BLOOD CELL IMAGE PREDICTION

1.1. INTRODUCTION:

We will create an Image Classifier of our own which can distinguish whether a given pic is a blood cell image or something else depending upon your fed data. To achieve our goal, we will use one of the famous machine learning algorithms out there which is used for Image Classification i.e. Convolutional Neural Network (or)CNN.

A possible change in the number of different leukocytes subtypes in the blood is utilized as sign for various diseases. Therefore, the counting of blood cells subtypes in the bone marrow of a patient constitutes a very informative factor in clinical practice since several blood-based diseases, infections and inflammations can often be early diagnosed by the characterization of patient blood samples. For example, patients with leukemia have often a higher level of lymphocytes due to malfunctioning of immune system and people suffering from allergies generally have an increase in their eosinophil counts.w

1.2. OBJECTIVE OF RESEARCH:

The manual approach is just as it sounds - a sample of blood is placed under a microscope and a pathologist manually counts the number of cells in each frame. The total count is then extrapolated by assuming that the distribution is uniform across the entire blood sample and multiplying up.The Machine Learning Way: Models That Improves with Time

The Machine Learning approach that we will be exploring through the rest of this blog post is a potentially promising advancement over such techniques due to a few reasons:

It requires far cheaper equipment thanks to being reliant on simple imaging.

It provides results nearly instantaneously unlike the above methods.

Like all Machine Learning, it promises to get better over time as we classify and count more and more blood cells and increase our dataset sizes. Moreover, given that it is software based, we can continuously update it over the air and provide consumers an experience that continually improve.

Instead of the image, the computer sees an array of pixels. For example, if image size is 300 x 300. In this case, the size of the array will be 300x300x3. Where 300 is width, next 300 is height and 3 is RGB channel values. The computer is assigned a value from 0 to 255 to each of these numbers. This value describes the intensity of the pixel at each point.

At the final, the image would be there in one among these four types of blood those are Lymphocytes, Eosinophil, Monocyte, and Neutrophil. If the image was one among them then it will be having a binary number identification of which that was belongs to one of them, otherwise the shape not found will be an output.

1.3. Problem Statement:

The blood cell image has to be taken and it need to be identified by the machine whether the image belongs to four of the classes of Lymphocytes, Eosinophil, Monocyte, and Neutrophil. that was also called as prediction of the image.

1.4. Industry Profile:

The layers begin to be added. This architecture was made on the principle of convolutional neural networks. It consists of 3 groups of layers, where the convolution layers (Conv 2D) alternate with the nonlinear layers (Relu) and the pooling layers (Max Pooling 2D). It then follows 2 tightly bound layers (Dense). Consider their structure in more detail.

From the view of Industry profile, we can see the in-depth view of the particular topic, such that the types of blood cells, and the problems for those blood cells and also the in-depth problems will be classified and they will be treated with the process of Image Classification.

