normalization_17 (BatchN  (None, 28, 28, 512)  2048      ['conv2d_17[0][0]']
ormalization)

add_4 (Add)                    (None, 28, 28, 512)  0          ['batch_normalization_17[0][0]',
                                                               'activation_12[0][0]']

activation_15 (Activation)     (None, 28, 28, 512)  0          ['add_4[0][0]']

conv2d_18 (Conv2D)             (None, 28, 28, 128)  65664      ['activation_15[0][0]']

batch_normalization_18 (BatchN  (None, 28, 28, 128)  512       ['conv2d_18[0][0]']
ormalization)

activation_16 (Activation)     (None, 28, 28, 128)  0          ['batch_normalization_18[0][0]']

conv2d_19 (Conv2D)             (None, 28, 28, 128)  147584     ['activation_16[0][0]']

batch_normalization_19 (BatchN  (None, 28, 28, 128)  512       ['conv2d_19[0][0]']
```

Fig 4.8: Model summary.

```
activation_17 (Activation)     (None, 28, 28, 128)  0          ['batch_normalization_19[0][0]']

conv2d_20 (Conv2D)             (None, 28, 28, 512)  66048      ['activation_17[0][0]']

batch_normalization_20 (BatchN  (None, 28, 28, 512)  2048      ['conv2d_20[0][0]']
ormalization)

add_5 (Add)                    (None, 28, 28, 512)  0          ['batch_normalization_20[0][0]',
                                                               'activation_15[0][0]']

activation_18 (Activation)     (None, 28, 28, 512)  0          ['add_5[0][0]']

conv2d_21 (Conv2D)             (None, 28, 28, 128)  65664      ['activation_18[0][0]']

batch_normalization_21 (BatchN  (None, 28, 28, 128)  512       ['conv2d_21[0][0]']
ormalization)

activation_19 (Activation)     (None, 28, 28, 128)  0          ['batch_normalization_21[0][0]']

conv2d_22 (Conv2D)             (None, 28, 28, 128)  147584     ['activation_19[0][0]']

batch_normalization_22 (BatchN  (None, 28, 28, 128)  512       ['conv2d_22[0][0]']
ormalization)

activation_20 (Activation)     (None, 28, 28, 128)  0          ['batch_normalization_22[
```


Fig 4.9: Model summary.

```
conv2d_23 (Conv2D)              (None, 28, 28, 512)  66048       ['activation_20[0][0]']

batch_normalization_23 (BatchN  (None, 28, 28, 512)  2048        ['conv2d_23[0][0]']
ormalization)

add_6 (Add)                     (None, 28, 28, 512)  0           ['batch_normalization_23[0][0]',
                                                                 'activation_18[0][0]']

activation_21 (Activation)      (None, 28, 28, 512)  0           ['add_6[0][0]']

flatten (Flatten)               (None, 401408)       0           ['activation_21[0][0]']

dense (Dense)                   (None, 4)            1605636     ['flatten[0][0]']

