# Chapter 23: Application of Deep Learning in Counting WBCs, RBCs And Blood Platelets Using Faster Region-based Convolutional Neural Network

Authors:
Nirav Jain (D.J. Sanghvi College of Engineering, niravjain1709@gmail.com )
Shail Shah (D.J. Sanghvi College of Engineering, shail.shah822@gmail.com )
Prof. Dr. Ramchandra Mangrulkar (D.J. Sanghvi College of Engineering, ramchandra.mangrulkar@djsce.ac.in )
Prof. Pankaj Sonawane (D.J. Sanghvi College of Engineering, pankaj.sonawane@djsce.ac.in )

## Abstract

The human body is comprised of many different elements that help in the proper functioning of the body. One such element is the blood. It contains about 7 percent of the human body by weight and controls and co-ordinates many functions of the body. It comprises different types of cells, namely, RBCs (Red Blood Cells), WBCs (White Blood Cells) and blood platelets, which helps in the body's proper functioning. Complete Blood Count (CBC) tests are the ones that are most commonly prescribed by doctors whenever the body faces an illness. CBC tests are used to count the number of WBC's, RBC's, and Platelets in the blood. The traditional methods for performing CBC tests are time-consuming and costly. The two methods used for CBC tests nowadays include "Coulter Counters and Laser Flow Cytometry" (automated machine-based counting) and the "Manual Way" (Manual counting). What if these tests can be conducted as quickly and as efficiently as possible? What if all the costly and heavy machinery is replaced by cheap and quick computer technology? This chapter proposes a method to count the number of WBC's, RBC's, and Platelets using a deep learning method called Faster R-CNN. The proposed methodology uses a dataset of microscopic images of blood cells which have been annotated and augmented for RBC's, WBC's, and blood platelets. This method uses these images to train the model, and then the model identifies the individual number of WBC's, RBC's and Platelets in the given images (i.e., in a single

microscopic frame), which can then be extrapolated to get the final blood count.

## 23.1 Introduction

Since the last few decades, Neural Networks have become quite popular due to the advancements in deep learning. This popularity has led to many new and robust algorithms in the field of computer vision. New doors have been opened, and new paths have been laid down for further advancement in this field. This technology is finding its way into many real-life applications. One such application that this chapter will be discussing is the Complete Blood Count Test using Faster R-CNN.

In clinical pathology tests such as Complete Blood Count (CBC), the precise counting of blood cells is very crucial. Such tests have been known to be of immense importance in the diagnosis of a large number of diseases such as infections, inflammations, dengue, malaria, and even deadly diseases like cancer, leukemia (Daqqa and Sarraj 2017, 638-643), Bone marrow failure, etc. Due to its application in a large number of essential areas, automating or, more precisely, speeding up the process of CBC tests seems to be of immense importance (Cruz et al 2017) (Habibzadeh and Fevens 2013, 263-274). The WBC count in the blood directly reflects the strength of an immune system. They are considered to be the defense blocks of the body. An enormous change in WBC count can affect the body in a negative way. A very high WBC count occurs when the body tries to fight a disease, causing problems like fevers, night sweats, etc. In contrast, a lower WBC count indicates a lower number of defense blocks to defend you against the invading disease, making you an easy target for such diseases.

Traditionally techniques such as "Coulter Counters and Laser Flow Cytometry" were used, which required a very complicated, costly, and time-consuming system for performing the CBC test. Modern techniques such as Automated Hematology Counter are very expensive to be used in rural areas. Automated test for CBC count uses the latest bioengineering techniques to give an accurate and faster result compared to the Manual CBC test. Sysmex XT-2000i is the latest machine used for calculating the CBC count. It is the successor of the famous XE-2100. The Sysmex XT-2100i examines blood samples at the rate of 80 samples/hour. Also, counting cells using image processing (Kaur and Garg 2016, 2574-2577) (Meimban et al 2018, 50-53) alone is not feasible as it fails to generalize the model. Hence an effective replacement of the above methods is required.

The blood count is the first step towards discovering a disease or any unnatural behavior by the body. The blood consists of three major types of cells: Red Blood Cells (RBCs) or erythrocytes, White Blood Cells (WBCs) or leukocytes, blood platelets, or thrombocytes**.** With the CBC count, the disease is diagnosed, and then the doctors take further actions accordingly. The CBC count is required during a patient's treatment to monitor the progress and the condition of the patient (George-Gay and Parker 2003, 96-114) (Meintker and Krause 2013, 641-650).

A standard CBC count contains the following information: -

1. Red Blood Cells Test -
    - A. The number of RBCs in the blood sample
    - B. The amount of Hemoglobin present in the blood sample
    - C. The percentage of volume of RBC in total volume of blood sample (Hematocrit)
    - D. The measurement of the mean size of the RBC known as Mean corpuscular volume
    - E. The calculated average of Hemoglobin known as Mean corpuscular Hemoglobin
    - F. The Hemoglobin concentration

   G. The distribution of the RBCs
   H. The measurement of young RBCs in the blood sample.

2. White Blood Cells Test -
   A. The number of WBCs in the blood sample
   B. Depending on the number of WBCs in the sample, further count of individual cells is required
3. Blood Platelets Test -
   A. The number of blood platelets in the blood sample
   B. The average size of the blood platelets known as Mean platelet volume
   C. The distribution of the blood platelets in the blood.