2. Review of Literature:

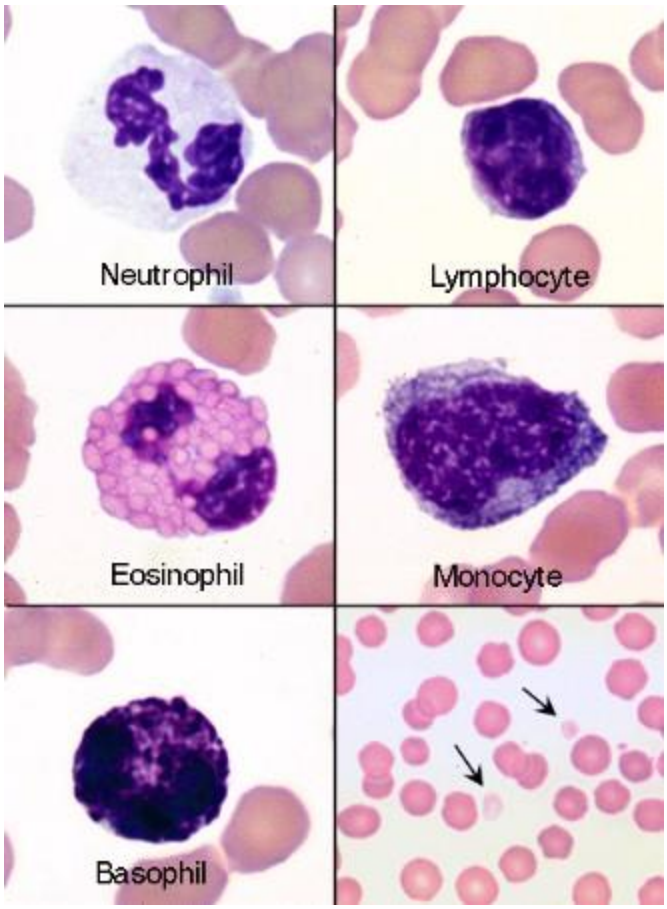
After getting the data set we need to preprocess the data a bit and provide labels to each of the image given there during training the data set. To do so we can see that name of each image of training data set is either start with blood cell so we will use that to our advantage then we use one hot encoder for machine to understand the labels(Lymphocyte[1, 0, 0 0] or Monocyte[0, 1, 0 0]).

The image is passed through a series of convolutional, nonlinear, pooling layers and fully connected layers, and then generates the output.

3.Data Collection:

For this Blood Cell Image Prediction we used images to train the model. For this, we collected images from various sites and divided the images as training set and testing set. From the total images, 80% of the images are partitioned as training set and the remaining 20% images are partitioned as test set.9A convolution neural network consists of an input and an output layer as well as multiple hidden layers. The hidden layers of a CNN(Convolution Neural Networks) typically consists of convolution layers, RELU layer i.e, activation function, pooling layers, fully connected layers and normalization layers.

The data which was collected has taken from many of the references and many of the datasets are found, and are related to this particular topic, among all those related data we had taken the dataset that was having the four of the classes of Lymphocytes, Eosinophil, Monocyte, and Neutrophil. these data will be useful for classification of our data and also used to predict the image that was taken by us and that dataset contains the data of images such are seen as below



The diagram that shows the general view of those four classes then our image that was which are taking will be having the some of the features that are to be identical and also the image has to be predicted with the classes that are having in the dataset, this is also having the advantage of finding of the image that was related to the dataset or not. If not it will be having as a output of the image was not found, the image that was not found is the output was given by the user that was user defined

A convolution neural network consists of an input and an output layer as well as multiple hidden layers. The hidden layers of a CNN (Convolution Neural Networks) typically consists of convolution layers, RELU layer i.e., activation function, pooling layers, fully connected layers and normalization layers.

4.METHODOLGY:

4.1. Exploratory Data Analysis:

ConvNets are the superheroes that took working with images in deep learning to the next level. With ConvNets, the input is a image, or more specifically, a 3D Matrix.

Let's start by looking at how a ConvNet looks!