==================================================================================================
Total params: 3,065,732
Trainable params: 3,055,620
Non-trainable params: 10,112
__________________________________________________________________________________________________
```

Fig 4.10: Model summary.

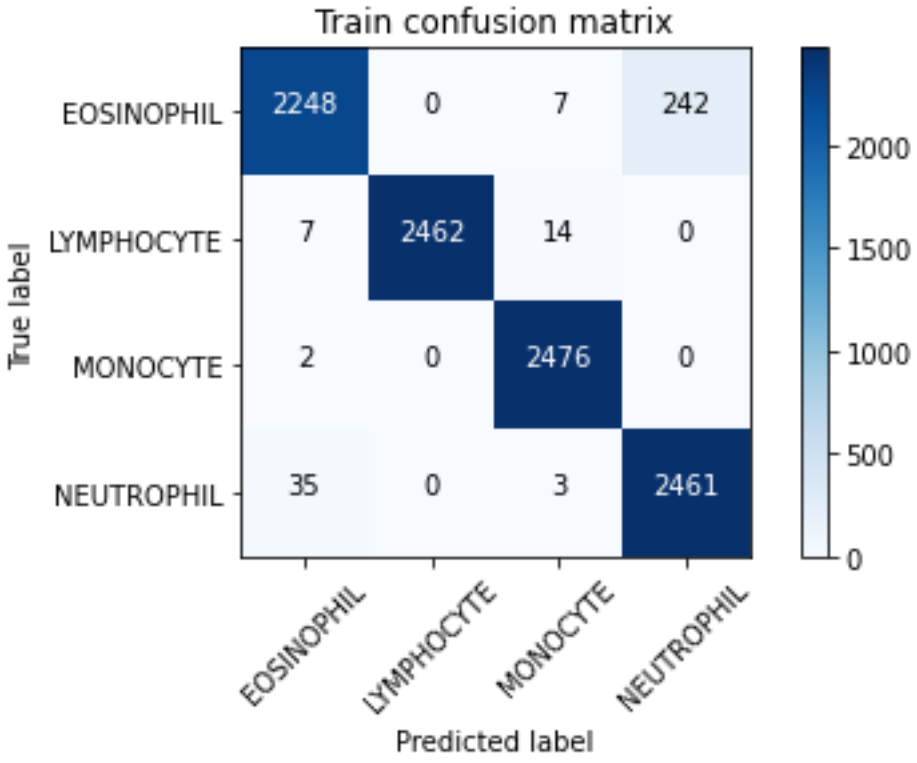

Fig 4.11: Train confusion matrix.