There are many ways available currently to calculate CBC count, but are slow and others are rather too expensive for the common people. There are two ways of conducting CBC count, Automated and Manual. Both these methods have their share of disadvantages. For instance, the machines used for automated counting are very expensive. Also, the minute variations in the shape of the cells, irregularities are not detected.

On the other hand, manual techniques are time-consuming wherein a pathologist counts the number of cells under a microscope and approximates them for the total count. The Hemocytometer is used to perform blood calculations manually in the lab. It requires a trained professional to check the blood sample under a microscope and deliver accurate results. It requires more time than the automatic system as a person is involved, and thus the throughput reduces considerably. By manual test, a rate of 3-4 samples/hour is achieved depending on the individual's speed. Flagging of a laboratory test result demands a labor-intensive manual examination of a blood smear. Also, the comments on red cell morphology cannot be created when manual testing is used. Platelet lumps are counted as one resulting in low platelet counts. Consequently, the results are error-prone. Furthermore, it is expensive. Hence an efficient solution to the problem is desired.

The work presented uses the power of a Convolutional Neural Network to develop the model to count the blood cells. It uses an optimized version of CNN, known as the Faster R-CNN, which processes the images in a fast and efficient manner. So, this chapter first discusses how deep learning algorithms can be used to solve the discussed problem. The chapter then explains each of the algorithms, namely, R-CNN, Fast R-CNN, and Faster R-CNN, in brief. It then explores the type of algorithm suitable for our problem and the various aspects of data collection, data preprocessing, creating the model, training, testing, and evaluating our models, which are the steps required for successful training of a deep learning model. The chapter then discusses the solution to the various problems that are faced during or after training the model. The chapter provides a complete methodology to accomplish the above steps. Finally, the chapter discusses the viability of the above solution in the real world.

## 23.2 Literature Review

In their paper "Classification of Leukocyte Images Using K-Means Clustering Based on Geometry Features" (Rosyadi et al 2016, 50-53) proposed a model to classify the WBCs from output image of an optical microscope. Initially, the leukocyte images were classified into their own clusters by using K-Means clustering. Neutrophil, Lymphocyte, Monocyte, and Eosinophil are the WBCs that can be classified using the given model. The classification is performed in steps, which are image preprocessing, leukocyte segmentation, feature extraction and classification. This model was tried with various features to test their accuracy and include them for the final classification. The study found that the highest accuracy from test images was of the 'circularity' (67%) feature, while the lowest feature accuracy was that of the 'eccentricity' (43%) feature. The conclusion obtained from the following study was that the accuracy of the model does not depend on the number of features but the characteristic of the feature to distinguish between different leukocyte.

DyaneshVarun D., Kandluri Reddy, Lakshmi Priya S., and Vasavi Abirami S. in their paper "Blood Cell Count using Digital Image Processing" [2] proposed a method to count RBCs and WBCs from the smear image of blood taken by a compound microscope. The steps involved to count the cells are Plane Extraction of the images, Edge detection to find edges in the image, and morphological filling operation to fill an image's shape. RBCs are counted via the Circular Hough transform, while WBCs are counted by detecting their boundary. The accuracy of counting RBCs from this model is 91% and that of counting WBCs is 85%. This method is tested with different datasets and is proven to be efficient and accurate. Also, the model can further be improved to count the platelets and the different types of WBCs.

Alam, Mohammad Mahmudul, and Mohammad Tariqul Islam in their paper "Machine learning approach of automatic identification and counting of blood cells" [3] proposed a model to count and identify blood cells using YOLO (You Only Look Once) object detection and classification algorithm to automate Complete Blood Count (CBC) test. The author proposes this model to overcome the challenges faced by the traditional methods of manual counting using a hemocytometer, which is both tedious and time-consuming. The author uses YOLO in combination with KNN and different correction methods like IOU to get accurate results. The drawbacks of YOLO, i.e., the difficulty in detecting crowded and small images, have been reduced by using KNN and IOU with a predefined threshold. In this paper, the author has proposed a fast and efficient way to automate blood cell counting.

Najmeddine Dhieb, Hakim Ghazzai, Hichem Besbes, and Yehia Massoud University of Carthage, Higher School of Communications of Tunis, Tunisia in their paper "An Automated Blood Cells Counting and Classification Framework using Mask R-CNN Deep Learning Model" [4] have proposed a CNN based model to classify and count RBC's and WBC's. They have proposed a method that uses mask R-CNN along with instance segmentation and transfer-learning to identify blood cells at a pixel level. Their methodology has the ability to identify overlapped and faded cells too. They have proposed the use of RESNET-101 as the backbone for feature pyramid network and MS-COCO pre-trained model to initiate model weights of the neural network. This paper also implements data augmentation and regulation techniques to improve the performance of the model. The author advocates the use of mask R-CNN not only for its accurate counting but also for its ability to distinguish between different types of cells through instance segmentation.

