Instance Segmentation of Peripheral Blood Smear and Refining Classification via Domain Adaptation

Jimut Bahan Pal

Department of Computer Science

Ramakrishna Mission Vivekananda Educational and Research Institute

Belur Math, Howrah

Pin - 711 202, West Bengal

A thesis submitted to

Ramakrishna Mission Vivekananda Educational and Research Institute

in partial fulfillment of the requirements for the degree of

MSc in Computer Science

2021

Dedicated to My Parents

The entire work is dedicated to my parents for whom I have got a chance to pursue higher education. They have been a constant source of moral, spiritual, financial and emotional support.

Acknowledgements

It is ritual that scholars express their gratitude to their supervisors. This acknowledgement is very special to me to express my deepest sense of gratitude and pay respect to my supervisor, **Br. Tamal Maharaj**, Department of Computer Science, for his constant encouragement, guidance, supervision, and support throughout the completion of this thesis. His close scrutiny, constructive criticism, and intellectual insight have immensely helped me in every stage of my work. I would like to thank him for patiently answering my often-naive questions related to Computer Vision.

I am thankful to **Aniket Bhattacharyea** for helping me with this thesis, without his help, certain part of the thesis would have taken a much longer time (specially Mask RCNN pipeline). I am indebted to **Dr. Debashish Banerjee** for providing with the gold standard blood smear data which was used in this thesis. I would like to thank all the professors of Department of Computer Science for creating an intellectually stimulating environment that enabled me to think more carefully and methodically about my research than I had ever done before. I would like to thank **Swathy Prabhu Mj** for for arranging Asus RTX 2080 Ti (12 GB) and Quadro GV100 (32 GB) GPUs with 64 GB RAM, to hasten the research, and also to **Dripta Mj** for the stimulating discussions that he had shared with me during the Advanced Machine Learning course.

I am grateful to my father, **Dr. Jadab Kumar Pal**, Deputy Chief Executive, Indian Statistical Institute, Kolkata for constantly motivating and supporting me to develop this documentation along with the proofreading. I will also mention about my brother **Jisnoo Dev Pal** and my mother **Sumita Pal**, for supporting me.

Ramakrishna Mission Vivekananda Educational
and Research Institute, Belur Math, West Bengal

Jimut Bahan Pal

May 31, 2021

CERTIFICATE FROM THE SUPERVISOR

This is to certify that the thesis entitled *'Instance Segmentation of Peripheral Blood Smear and Refining Classification via Domain Adaptation'* submitted by *Mr. Jimut Bahan Pal*, who has been registered for the award of MSc in Computer Science degree of Ramakrishna Mission Vivekananda Educational and Research Institute, Belur Math, Howrah, West Bengal is absolutely based upon his own work under the supervision of *Br. Tamal Maharaj* of Department of Computer Science, Ramakrishna Mission Vivekananda Educational and Research Institute and that neither his/her thesis nor any part of the thesis has been submitted for any degree/diploma or any other academic award anywhere before.

Br. Tamal Maharaj
Assistant Professor
Department of Computer Science
Ramakrishna Mission Vivekananda Educational and Research Institute
Belur Math, Howrah, 711 202, West Bengal

Abstract

Deep Learning models are data dependent, i.e., they only perform well on unseen data from the same distribution on which they were trained on. In the case of medical sectors, collecting and annotating data is a very costly process, in the sense, it requires expert annotators who can hardly give enough time for annotating and collecting data. For this reason, there are small repositories of data present everywhere which varies even if the number of classes are same. In this work, we have built an automated process to annotate and mask cells of Peripheral Blood Smear, also classify the data using a closely related PBC dataset via domain adaptation to learn domain invariant features. This will help the process of counting and identifying different types of white blood cells present in smear to identify certain diseases according to some distribution. A novel demographic smear dataset is also built in the process which will further help researchers to do novel innovative tasks, e.g., generating new data synthetically via original dataset to suit any demography. In the process, we have also surpassed the state-of-the-art model for PBC dataset classification, and explored several ingenious ways to perform masking.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Peripheral blood smear is a procedure to count and investigate blood samples [15] under a microscope. There are various types of white blood cells that are present in human body, some of them are shown in Figure 1.1. These cells arise in the bone marrow and gives rise to different types of other blood cells, and hence known as hematopoietic cells. The problem is to find instances of White Blood Cells (WBC's) from peripheral blood smear and detect which classes it belongs to.