```
Epoch 1/20
77/77 [==============================] - 153s 2s/step - loss: 8.5784 - accuracy: 0.2881 - val_loss: 3.3456 - val_accuracy: 0.2821
Epoch 2/20
77/77 [==============================] - 150s 2s/step - loss: 2.0851 - accuracy: 0.4394 - val_loss: 1.8125 - val_accuracy: 0.4626
Epoch 3/20
77/77 [==============================] - 148s 2s/step - loss: 1.3007 - accuracy: 0.6196 - val_loss: 2.3210 - val_accuracy: 0.4494
Epoch 4/20
77/77 [==============================] - 153s 2s/step - loss: 0.9019 - accuracy: 0.7498 - val_loss: 1.1379 - val_accuracy: 0.6776
Epoch 5/20
77/77 [==============================] - 148s 2s/step - loss: 0.5292 - accuracy: 0.8423 - val_loss: 1.4047 - val_accuracy: 0.6817
Epoch 6/20
77/77 [==============================] - 150s 2s/step - loss: 0.4111 - accuracy: 0.8808 - val_loss: 0.8639 - val_accuracy: 0.7673
Epoch 7/20
77/77 [==============================] - 148s 2s/step - loss: 0.3055 - accuracy: 0.9126 - val_loss: 0.6858 - val_accuracy: 0.8277
Epoch 8/20
77/77 [==============================] - 147s 2s/step - loss: 0.3357 - accuracy: 0.9174 - val_loss: 3.2723 - val_accuracy: 0.6238
Epoch 9/20
77/77 [==============================] - 149s 2s/step - loss: 0.2653 - accuracy: 0.9351 - val_loss: 0.9567 - val_accuracy: 0.8273
Epoch 10/20
77/77 [==============================] - 150s 2s/step - loss: 0.3846 - accuracy: 0.9251 - val_loss: 0.8964 - val_accuracy: 0.8446
Epoch 11/20
77/77 [==============================] - 149s 2s/step - loss: 0.2259 - accuracy: 0.9462 - val_loss: 0.8906 - val_accuracy: 0.8475
Epoch 12/20
77/77 [==============================] - 150s 2s/step - loss: 0.2243 - accuracy: 0.9536 - val_loss: 2.0572 - val_accuracy: 0.8137
Epoch 13/20
77/77 [==============================] - 148s 2s/step - loss: 0.2326 - accuracy: 0.9504 - val_loss: 1.3918 - val_accuracy: 0.8326
Epoch 14/20
77/77 [==============================] - 150s 2s/step - loss: 0.1714 - accuracy: 0.9632 - val_loss: 1.9810 - val_accuracy: 0.8170
Epoch 15/20