## 23.2 Convolutional Neural Network (CNN):

Convolutional Neural Network is a part of Deep Neural Network, which is mainly aimed at developing machine learning models on visual imagery. Also referred to as the shift invariant artificial neural network (SIANN), due to a shared weight architecture. Convolutional Neural Network is a concept developed by comparison to a biological process, and the connectivity resembles that of animal visual cortex. CNN is a fully connected network of nodes, which means every node is connected to every other node in the next layer. CNN has different layers which is the key to identifying the distinguishing features in the images. The completeness of the networks makes it less susceptible to overfitting of data. CNN uses a special type of linear operation, named convolution instead of traditional matrix multiplication in at least one of their layers (Krizhevsky and Hinton 2012, 1097-1105).

Insert Chp23_Fig01 here

### *23.2.1 Activation Function:*

Convolutional Neural Network unlike other deep learning models does not only use activation functions like

Tanh, sigmoid, ReLU or Leaky ReLU

But the convolutional layers or the hidden layers in the CNN also uses two different functions, namely

- Convolution Function
- Pooling Function

### *23.2.2 Convolutional Function:*

Convolutional Function is a linear function used as an activation function in CNN. It takes in two tensors (images) and returns a third image. It can be viewed as application of a filter to the image in order to generate a new image.

The second tensor is known as kernel, which is generally smaller as compared to the Input image. The kernel moves across all of the image and generates a resultant image based on the dot product operation.

The mathematical formula for the Convolutional Function can be as follows:

$$(f * g)(i) = \sum_{j=1}^{m} \blacksquare\, g(j).f(i-j+m/2)$$

The image below depicts the working of a Convolution Function applied on an image.

Insert Chp23_Fig02 here

***23.2.3 Pooling Function:***

Pooling is used to make assumptions regarding the features of the input image, based on sample discretization. It is another type of function used in the Convolutional neural network which is majorly used to decrease the dimensionality of the input matrix, which means if the input matrix is n x n, the pooling will decrease it to m x m where m < n.

There are two types of pooling

- Max Pooling - The maximum element is taken from the matrix of kernel size and is added to the Output image. In the image below the kernel size is 4.

  Insert Chp23_Fig03 here

- Min Pooling - The minimum element is taken from the matrix of kernel size and is added to the Output image. In the image below the kernel size is 4.

  Insert Chp23_Fig04 here

***23.2.4 Fully Connected Layer:***

The fully connected layer takes the output of the last pooling or convolutional layer as an input and then flattens the resulting tensor into a single vector, so as to ease the process of training for the further neural network. The result is then fed to the Fully Connected Layers.

The word Flatten means the final matrix is reduced from a 3-dimensional matrix to a 1-

dimensional vector.

Insert Chp23_Fig05 here

The final few Fully Connected networks behave as an Artificial Neural Network and have the same functionality as that of an ANN.

Each Layer calculates the following:

$$g(Wx + b)$$

where,

| | |
|---|---|
| x | - is the input vector with dimension [m,1] |
| W | - is the weight matrix with dimension [m,n] where m is no.of neurons in previous layer and n is no.of neurons in the next layer |
| b | - is the bias vector with dimension [m,1] |
| g | - is the activation function (Usually ReLU) |

## 23.3 Region-based Convolutional Neural Network (RCNN):

Region-based CNNs or regions having CNN features is a modified machine learning algorithm which aims at detecting multiple objects in a single image. RCNN divides the whole image into Regions based on a selective search algorithm, which is a fixed algorithm and it does not learn or adapt itself (Zhang et al, 2020) (Weng, 2017).

The RCNN algorithm follows the given steps in order to classify data and detect object:

1. Divide the Image into Regions with help of the Selective search algorithm.

2. A pre-trained CNN is then applied on the output of the Selective Search algorithm and the CNN converts the region into suitable form for further processing. Then it generates an output of classified features of the regions.
3. Results from Several regions are combined together to train the Support Vector Machine for classification of Objects, where a certain SVM decides whether an object belongs in a certain category or not.
4. All the features and bounding box labels of all the regions are combined in order to train a linear regression model.

### ***23.3.1 RCNN Architecture -***

Insert Chp23_Fig06 here

There are various algorithms used in order to identify the regions optimally and increasing the accuracy of R-CNN, namely:

1. Non-Max Suppression: - Non-Max Suppression deals with the extra bounding boxes associated with the same object. It follows a *greedy* approach and recursively sorts the bounding boxes on basis of the confidence scores and discards the one with the lowest score. It ignores the boxes with high IOU (> 0.5) with the selected box. This helps in removal of excess bounding boxes for the same object.
2. Hard Negative: - The regions with no object that is the region possessing the background, noise and other parts except object are termed as negative regions. Some of these negative regions are easy to classify, like the region containing the background, and thus are known as "easy negatives". While other negatives, which contain some part of the object and noise are classified as "hard negatives". The "hard negatives", usually decrease the accuracy of the model and thus after classifying them, they can be used to train and

evaluate the model in order to decrease the false positives, and thus increase the accuracy of the model.