The peripheral blood smear deals with counting and classifying what cells are present in a slide, hence our task is to detect and classify, and then segment what type of blood cells are present in a given image. This will help the doctors to automate the process of analyzing slides containing cells. There is very little related work in this field since, getting the data is very costly. Data creation is an even costlier process, given it needs to be labelled by expert annotators. The data may vary according to demography of a place and analyzing the slides is only done by medical experts.

We have received a part of our data from leading hematologist (Ramakrishna Mission Seva Pratishthan, Kolkata [1]). We have labelled each of the data by creating segmentation masks. A typical smear slide is shown in Figure 1.2. Since data is limited, we need to adapt the data to another similar dataset with similar classes. For this purpose, we perform domain adaptation on the PBC dataset which has 17092 images to classify data from smear slides. The pipeline for doing so is straightfor-

[1]https://belurmath.org/ramakrishna-mission-seva-pratishthan-kolkata/

ward. First, we have to detect all the cells present in the slide via Mask RCNN, Second, using the coordinates of the detected cell, we have to crop out and classify which cell it is via the domain adaptation target classifier. We will see the methods involved in doing so in the preceding sections.

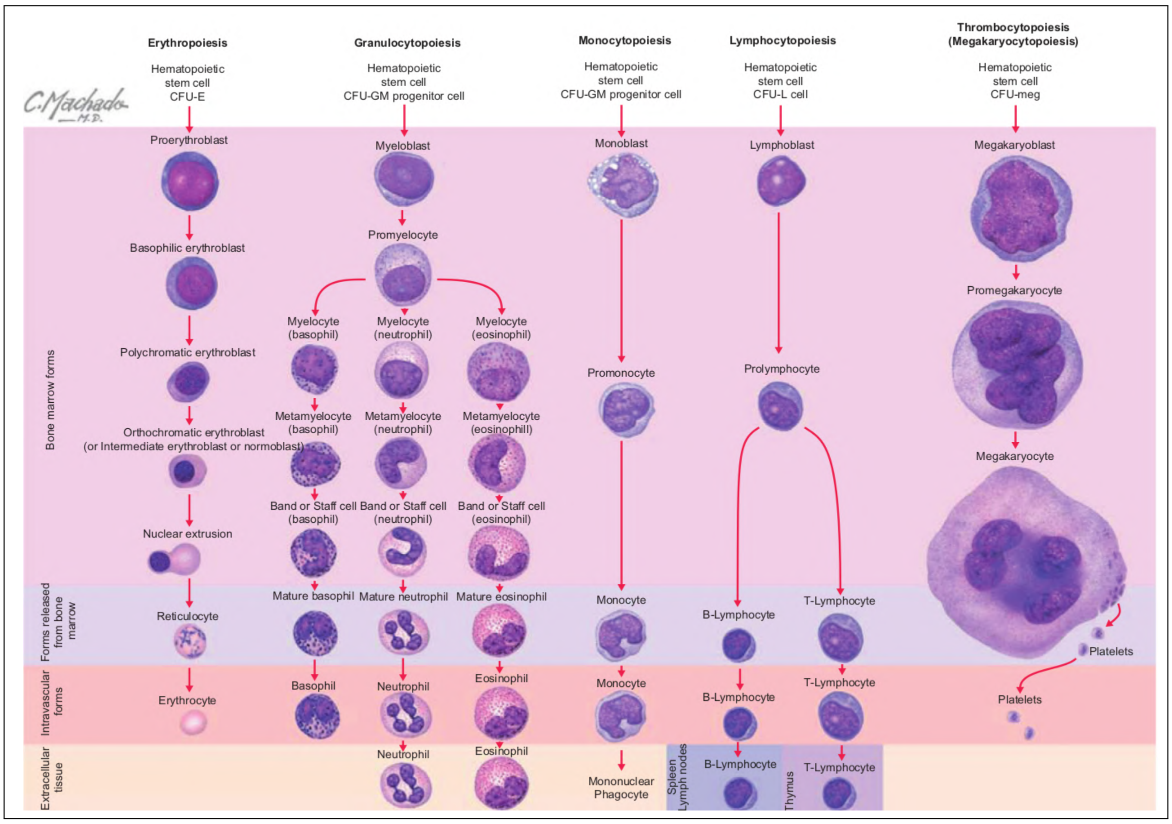


Figure 1.1: Maturation and different types of human hematopoietic cells, Picture Courtesy: Elsevier Inc `https://www.netterimages.com/`. We need not classify all the cells present in this picture, since some cells are very rare and some forms a class of other cells. The cells that are needed to be classified are discussed in the dataset section.