77/77 [==============================] - 149s 2s/step - loss: 0.1732 - accuracy: 0.9651 - val_loss: 1.5645 - val_accuracy: 0.8240
Epoch 16/20
77/77 [==============================] - 148s 2s/step - loss: 0.1315 - accuracy: 0.9738 - val_loss: 0.9877 - val_accuracy: 0.9025
Epoch 17/20
77/77 [==============================] - 147s 2s/step - loss: 0.1984 - accuracy: 0.9625 - val_loss: 5.0033 - val_accuracy: 0.7204
Epoch 18/20
77/77 [==============================] - 148s 2s/step - loss: 0.2592 - accuracy: 0.9579 - val_loss: 2.3475 - val_accuracy: 0.8067
Epoch 19/20
77/77 [==============================] - 148s 2s/step - loss: 0.2323 - accuracy: 0.9630 - val_loss: 2.1610 - val_accuracy: 0.8207
```

Fig 4.12: Epochs of Model

4.1.3. Testing

Testing or validation data is used to evaluate our model's accuracy. To check whether the application is able to correctly predict the output. The models are tested along with training. At each epoch the model tries to predict the unseen test data. The accuracy on the test dataset changes with each epoch and the best model with maximum accuracy and minimum loss is selected. I had set 20 epochs with an early stopping feature so it ran 19 epochs and achieved testing accuracy of 82% during final epoch from which the maximum testing accuracy is 90%. Therefore, that epochs hyperparameters are selected to build the model. minimum validation loss 0.9877 is achieved in the 17th epoch.

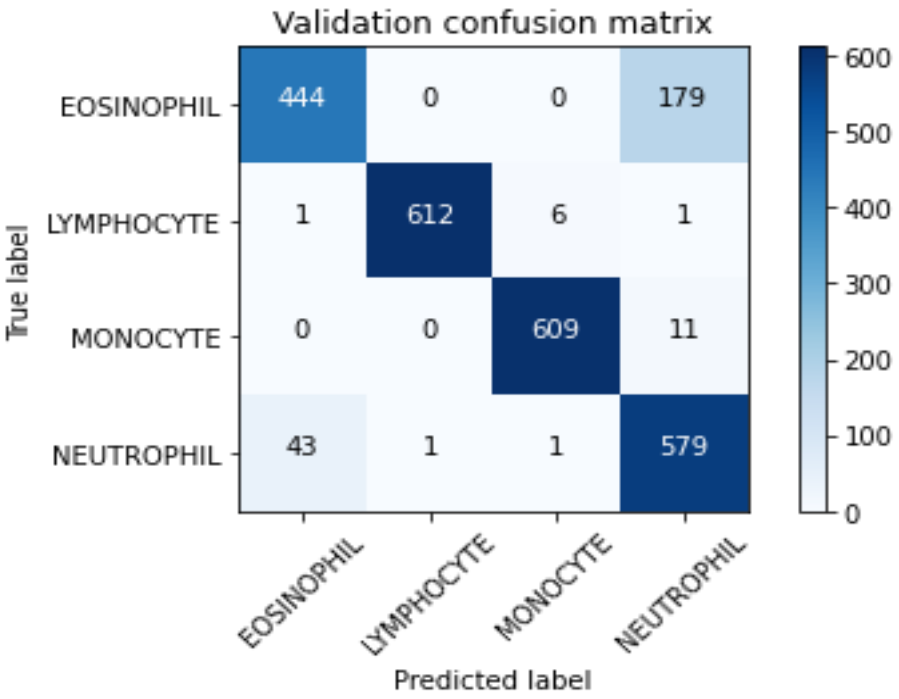

Fig 4.13: Validation confusion matrix.

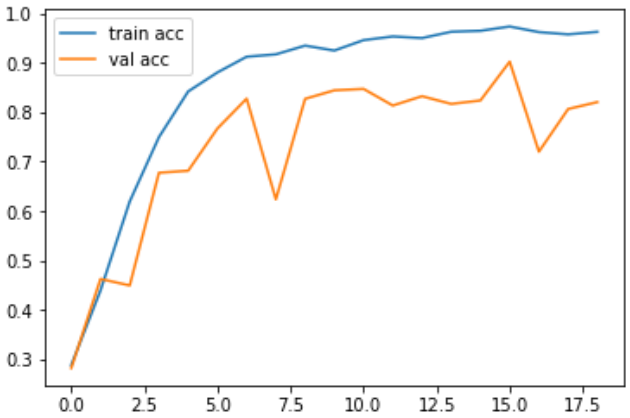

Fig 4.14:Accuracy graph.

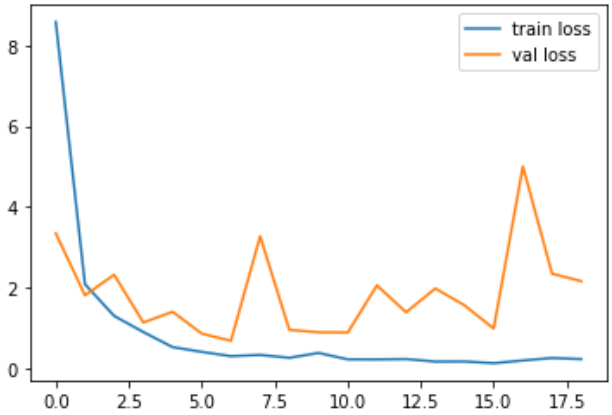

Fig 4.15: Loss graph.

5. RESULTS AND DISCUSSION

My aim was to develop an application that will detect WBC cell and identify the type of WBC. By now, I successfully completed my project. The model developed using Custom ResNet architecture is producing accurate results. The accuracy is the metrics used in the training of the dataset. Accuracy is a measurement of observational error. It defines how close