**23.4 Fast RCNN:**

The major drawback of Region based Convolutional Neural Network (RCNN), is that it is a very slow approach as 2000 regions are passed on to the CNNs for feature extraction and classifying the image. To overcome this drawback, a faster algorithm known as Fast RCNN was developed. Fast RCNN decreases the computational time to a very great extent. Instead of feeding the regions to the CNN (in case of RCNN), the input image itself is fed to the CNN. This way it saves a huge amount of computational time (Girshick 2015, 1440-1448).

The steps involved in the process of object detection with Fast RCNN are:

1. The input image (size independent) is fed to a Convolutional Neural Network.
2. The CNN extracts the Convolutional Feature Map from the input image, and extracts Region of Interests or ROI from the feature map.
3. Then the regions obtained from the feature map are passed on to the ROI pooling layer which resizes them to a constant size.
4. The output of the ROI pooling layer needs to be of the same size so as to transfer the output to the fully connected layer for further processing.
5. The SoftMax classifier along with a linear regression is used to classify the final output obtained.

Note that in Fast R-CNN the input image is fed to the network unlike R-CNN where the region proposals are fed to the network.

Insert Chp23_Fig07 here

## 23.5 Faster-RCNN:

Faster-RCNN is an improvised version of the Fast-RCNN. It uses Region Proposal Network (RPN) for the selection of the Regions instead of the selective search algorithm. The RPN decreases the amount of computation considerably and is thus used in place of the selective search algorithm (Ren and Sun 2015, 1137-1149).

### *23.5.1 Region Proposal Network (RPN):*

It is an intelligent substitute to the naiver selective search algorithm. It uses a CNN which is used to recognize regions. It not only just decreases the per image time from 2s (in case of selective search) to 10ms in RPN, but also shares some features with the detection stage which improves the efficiency of the overall algorithm.

The steps involved in RPN are as follows:

1. Input image is resized to 600 x 1000 dimension and then it is forwarded to the backbone CNN (in this chapter usually VGG and ZF-net) which gives an output in the form of HxW, which is smaller than the original image and depends generally on the stride used in the backbone CNN (16 in the proposed methodology).
2. RPN uses a concept of Anchors, which are boxes of different scales and aspect ratios. They are used to predict the presence and location of an object in the image.
3. Every output pixel of the feature map consists of 9 anchor boxes. These Anchor boxes each contain a prediction indicating whether an object is present or not (foreground or background) and a prediction of the anchor boxes' co-ordinates, which are height, width and center (h, w, x, y).
4. The last part of the RPN is the predictor. There are two types of predictor used in the RPN, one is the Classifier which is used to predict whether an object is present in the

image or not and the second is the Regressor used to predict the co-ordinates of the anchors used.

5. The output of the RPN contains both, the classification and the corresponding bounding box co-ordinates, which are then forwarded to RoI pooling layer for further processing.

Insert Chp23_Fig08 here

The output of the RPN is the prediction whether the anchor box, having co-ordinates given by the regressor (output of RPN), contains an object or not.

### *23.5.2 Faster-RCNN Architecture:*

The steps involved in Object Detection by Faster-RCNN are:

1. Input Image is passed through a Convolutional Neural Network, which returns the feature map of the image.
2. The feature map is then passed to the RPN, to get the region proposals for further processing.
3. The RPN processes the Feature Map and returns the co-ordinates of the proposal and whether it contains the object.
4. The output of the RPN is then passed to the RoI pooling layer, which converts all the proposals to the same size, so as to make it feasible for the fully connected network.
5. The output of the RoI pooling is passed on to the fully connected layer.
6. The output of the fully connected network is then passed onto the softmax layer to classify the proposals as objects of different classes. Also, a regressor is used to improve the co-ordinates of the bounding boxes of the proposals.

Insert Chp23_Fig09 here

## 23.6 Implementation

As the discussion the problem, the different types of CNNs that can be used and why the proposed methodology uses Faster R-CNN, lets dive straight into its implementation. This chapter won't be going into too much detail about how Faster R-CNN is actually implemented, rather this work will be using already implemented Faster R-CNN architecture (in TensorFlow) by industry specialists. This section of the chapter gives detailed step-by-step instructions on how to use TensorFlow Object Detection API for training the model.

Before starting with the implementation let's first understand what is TensorFlow Object Detection API. It is an open source framework built by Google Brain, a deep learning artificial intelligence research team at Google, for making the tedious task of object detection easier. It is built on top of TensorFlow (python library for deep learning) and provides a lot of easy-to-use tools to implement deep learning models without the need of writing huge amounts of code. It makes the task of constructing, training and deploying new models way easier than implementing it from scratch. Also, it is implemented by industry specialists and is open source, hence the chances of getting an error are way too low. Even if there are any errors the open source community is always there to help. It supports a variety of CNN architectures like Faster-RCNN, SSD, etc. Still it is highly recommended that the reader implements the Faster R-CNN architecture (or at least a basic implementation of the architecture) if he/she understands it completely and has the required knowledge. So, let's start with the implementation.