The thesis is divided into the following chapters, first, we will look into the datasets used and their overview. For getting the actual cell image from slides, we have to perform certain masking techniques, we have tested various classical methods for masking as well as novel deep learning methods. After getting the mask, we need to extract the cell, hence we need precise bounding boxes and masks for doing so. After the cell is extracted via Mask RCNN which takes care of region proposal network and instance segmentation, we pass the segmented cell to the classifier

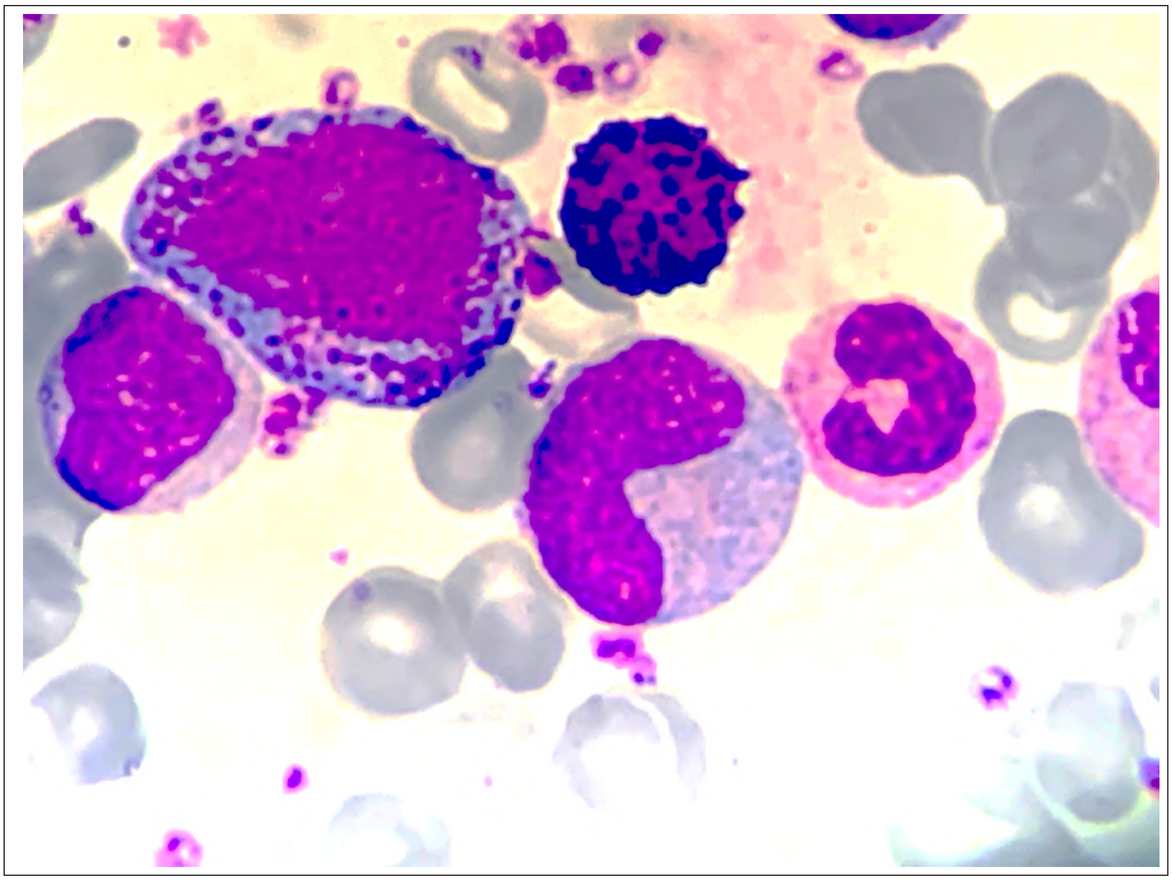


Figure 1.2: A typical sample of smear slide under a microscope. Picture Courtesy: Dr. Debashish Banerjee. A smear may contain various types of cells, mostly Red Blood Cells (RBCs) as shown in gray disc shapes present all over the slides, we need to take care of only White Blood Cells (WBCs), which are colored via certain chemicals in purple that helps them to distinguish under a microscope.

which has been trained via domain adaptation to get actual detected labels of cells, which acts as a layer of confidence from the detected label from Mask RCNN. This helps the classifier learn features from both the domains of the data in a domain invariant way. Using as much data as possible, since medical data are scarce, we have created a pipeline which can detect, segment and also create mask for unseen images.

Chapter 2

Datasets

We have created a variety of dataset in this study, mainly a segmentation dataset and a few classification dataset. We have also used the standard PBC dataset for our study. The datasets are discussed in the following sections.