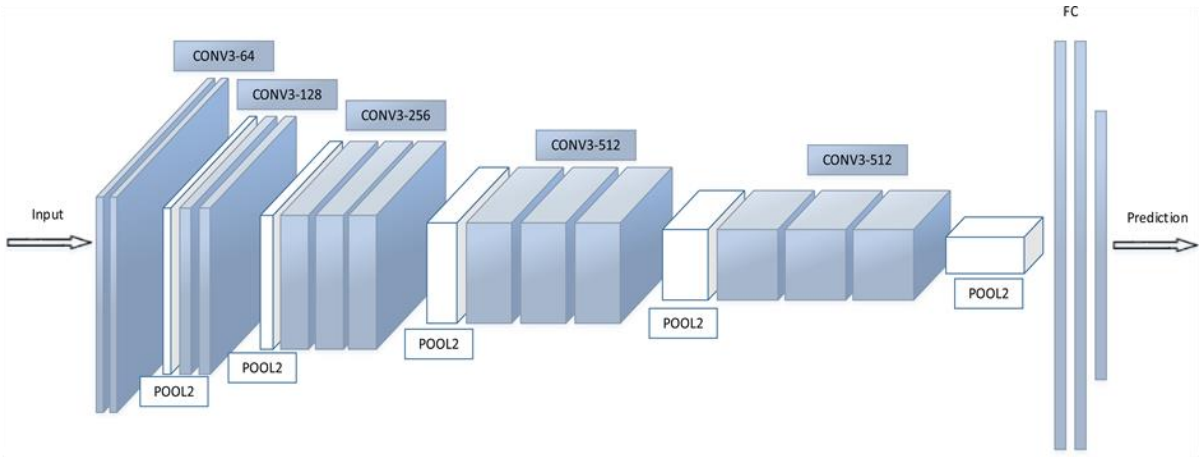


fig: Convolution Neural Network

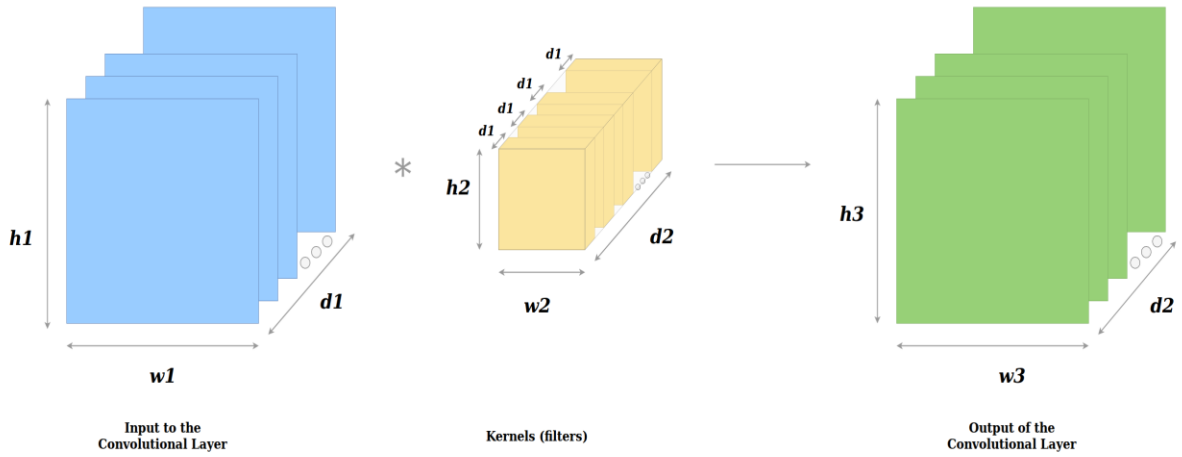
A ConvNet usually has 3 types of layers:

1. Convolution Layer(CONV)

2.Pooling Layer(POOL)

3.Fully Connected Layer(FC)

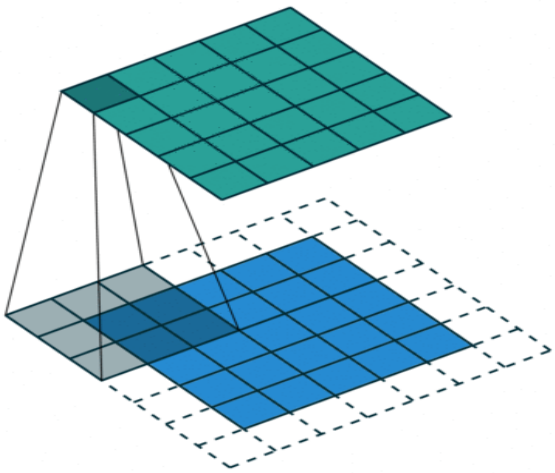
1. Convolution Layer(CONV):