or far off a given set of measurement are to their true value. This model showed better accuracy with increase in each epoch during training. Even though I got models with higher training accuracy during the model training, I selected the model saved at the 17th epoch with validation accuracy 90.25% and minimum validation loss 0.9877. This shows that even at high validation accuracy the model may not provide the minimum validation loss. This shows that by increasing the number of epochs in training, we can improve the quality of prediction. It is like, more data for training the system is, more correct the prediction will be.

6. MODEL DEPLOYMENT

This figure shows the user interface of this application. The interface is very simple and easy to understand. There are only some elements displayed on the screen. There is a file upload option provided. The user can choose blood smear image from the local storage through that. Then there is a submit button. On click of the button, the uploaded image will be given to the models. After processing for few seconds, the results returned by the models are compared and the output is displayed. The output may be one of the 4 labels Eosinophil, Neutrophil, Monocyte and Lymphocyte.

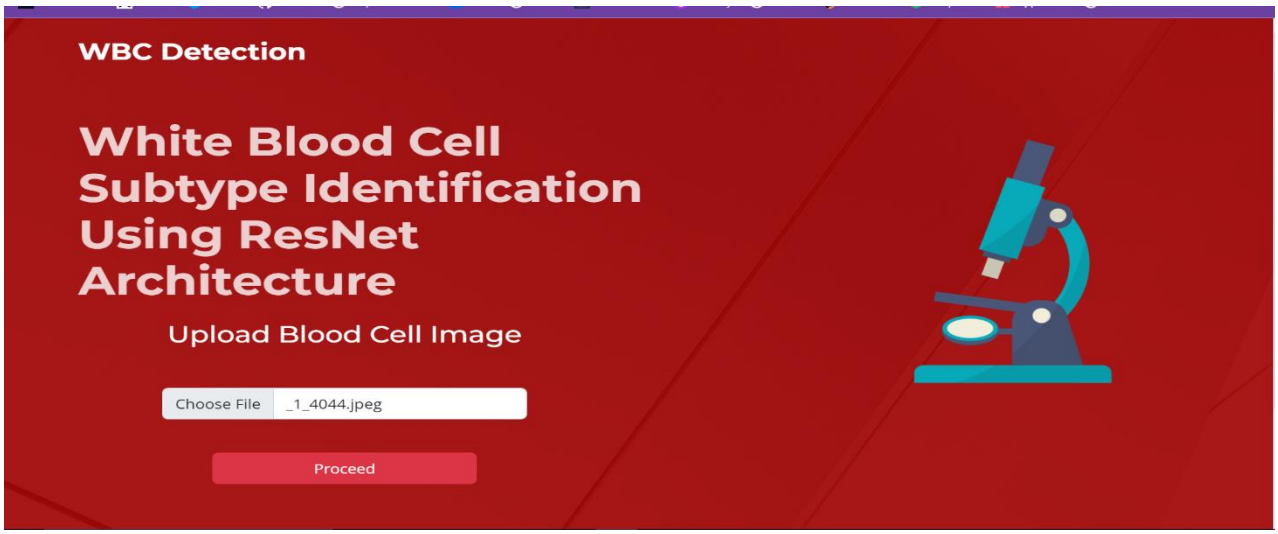

Fig 6.1: UI

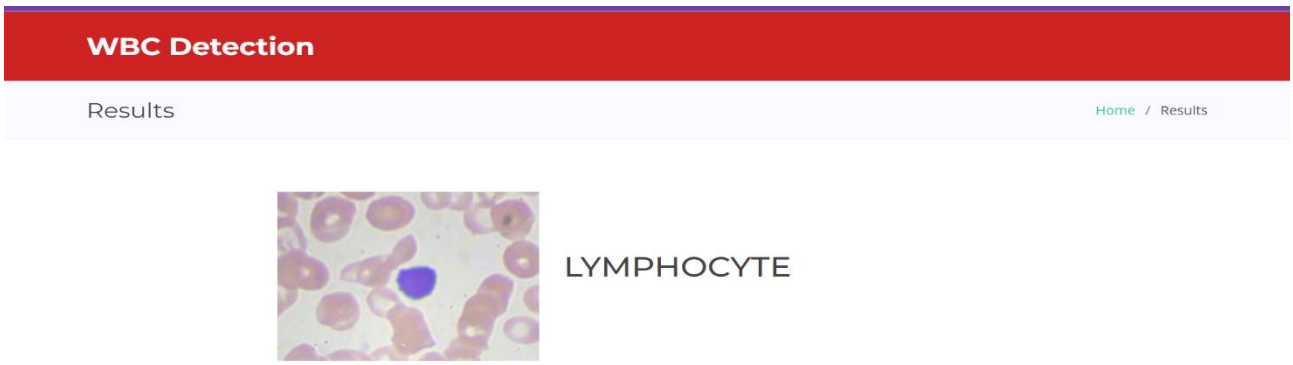

Fig 6.2: UI

7. GIT HISTORY

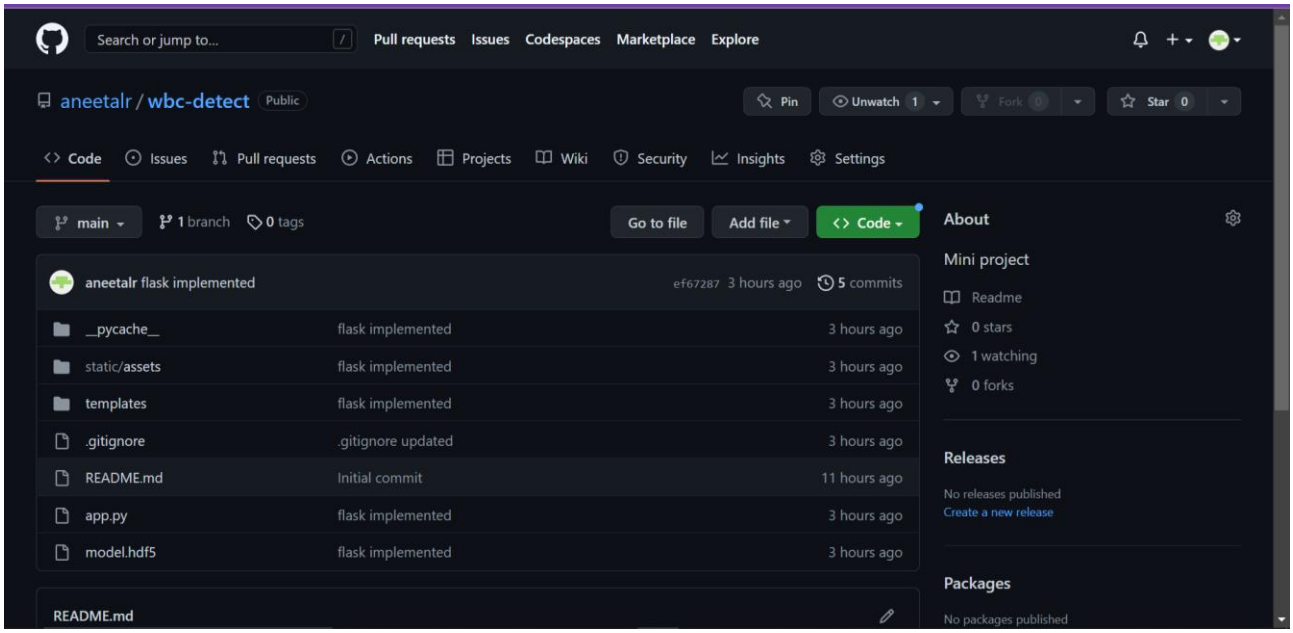

Fig 7.1: Git history.

8. CONCLUSION

This project is a deep learning project, which aims to detect WBC cell and identify its subtype. The idea to develop this application as my academic project came to my mind after reading some research papers. As the paper suggested it can be done using CNN which is deeper. While going through some CNN papers I found that ResNet is the most efficient one among other CNN architectures. As a part of my contribution in thought of customizing the ResNet. Due to lesser number of classes in this project I decided to go with a less number layer ResNet. So, I took the first 