2.1 PBC_dataset_normal_DIB

This is a classification dataset that is taken from Mendeley [1] (`https://data.mendeley.com/datasets/snkd93bnjr/1`). There are a total of 17092 images present in this dataset [3]. The classes that are present are:

- **Basophil** (BA)
- **Eosinophil** (EO)
- **Lymphocyte** (LY)
- **Monocyte** (MO)
- **Immature Granulocytes** (IG)
  - Immature Granulocytes (IG)
  - Myelocyte (MY)
  - Metamyelocyte (MMY)
  - Promyelocytes (PMY)

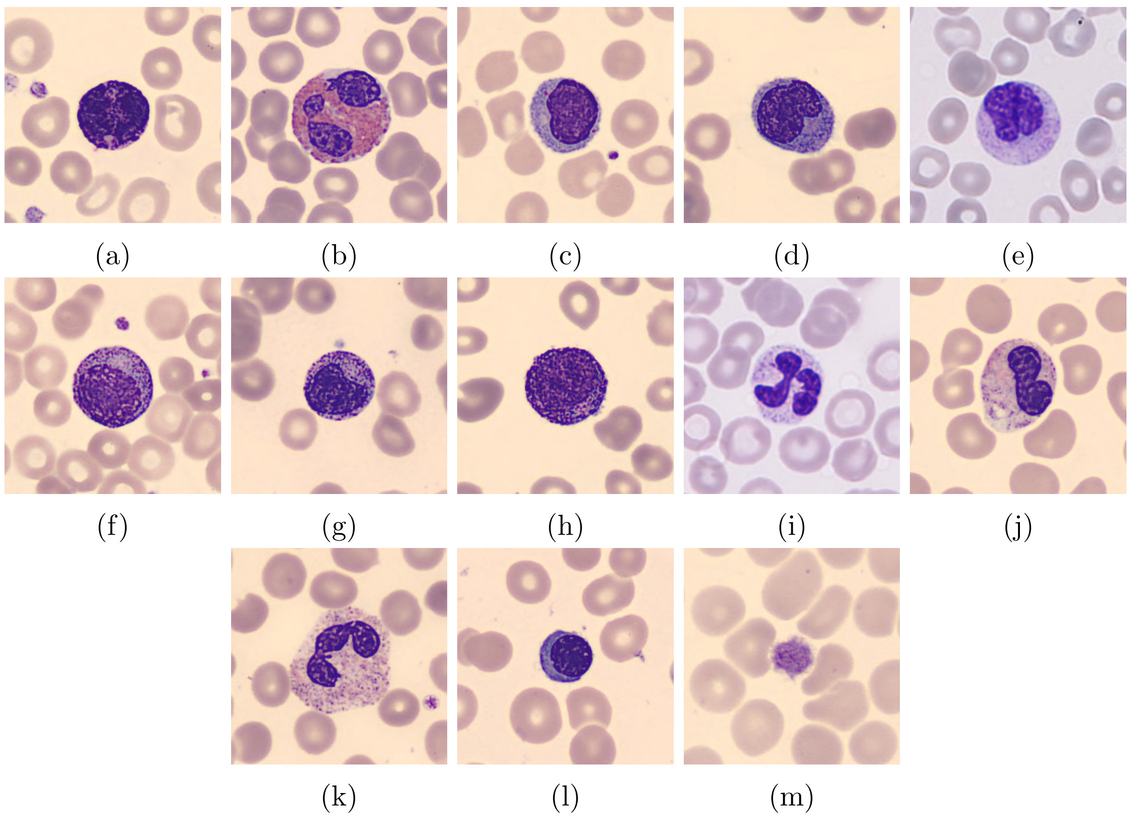


Figure 2.1: Images of samples taken from PBC_dataset_normal_DIB dataset. From top left to bottom right: (a) **Basophil** (BA), (b) **Eosinophil** (EO), (c) **Lymphocyte** (LY), (d) **Monocyte** (MO), (e) **Immature Granulocytes** (IG), (f) Myelocyte (MY), (g) Metamyelocyte (MMY), (h) Promyelocytes (PMY), (i) **Neutrophil** (NEUTROPHIL), (j) Band Neutrophils (BNE), (k) Segmented Neutrophils (SNE), (l) **Erythroblast** (ERB), (m) **Platelet** (PLATELET).

- **Neutrophil** (NEUTROPHIL)
  - Neutrophil (NEUTROPHIL)
  - Band Neutrophils (BNE)
  - Segmented Neutrophils (SNE)
- **Erythroblast** (ERB)
- **Platelet** (PLATELET)

The distribution of the individual classes of the dataset is shown in Figure 2.2.