### *23.6.1 STEPS FOR IMPLEMENTATION*

- **Step 1: Collecting the dataset**

  Download the dataset from https://github.com/MahmudulAlam/Complete-Blood-Cell-Count-Dataset.git prepared for the paper "Machine learning approach of automatic identification and counting of blood cells" (Mahmudul and Islam 2019, 103-108). This dataset consists images that have been hand labeled and annotated and each image is of size 640*480. The training set consists of 300 images whereas the validation as well as testing set consists of 60 images. As the dataset is very small, this chapter uses a subset of training data for validation.

- **Step 2: Cleaning the dataset**

  As always, the first step after collecting the dataset is to clean the dataset according to the problems needs. The first step, as always, is cleaning the dataset by quickly checking the bounding boxes and the distribution of RBCs, WBCs and Blood Platelets. The dataset used in this chapter is already clean to a great extent. The only thing left is to augment the dataset. This is necessary as the dataset used for training is very limited. As the orientation of the image does not matter for the problem being discussed, dataset is augmented by flipping the images in every possible direction. Thus, the size of the training data is increased from 300 images to 1200 images. Other augmentations such as changing brightness levels, contrasts, blurs etc. can also be applied.

- **Step 3: Using Google Colab**

  Create a new notebook on google colab as shown in the below figure

  Insert Chp23_Fig10 here

- **Step 4: Folder Structure**

  The folder structure this chapter uses is as follows (of course you can choose any structure you are comfortable with):

  Insert Chp23_Fig11 here

- **Step 5: Setting Up the Data**

As seen in the folder structure the data will be split in three ways i.e. Training, Testing and Validation.

Training - 300 Images and their Annotations for training

Testing - 60 Images and their Annotations for testing

Validation - 60 Images and their Annotations for testing as raw data

Since the number of Images for training are only 300, the data is augmented by flipping it into the remaining 3 directions and changing the augmentations accordingly which is done in the data_gen.py.

This file includes the python script to flip and mirror the images and then create corresponding annotations so that there are 4 images for every single image. Thus, the total number of training images becomes 1200.

The model's folder stores the model checkpoints. It is empty initially.

- **Step 6: Understanding the different scripts and files required by TensorFlow Object Detection API**

    1. **label_map.pbtxt**

       This is a simple text file containing mapping between classes and integers. It maps each and every prediction class to an integer in a dictionary format. This file is used by the API for both training and detection purposes.

    2. **pipeline.config**

       This file contains the various hyperparameters and paths to various folders and files required for training. It contains hyperparameters such as the initial learning rate, batch_size, size_of_anchor_boxes, etc. and paths to different record files, label_map files, etc. It is also the place where you specify your pretrained-model weights. So basically, this file is the soul of your model training and all the tuning that you need to do on your model can be done here. The proposed work uses Resnet_101 pre-trained model for training.

3. **train.csv and test.csv**

   These files contain complete information of the training and testing data. They contain the images to be trained and information about each and every labelled bonding box the image has i.e. it contains the following columns (filename, width, height, class, xmin, ymin, xmax, ymax) as shown in the figure below.

   Insert Chp23_Fig12 here

   These files are generated automatically using the script xml2csv.py which converts the xml (VOC) files i.e. the annotation files into corresponding rows in the csv. But the data might not always be in xml format. So whatever format the data might be in you either convert it directly to a csv file or you can convert it into xml format and then use the above script file. For e.g., say the data is in yolo format, so you first convert the annotations to voc using the script yolo2voc.py and then to csv using xml2csv.py. These are basic python scripts and do not require any special knowledge.

4. **train.record and test.record**

   The record files are used to store the data in the csv files into a format suitable for training by the object detection api. It contains the complete dataset in one file i.e the images and their annotations all at one place. The only reason for creating the csv files in this implementation is to convert them to record files using the script in gen_tfrecords.py file. Thus, two record files are generated, one for training data and the other for testing data. If you are not comfortable with writing you own script for generating tfrecords, the object detection api has several example scripts to generate tfrecord inside the folder "models/ research/ object_detection/ dataset_tools/" which you can edit and use.

- **Step 7: Installing TensorFlow Object Detection API on colab**

  Follow the steps mentioned on the official GitHub page of TensorFlow object detection api to install it correctly.

- **Step 8: Setting up tensorboard (python library to visualize your results)**

  1. **Load tensorboard**

     ```
     %load_ext tensorboard
     ```

  2. **Give tensorboard the path where you store your model's checkpoints to start it in Google Colab's cell.**

     ```
     %tensorboard --
     logdir "/content/gdrive/My Drive/tf_object_detection/Projects/CRC_Press
     /models"
     ```

     Output:

     Insert Chp23_Fig13 here

     This board is a great tool to visualize your results.

     a. The Scalars tab shows different types of graphs related to your model's performance like precision, recall and loss.

     b. The Images tab is used for visualizing your predictions on the validation set.

     c. You can use the graphs tab to visualize your whole TensorFlow Graph.

d. You can't view Projector Tab in Colab as of now but it is used to graphically represent high dimensional embeddings.

All these results can be seen once the events file for tensorboard have been created i.e. once you begin your training.