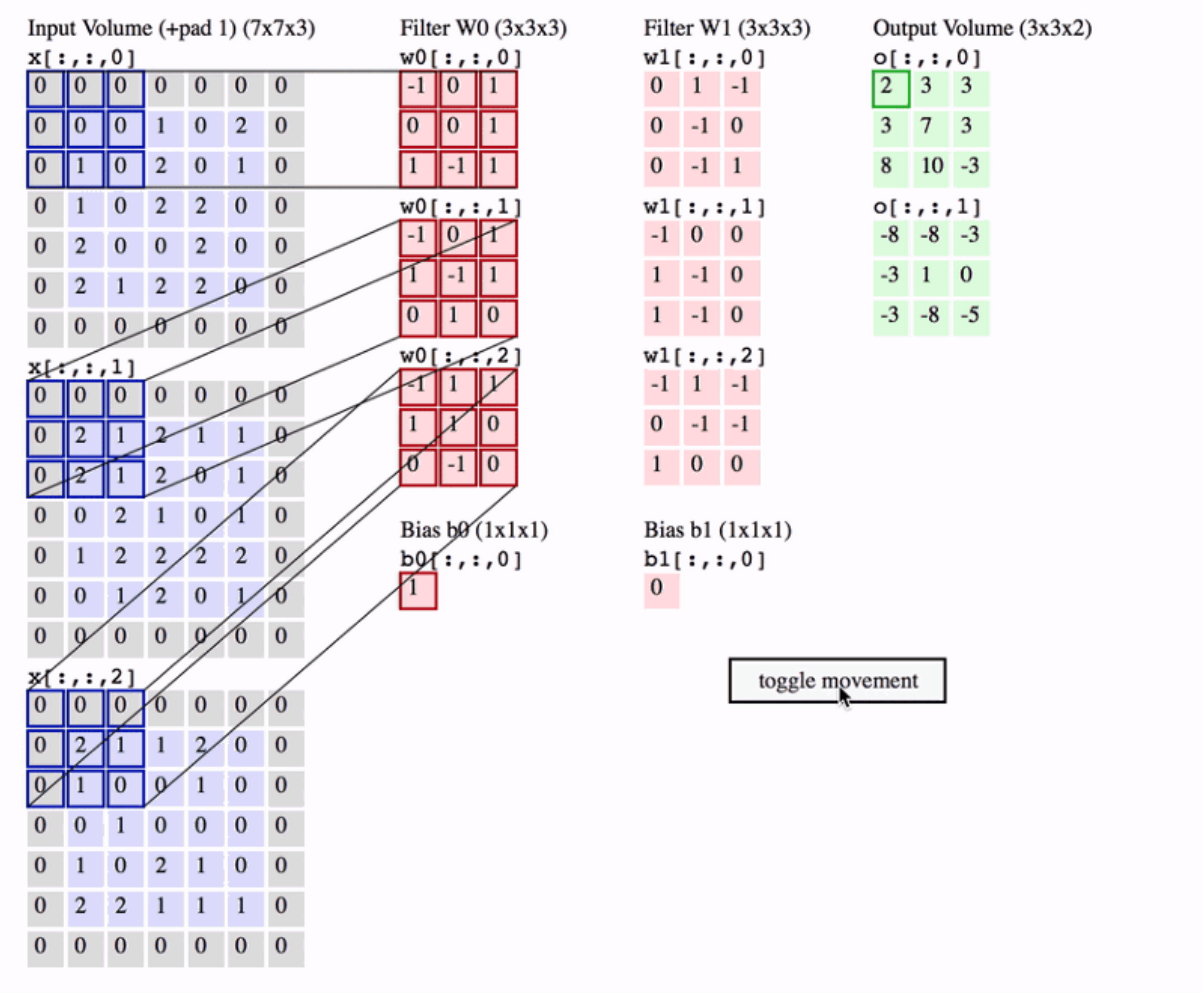


Let's throw light on some obvious things from above

1) The depth *(*d1*)* (or the number of channels) of the input and of one kernel is the same**.**
2) The depth (*d2*) of the output is equal to the number of kernels (i.e. the depth of the orange 3-dimensional matrix).