Samples taken from each of the classes are shown in Figure 2.1. The classes labelled in bold are the superset of classes that is followed by them. There are 8

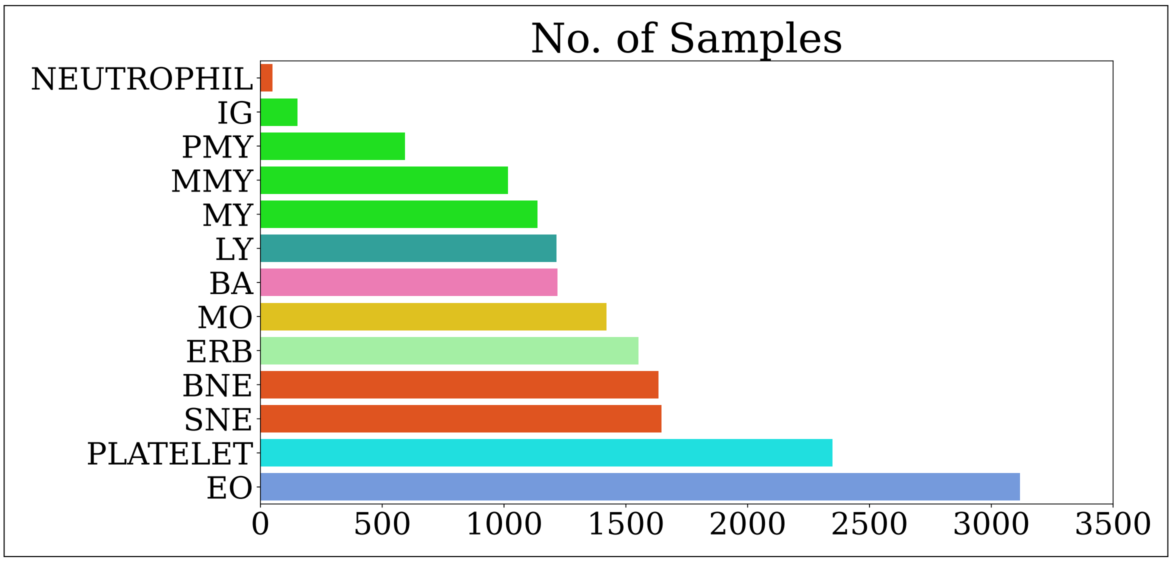


Figure 2.2: The distribution of individual samples of the PBC_dataset_normal_DIB dataset. Note that, the dataset is highly imbalanced but we need to group some of the classes to reduce the discrepancy.

classes in total that comprise this dataset.

2.2 PBC_dataset_normal_DIB_cropped

This dataset is a modified version of previous PBC_dataset_normal_DIB dataset. Thus, the distribution of different classes is same as the previous one. It is easy for the model to learn essential features during domain adaptation when the two classes have high resolution (same type) imagery in them, in that way the model can focus on only the relevant cells rather than a bigger slide containing cluttered RBC's.

The algorithm for extracting the cropped image is discussed in Algorithm 15. We use color thresholding to get the mask using Hue Saturation Value [1] scale. After that we use watershed [4] algorithm to extract the contours that is got from the mask. We will discuss more about water shed algorithm in Methods for Masking section. Since most of the cells start at the center of the PBC slide, but are of different shapes and sizes, we can crop out a radius of the biggest contour starting from center. We repeat this for every image of the dataset to get the whole cropped