- **Step 9: Actual Training**

Run the following command to start the actual training

```
!python object_detection/model_main.py \
    --pipeline_config_path=
    /content/gdrive/My\ Drive/tf_object_detection/Projects/CRC_Press/pi
    peline.config \
    --model_dir=
    /content/gdrive/My\ Drive/tf_object_detection/Projects/CRC_Press/mo
    dels \
    --num_train_steps=5000 \
    --sample_1_of_n_eval_examples=1 \
    --alsologtostderr
```

**Key Terms**:

**pieline_config_path**: path to the config file of your model (i.e. pipeline.config file)

**model_dir**: path where the model checkpoints are to be saved. Basically it is the path to your output directory (i.e. models directory)

**num_train_steps**: Number of epochs for which you want to train your model

**sample_1_of_n_eval_examples**: As the name suggests it will sample 1 of every n input examples. If set to 1 it will sample all the eval examples

**alsologtostderr**: It simple sends the logs to STDERR standard file

As the training continues, you can see that the output of the cell shows a lot of lines. Well no need to worry! This chapter will go through the important ones here.

**1.**

Insert Chp23_Fig14 here

a. The global_step here denotes the number of batches seen by the graph. Therefore, global_step/sec simply refers to the number of batches seen by the graph per second.

b. Step simply refers to the epoch number

c. Loss as the name suggests is the total loss of the model and should be as low as possible.

**2.**

Insert Chp23_Fig15 here

The above images help in the calculation of mean average precision (mAP). Now to calculate mAP it is necessary to know a few things. So, let's start wth "maxDets". "maxDets" in the above figure simply means the maximum number of detections with the highest scores. The second important term is IoU (Intersection over Union). IoU simply suggests the correction of each bounding box. As the name suggests it is the ratio of the area of intersection to the area of union between the ground truth bounding box and the predicted bounding box.

Insert Chp23_Fig16 here

The area of intersection is the common area between the two bounding boxes whereas the area of union is the total area covered by the two bounding boxes. Now to calculate precision or recall a few metrics like true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) are required. Now a detection is said to be true positive if it's IoU is above a specified threshold else it is true negative. The most commonly used threshold is 0.5. True negatives simply refer to every part of the image where there are no bounding boxes and False negatives are simply those parts of the image where the model misses out the detections. Now as all the 4 required variables to calculate precision and recall are available, the following formulae to calculate precision and recall for the detections can be used.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

In order to calculate mAP, as discussed in the paper "The PASCAL Visual Object Classes (VOC) Challenge" (Everingham and Zisserman 2009, 303-338), Precision for different model thresholds where the recall normally has the values 0, 0.1, 0.2 …. 1.0 is calculated i.e. 11 precision values for 11 chosen thresholds which gives us an overall review of the precision recall curve are calculated.  The mAP hence is the Mean of all the Average Precision values across all your classes as measured above. In short mAP is an approximation to the area under the precision recall curve.

The model has been trained for only 20000 steps, which you can, of course, change for better accuracy but you need to beware of overfitting. Normally a good idea when you start your model training is to view the different graphs on the tensorboard which helps us in determining problems, if any, at the earlier steps of training.

The various graphs that can be visualized in tensorboard are as follows:

1. **Graphs of precision vs steps**

   Insert Chp23_Fig17 here

2. **Graphs of recall vs steps**

   Insert Chp23_Fig18 here

3. **Graphs for global_norm and global_steps**

   Insert Chp23_Fig19 here

4. **Graph for loss**

   Insert Chp23_Fig20 here

   When the training is completed, always choose the checkpoint having the least loss. If while training the loss saturates or stops decreasing or increases instead, stop the training and use the checkpoint having the lowest loss till now.

Apart from graphs image predictions on test set can also be seen to check how well the model performs, for e.g.

Insert Chp23_Fig21 here

- **Step 10: Inference**

After training the model, comes the part of evaluation. TensorFlow provides this amazing functionality of exporting a model so that it can be used or served anywhere i.e. in a web-app or an android app. So, the first step to all of this is to create a frozen graph which stores all the values from the model and then can be used for evaluation. To do so run the following command in a cell.

```
!python object_detection/export_inference_graph.py \
      --input_type=image_tensor \
      -- pipeline_config_path=
      /content/gdrive/My\ Drive/tf_object_detection/Projects/CRC_Press/
      pipeline.config \
      --trained_checkpoint_prefix=
      /content/gdrive/My\ Drive/tf_object_detection/Projects/CRC_Press/
      models/model.ckpt-3842 \
      --output_directory=
      /content/gdrive/My\ Drive/tf_object_detection/Projects/CRC_Press/
      models
```

- **Step 11: Evaluation on unseen data:**

The validation folder contains data which has neither been used for training nor testing. Generally, the meaning of Validation set is different but here it's just a name of a folder which has raw or unseen images. To evaluate the model, the frozen graph that was created. To do so, run the object_detection_tutorial.ipynb file from the tensorflow object detection API folder (Complete path from api root folder: /models/research/object_detection/object_detection_tutorial.ipynb)

You just need to edit the paths to different files, more specifically you need to edit the following lines:

```
# Path to frozen detection graph. This is the actual model that is used
 for the object detection.
PATH_TO_FROZEN_GRAPH = '/content/gdrive/My Drive/tf_object_detection/Pr
ojects/CRC_Press/models/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = '/content/gdrive/My Drive/tf_object_detection/Projects
/CRC_Press/label_map.pbtxt'
```

```
# If you want to test the code with your images, just add path to the i
mages to the TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = '/content/gdrive/My Drive/tf_object_detection
/Projects/CRC_Press/data/Validation/Images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR,file ) for fi
le in os.listdir(PATH_TO_TEST_IMAGES_DIR) ]
# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)
TEST_IMAGE_PATHS
```

After you get the results you can validate them with the original annotations and check for accuracy, precision, recall, etc.