Alright, so we have inputs, kernels and outputs. Now let's look at what happens with a 2D input and a 2D kernel, i.e. *d1=1*.





2.Pooling Layer(POOL)

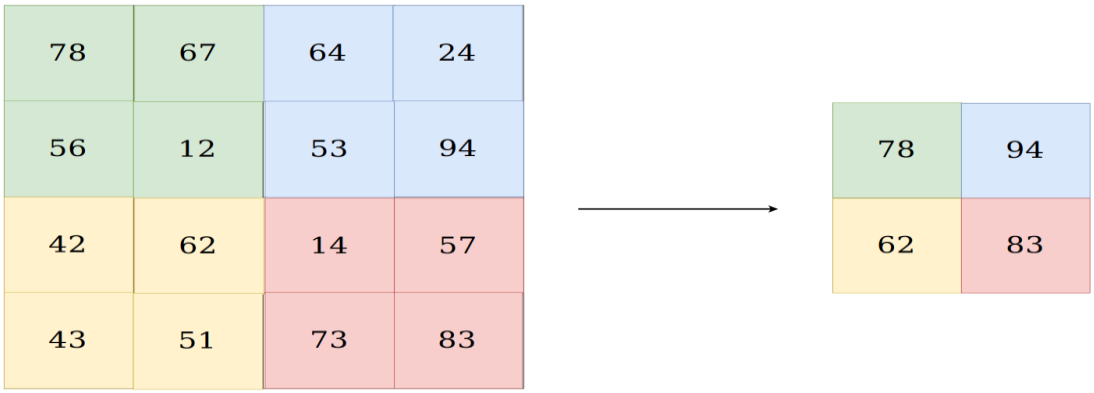


fig: Max Pooling

There are two types of pooling:

1) Max Pooling

2) Average Pooling

The main purpose of a pooling layer is to reduce the number of parameters of the input tensor and thus

- Helps reduce overfitting
- Extract representative features from the input tensor
- Reduces computation and thus aids efficiency

The input to the Pooling layer is tensor.

3.Fully Connected Layer(FC):