[1] `https://en.wikipedia.org/wiki/HSL_and_HSV`

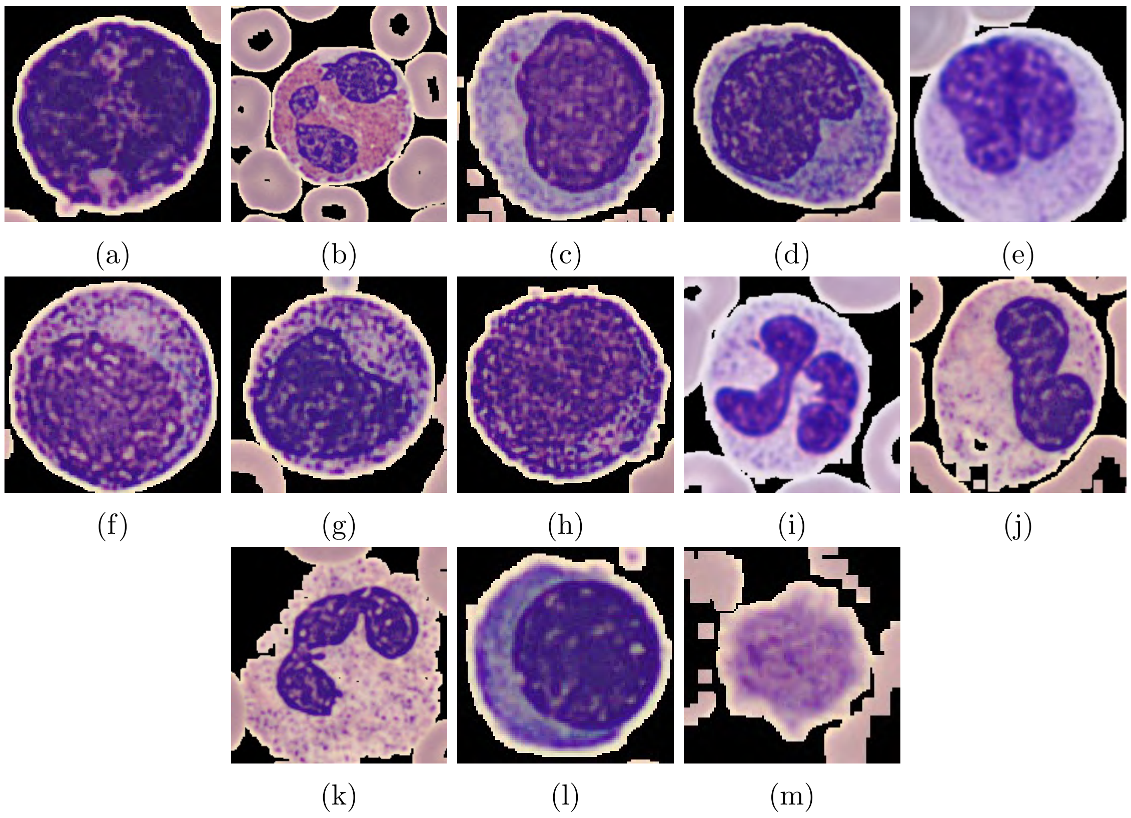


Figure 2.3: Images of samples taken from PBC_dataset_normal_DIB_cropped dataset. From top left to bottom right: (a) **Basophil** (BA), (b) **Eosinophil** (EO), (c) **Lymphocyte** (LY), (d) **Monocyte** (MO), (e) **Immature Granulocytes** (IG), (f) Myelocyte (MY), (g) Metamyelocyte (MMY), (h) Promyelocytes (PMY), (i) **Neutrophil** (NEUTROPHIL), (j) Band Neutrophils (BNE), (k) Segmented Neutrophils (SNE), (l) **Erythroblast** (ERB), (m) **Platelet** (PLATELET).

dataset. It took **0.1448 seconds** to process an image on average. The results of the methods related to the algorithm are shown in Figure 2.6. The samples of the images from this dataset is shown in Figure 2.3.

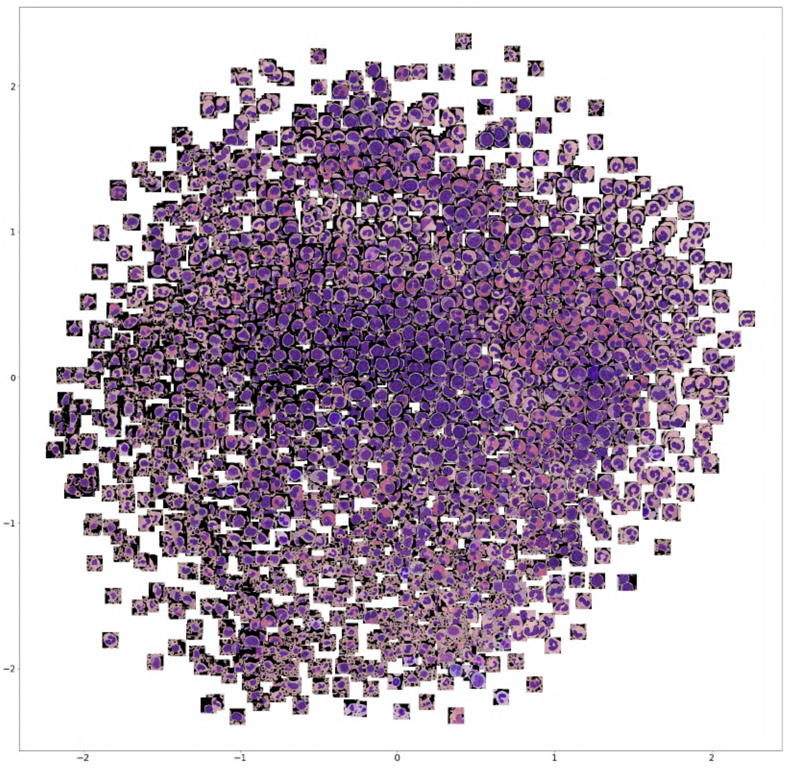
Visualization of PBC_dataset_normal_DIB_cropped dataset's test set, i.e., 2609 images shown using t-SNE [24] (perplexity 50) visualization in Figure 2.4. This is created by firstly doing PCA (Principal Component Analysis) [13] using 200 components on images of size (360, 360, 3) and then visualizing the images. Similarly, using a perplexity of 30 and taking 50 components on images of sizes (360, 360, 3), we visualize the Smear Slides Cropped Dataset using t-SNE as shown in Figure 2.5 on 151 images (from test dataset).