After running the jupyter notebook the predictions can be seen in the output of the last cell:

**Prediction:**

Insert Chp23_Fig22 here

**Ground Truth:**

Insert Chp23_Fig23 here

As can be seen, by only training the model for 20000 steps and without any tuning the model has an accuracy of approximately 92.51%. But accuracy shouldn't be the metric (or the only) that a model should focus on. In the discussed problem statement, predicting more blood cells than a person has cannot be afforded and hence the model should focus on reducing the number of false negatives it gets. Hence the correct metric for prediction should be recall rather than accuracy. As can be seen from Table 1 the model fails to successfully detect blood platelets.

**Table 1: Classification Report for WBC's, RBC's and blood platelets**

| | **RBC's** | **WBC's** | **Platelets** |
|---|---|---|---|
| **Accuracy** | 84.78 | 95.65 | 57.97 |
| **Precision** | 95.12 | 90.90 | 93.75 |
| **Recall** | 88.63 | 97.56 | 52.63 |
| **F1-Score** | 91.76 | 94.11 | 67.41 |

Now this can be due to lack of training or bias in the training data wherein the dataset might have a smaller number of images with blood platelets. A good way to reduce these problems might be getting more data, increasing the number of training steps or changing the values of various hyperparameters like learning rate, size of anchor boxes, etc. The model shows high recall for RBC's and WBC's. F1-score is used if we want to maintain a balance between precision and recall. As we can see the model detects WBC's and RBC's easily but detecting platelets is a difficult task for the model. Also, the trained model detects a few cells which haven't been labeled in the dataset, which already shows that it's doing a pretty good job! The overall performance of the model can be seen from Table 2. The model has an overall recall of 75.61 after 20000 steps.

**Table2: Confusion Matrix**

| | | **Predicted** | |
|---|---|---|---|
| | | **Positive** | **Negative** |
| **Actual** | **Positive** | 217 | 70 |
| | **Negative** | 14 | 113 |
| **Recall** | | 75.61 | |

### *23.6.2 Some problems, solutions and suggestions*

1. **Nan Loss error**

   There can be many reasons for this error to occur such as:

   1. High learning rate, leading to divergence of model to infinity.
   2. If the cost function is logarithmic, then it leads to divergence of the model as the cost reaches zero.
   3. Division by zero can also be one of the problems.
   4. Input data contains empty or Nan values.

   One of the reasons why this error could occur with the trained model is maybe because of the manual augmentation. While creating the augmentation script keep in mind that when you change the annotations the bounding boxes should be well within the image. Having bounding boxes outside the image (not intentionally!) could also lead to this problem. Also why changing the annotations keep in mind that none of the values viz. xmin, ymin, xmax, ymax can be zero! This may also lead to the same problem.

2. **GPU cannot allocate memory**

   Now this problem might most probably not occur on google colab but it occurs a lot of times when you train your model offline. The most common reasons for the same being:

   1. Large batch size then can be handled by your computer.
   2. Large prefetch_queue_capacity (Buffer which gets the images beforehand for training)
   3. Large image dimensions then can be stored for a particular queue size

   If you get this error simply change their values in the config file.

**3. Transport endpoint not connected - only on colab**

This error occurs because of communication problems between drive and google colab. To get rid of this problem simply restart the runtime and the problem will be solved.

**4. Problem Generating tfrecords**

The most common reason for this problem is missing annotation for corresponding images in the csv files. Another thing that people generally forget, which creates problems, is changing the mapping according to the label_map.txt file in the gen_tfrecords.py file.

```
# TO-DO replace this with label map
def class_text_to_int(row_label):
    if row_label == 'RBC':
      return 1
    elif row_label == 'WBC':
      return 2
    elif row_label == 'Platelets':
      return 3
```

There can be many more big and small problems, so to avoid those you can follow the following suggestions:

1. Check the data properly before using it for training.
2. Make sure that the data is properly distributed among the different classes to avoid introducing any kind of bias.
3. Have a proper project structure to avoid confusion.
4. Follow the above steps in the given sequence only.
5. If you augment the data manually, make sure you check the data properly after augmenting as it can result in unexpected results or even some painful errors!
6. After the training is completed, always select the checkpoint with the lowest loss for better results and add the checkpoints name in the "checkpoint" file in the

"models" directory so that whenever you increase the number of steps for the model it starts training from the best step till now.

7. Do not increase the number of training steps too much as it can lead to overfitting.
8. Once you have completed your training, try experimenting with the hyperparameters for better results.