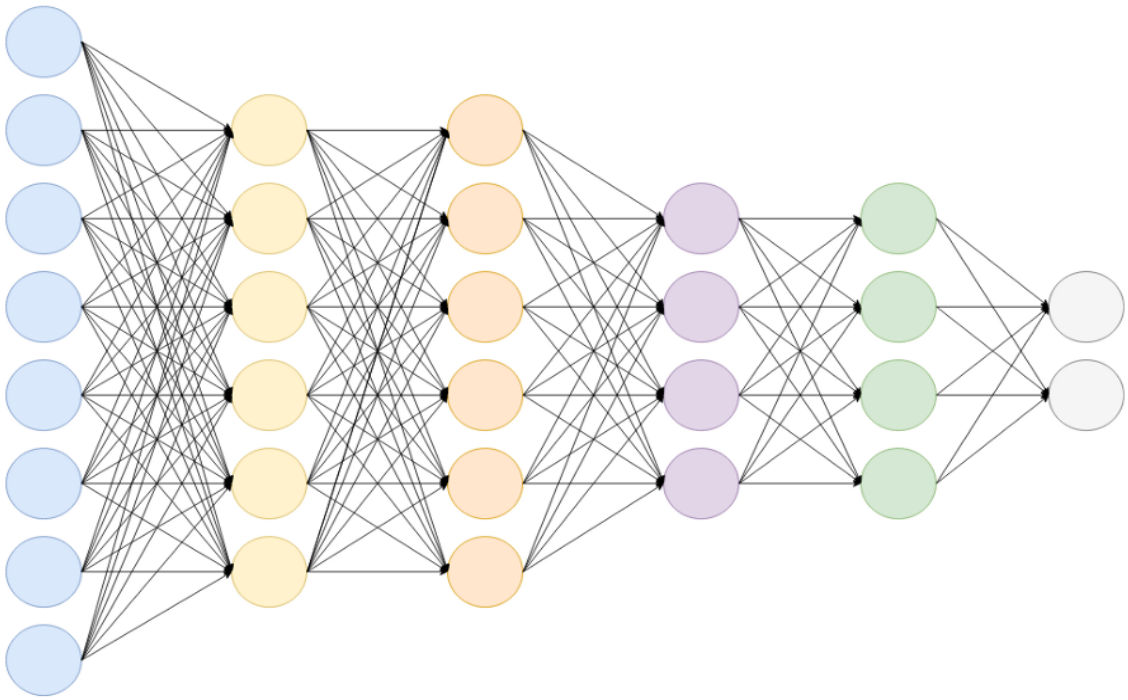


fig: Fully Connected Network

let's visualize how to calculate the dimensions of the output tensor from the input tensor.

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

where,

W1—is the width / height of the input tensor

F—is the width / height of the kernel

P—is the padding

S—is the stride

W2—is the output width / height

4.2 Statistical techniques and visualization:

NumPy stands for 'Numerical Python' or 'Numeric Python'. It is an open source module of Python which provides fast mathematical computation on arrays and matrices. Since, arrays and matrices are an essential part of the Machine Learning ecosystem, NumPy along with Machine Learning modules like Scikit-learn, Pandas, Matplotlib, TensorFlow, etc. complete the Python Machine Learning Ecosystem. NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation. NumPy can be imported into the notebook using NumPy's main object is the homogeneous multidimensional array. It is a table with same type elements, i.e, integers or string or characters (homogeneous), usually integers. In NumPy, dimensions are called axes. The number of axes is called the rank. There are several ways to create an array in NumPy like np.array, np.zeros, no.ones, etc. Each of them provides some flexibility.

Keras:

Keras is a high-level neural networks API,capable of running on top of Tensorflow, Theneo and CNTK. It enables fast experimentation through a high level, user-friendly, modular and extensible API. Keras can also be run on both CPU and GPU. Keras was developed and is maintained by Francois_Chollet and is part of the Tensorflow core, which makes it Tensorflows preferred high-level API. Keras can be installed using pip or conda Keras provides seven_different datasets, which can be loaded in using Keras directly. These include image datasets as well as a house price and a movie review datasets. To feed the images to a convolutional neural network we transform the data frame to four dimensions. This can be done using NumPys reshape method. We will also transform the data into floats and normalize it. The easiest way of creating a

model in Keras is by using the sequential API, which lets you stack one layer after the other. The problem with the sequential API is that it doesn't allow models to have multiple inputs or outputs, which are needed for some problems. Nevertheless, the sequential API is a perfect choice for most problems. To create a convolutional neural network we only need to create a sequential object and use the add function to add layers. The code above first of creates a Sequential object and adds a few convolutional, max pooling and dropout layers. It then flattens the output and passes it two a last dense and dropout layer before passing it to our output layer. If you aren't confident build a convolutional neural network(CNN) check out this great tutorial. The sequential API also supports another syntax where the layers are passed to the constructor directly. Alternatively, the functional API allows you to create the same models but offers you more flexibility at the cost of simplicity and readability. It can be used with multiple input and output layers as well as shared layers, which enables you to build really complex network structures. When using the functional API we always need to pass the previous layer to the current layer. It also requires the use of an input layer. Before we can start training our model we need to configure the learning process. For this, we need to specify an optimizer, a loss function and optionally some metrics like accuracy. The loss function is a measure on how good our model is at achieving the given objective. An optimizer is used to minimize the loss(objective) function by updating the weights using the gradients. Augmentation is a process of creating more data from existing once. For images you can to little transformations like rotating the image, zooming into the image, adding noise and many more. This helps to make the model more robust and solves the problem of having not enough data. Keras has a method called Image Data Generator which can be used for augmenting images. This Image Data Generator will create new images that have been rotated, zoomed in or out, and shifted in width and height. Now that we defined and compiled our model it's ready for training. To train a model we would normally use the fit method but because we are using a datagenerator we will use fit_generator and pass it our generator, X data, y data as well as the number of epochs and the batch size. We will also pass it a validation set so we can monitor the loss and accuracy on both sets as well as steps per epoch which is required when using a generator and is just set to the length of the training set divided by the batch size. We can visualize our training and testing accuracy and loss for each epoch so we can get intuition about the performance of our model. The accuracy and loss over epochs are saved in the history variable we got whilst training and we will use Matplotlib to visualize this data.