23 layers of ResNet50 and a fully connected layer. Because ResNet50 has better performance than other architectures, ResNet18 and ResNet50 and also it is the most commonly used architecture in ResNet versions. There are 4 classes of WBC subtype Eosinophil, Neutrophil, Monocyte and Lymphocyte. It is a tedious task to predict correct subtype because there are similarities in the structure. The model has 90% test accuracy and 96% train accuracy as the better one. I used Google Colab and Jupyter Notebook for developing this application. It made the work so easy and efficient. The use of Graphics Processing Unit (GPU) can accelerate the training process and yield faster results. The models were trained six to ten times for different number of epochs and finally an optimum number 20 was chosen as the number of epochs for training in the final stage. Saving the model after each epoch can help us to choose the best model based on the validation loss and validation accuracy. By this I have achieved the above-mentioned accuracy.

9. FUTURE WORK

One of the disadvantages of the application is its inability to identify other cells rather than WBC. Smear images contain all the cells but it cannot detect other cells except WBC. And also, the dataset doesn't contain any Basophil images to train the model. The future work will focus on developing a model that can give better performance on detecting and classifying Basophil, RBC, Platelet and Abnormal cells.

10. APPENDIX

10.1 Minimum Software Requirements

Operating System	:	Windows, Linux, Mac
Software	:	Jupyter Notebook, Google Colab

10.2 Minimum Hardware Requirements

Hardware capacity	:	256 GB (minimum)
RAM	:	4 GB
Processor	:	Intel Core i3 preferred
Display	:	1366 * 768

11. REFERENCES

1. https://www.tensorflow.org/
2. https://keras.io/
3. https://nodejs.dev
4. https://arxiv.org/abs/1512.03385
5. https://viso.ai/deep-learning/resnet-residual-neural-network/
6. https://towardsdatascience.com/understanding-residual-networks-9add4b664b03