These were a few tips but the list never ends and as always internet community is always there to help.

## 23.7 Viability of the Solution:

The problem that this chapter solves is based on one of the most common issues from the daily life. Everyone in the world, at least once in their lifetime, goes through a CBC Test. A general CBC test takes around 24 hrs and during this 24 hrs a lot of manual effort goes into completing these tests and getting the results which otherwise could have been used to get more important tasks done. The proposed solution helps in getting the results within a few minutes which would have otherwise taken hours.

Even though it sounds excellent, the main problem that this kind of solutions face is **"trust"**. Trusting this type of solutions is a major challenge today because these solutions are never accurate and the type of problem that the chapter discusses doesn't allow mistakes. Still this problem does not largely affect the solution that this chapter discusses because in the end the results are extrapolated in the most modern techniques too and with a bit of tuning and more data, the recall can be easily increased and made greater than 95%. So, after extrapolating the results from the described models predicrions, the final results shouldn't be too different from the original ones. It can provide the same results in less time and with a few clicks from your computer. Hence, with the above recall and performance the above solution can help in revolutionizing the way medical science has been working till date and help in easing up the process of Complete Blood Count Tests.

## 23.8 References

1. Daqqa, Khaled A. S. Abu, Ashraf Y. A. Maghari and W. A. Sarraj. 2017 "Prediction and diagnosis of leukemia using classification algorithms." 8th International Conference on Information Technology (ICIT): 638-643.
2. Cruz, D., C. Jennifer, Valiente, Leonardo C. Castor, Celine Margaret T. Mendoza, B. Jay, L. Jane and P. Brian. 2017 "Determination of blood components (WBCs, RBCs, and Platelets) count in microscopic images using image processing and analysis." IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM): 1-7.
3. Habibzadeh, M., A. Krzyżak and T. Fevens. 2013 "White Blood Cell Differential Counts Using Convolutional Neural Networks for Low Resolution Images." ICAISC.
4. Kaur, P., Vishakha Sharma and N. Garg. 2016 "Platelet count using image processing." 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom): 2574-2577.
5. Meimban, Raymond Joseph, Alexa Ray Fernando, Armil Monsura, Jonrey V. Rañada and J. C Apduhan. 2018 "Blood Cells Counting using Python OpenCV." 14th IEEE International Conference on Signal Processing (ICSP): 50-53.

6. George-Gay, B. and K. Parker. 2003 "Understanding the complete blood count with differential." Journal of perianesthesia nursing : official journal of the American Society of PeriAnesthesia Nurses 18 2: 96-114; quiz 115-7 .
7. Meintker, L., J. Ringwald, M. Rauh and S. Krause. 2013 "Comparison of automated differential blood cell counts from Abbott Sapphire, Siemens Advia 120, Beckman Coulter DxH 800, and Sysmex XE-2100 in normal and pathologic samples." American journal of clinical pathology 139 5: 641-50 .
8. Rosyadi, Tsalis, Agus Arif, Nopriadi, Balza Achmad and Faridah. 2016 "Classification of leukocyte images using K-Means Clustering based on geometry features." 6th International Annual Engineering Seminar (InAES): 245-249.
9. Dvanesh, Varun D, P. S. Lakshmi, K. Reddy and Abirami S Vasavi. 2018 "Blood Cell Count using Digital Image Processing." International Conference on Current Trends towards Converging Technologies (ICCTCT): 1-7.
10. Alam, Mohammad Mahmudul and M. T. Islam. 2019 "Machine learning approach of automatic identification and counting of blood cells." Healthcare Technology Letters 6: 103 - 108.
11. Dhieb, Najmeddine, Hakim Ghazzai, Hichem Besbes and Y. Massoud. 2019 “An Automated Blood Cells Counting and Classification Framework using Mask R-CNN Deep Learning Model.” 31st International Conference on Microelectronics (ICM): 300-303.
12. Krizhevsky, A., Ilya Sutskever and Geoffrey E. Hinton. 2017 "ImageNet classification with deep convolutional neural networks." CACM.
13. Aston Zhang, Zachary C. Lipton, Mu Li et al, 2020 "Dive into Deep Learning"

https://d2l.ai/chapter_computer-vision/rcnn.html

14. Lilian Weng, 2017 "Object Detection for Dummies Part 3: R-CNN Family" https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html
15. Girshick, Ross B.. "Fast R-CNN." 2015 IEEE International Conference on Computer Vision (ICCV): 1440-1448.
16. Ren, Shaoqing, Kaiming He, Ross B. Girshick and J. Sun. 2015 "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." IEEE Transactions on Pattern Analysis and Machine Intelligence 39: 1137-1149.
17. Alam, Mohammad Mahmudul and M. T. Islam. 2019 "Machine learning approach of automatic identification and counting of blood cells." Healthcare Technology Letters 6: 103 - 108.
18. Everingham, M., L. Gool, C. K. Williams, J. Winn and Andrew Zisserman. 2009 "The Pascal Visual Object Classes (VOC) Challenge." International Journal of Computer Vision 88: 303-338.