OpenCV (*Open source computer vision*) is a library of programming functions mainly aimed at real-time computerized vision.Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross platformand free for use under the open-source BSD license.

OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe.

4.3 Data Modeling and visualization

Step 1—Collecting the Dataset

In order to train our machine, we need a huge amount of data so that our model can learn from them by identifying out certain relations and common features related to the objects. Fortunately many such datasets are available on internet. Here is a link for the helmets and without helmets dataset which consist of 500 images—250 of each. This will help in training as well testing our classifier.

Step 2—Importing Libraries and Splitting the Dataset

To use the powers of the libraries, we first need to import them After importing the libraries, we need to split our data into two parts- training set and test set. In our case, the dataset is already split into two parts. The training set has 400 images each of helmets and without helmets while the test set has 50 images of each.

Step 3—Building the CNN

This is most important step for our network. It consists of three parts -

1. Convolution
2. Polling
3. Flattening

Step 4—Full Connection

Full connection is connecting our convolutional network to a neural network and then compiling our network. Here we have made 2 layer neural network with a sigmoid function as an activation function for the last layer as we need to find the probability of the object being a helmet or a without helmet.

Step 5—Data Augmentation

While training your data, you need a lot of data to train upon. Suppose we have a limited number of images for our network. What to do now? You don't need to hunt for novel new images that can be added to your dataset. Why? Because, neural networks aren't smart to begin with. For instance, a poorly trained neural network would think that these three tennis balls shown below, are distinct, unique images. So, to get more data, we just need to make minor alterations to our existing dataset. Minor changes such as flips or translations or rotations. Our neural network would think these are distinct images anyway. Data augmentation is a way we can reduce overfitting on models, where we increase the amount of training data using information only in our training data. The field of data augmentation is not new, and in fact, various data augmentation techniques have been applied to specific problems.

Step 6--Training our Network

So, we completed all the steps of construction and its time to train our model. If you are training with a good video card with enough RAM (like an Nvidia GeForce GTX 980 Ti or better), this will be done in less than an hour. If you are training with a normal CPU, it might take a lot longer. With increasing number of epochs, the accuracy will increase.

Step 7--Testing

Now let's test a random image. And, yes !!our network correctly predicted the image of the blood cell!! Though it is not 100% accurate but it will give correct predictions most of the times. Try adding more convolutional and pooling layers, play with the number of nodes and epochs, and you might get high accuracy result. You can even try it with your own image and see what it predicts. Whether you look close to a helmet or a without helmet. Now let's test a random image.And, yes !! our network correctly predicted the image of the blood cell!! Though it is not 100% accurate but it will give correct predictions most of the times. Try adding more convolutional and pooling layers, play with the number of nodes and epochs, and you might get high accuracy result. You can even try it with your own image and see what it predicts. Whether you look close to a blood cell.

5. FINDINGS AND SOLUTION:

From our project we can predict the image of blood cell subtype such as from the Lymphocytes, Eosinophil, Monocyte, and Neutrophil the data that was which obtained from the datasets, the main aim of this solution was at last the end user should be get satisfied with the solution of the prediction of the cell that was obtained when the user has chosen an image to predict with the other four cells. 21.This was the solution that was found out when the user has wanted to predict the data he wanted.

6 .CONCLUSION:

The project that was done with the correct solution that which all the prediction of are taken and then the data that would be similar to one of those four classes if the image was not one of them it would be the output of shape not found or the user preference .