Algorithm 1: Algorithm for extracting the cells from **PBC_dataset_normal_DIB** dataset.

```
for each image of the PBC_dataset_normal_DIB dataset do
    image ← individual image of PBC_dataset_normal_DIB dataset;
    img ← copy of image;
    hsv ← convert img from BGR to HSV;
    lower_blue ← (100, 20, 20);
    upper_blue ← (300, 245, 245);
    mask ← threshold hsv within [lower_blue, upper_blue];
    mask ← dilate mask with 3 iterations;
    res ← bitwise and with img and mask;
    local_Max ← compute the exact euclidean distance from every binary
      pixel to the nearest zero pixel, then find peaks in this distance map
      with min distance of 20 on mask;
    markers ← perform a connected component analysis on the local
      peaks, using $8 - connectivity$, then appy the watershed algorithm
      using local_Max;
    labels ← apply watershed algorithm using markers and mask;
    image ← draw circles from the uniquely identified contours and find
      the coordinates of the biggest contour and store the radius using
      labels on image;
    crop ← Take H/2, W/2 as the central point and crop out the masked
      image by taking the radius as the boundary;
end
```

2.3 Peripheral Blood Smear Slides and Smear Slides Cropped

This is an ongoing dataset that is being created using CVAT, and the raw cell images are being provided continuously. Since this is a segmentation dataset, we can crop out the images as shown in Figure 2.7 to create a classification dataset for domain adaptation. A sample of the dataset being created for the Basophil class is shown in Figure 2.8. The distribution of cells extracted from this dataset is shown in Figure 2.9b. We will use this part of the dataset in performing Domain Adaptation, by using those classes which are common to both PBC cropped dataset and Smear Slide Cropped dataset.



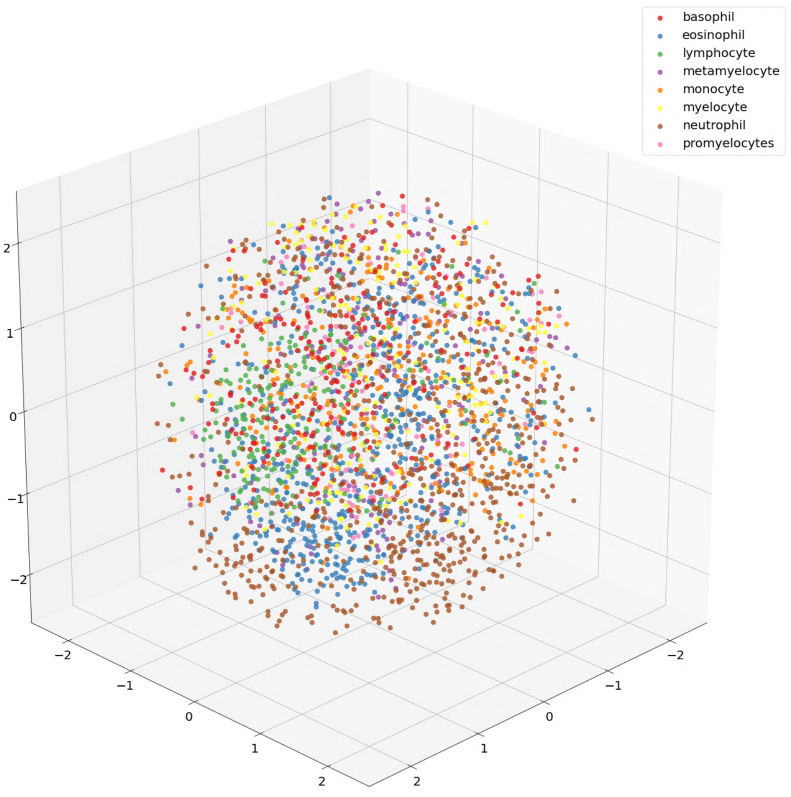
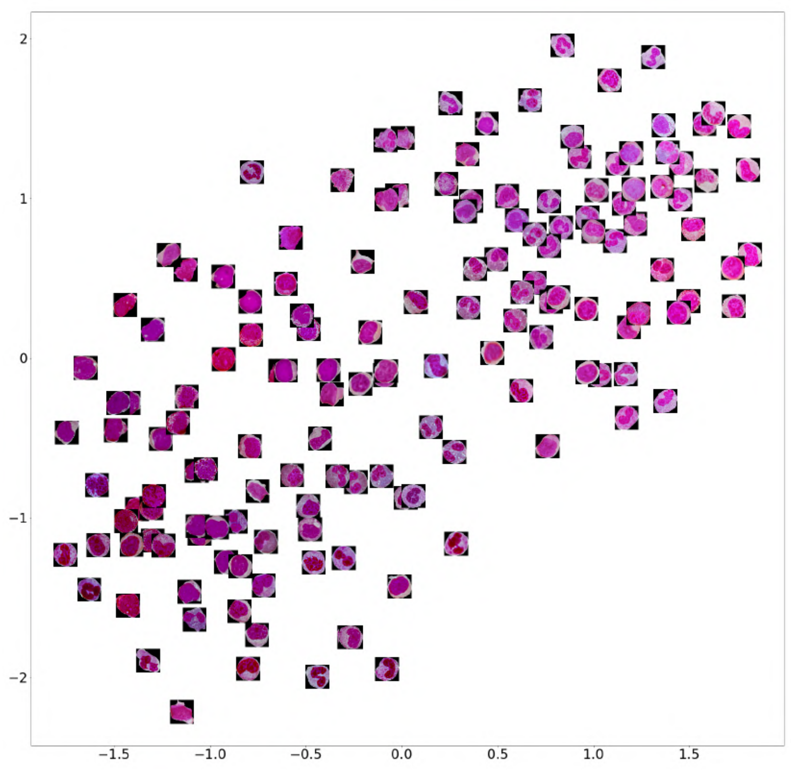


Figure 2.4: t-SNE visualization of 2609 test images of PBC_dataset_normal_DIB_cropped dataset's test set (Up: Only images in 2D, Bottom: Each of the individual images represented by class in 3D space).



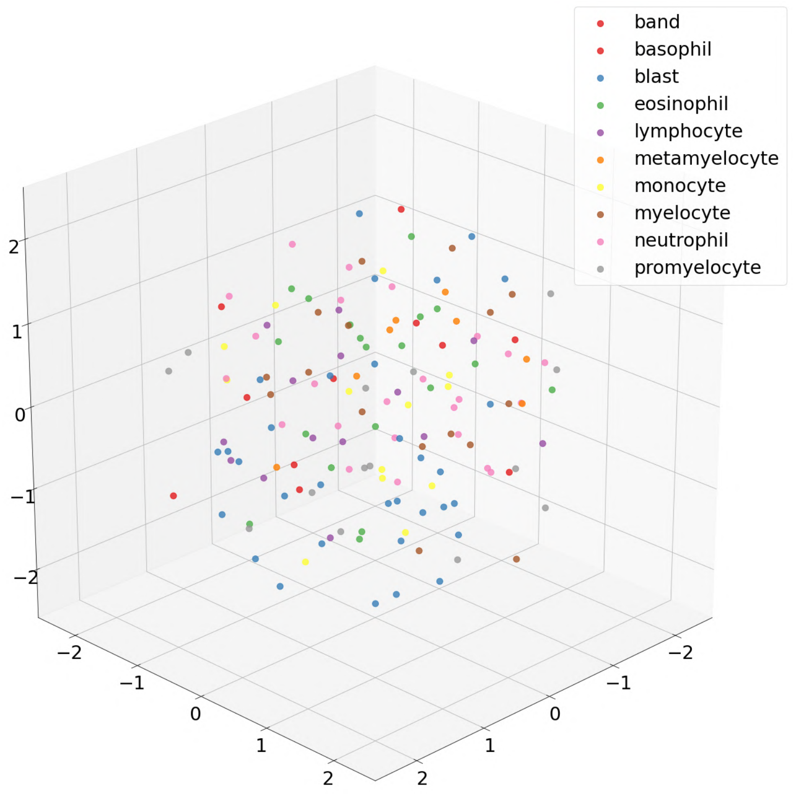


Figure 2.5: t-SNE visualization of 151 test images of Smear Slides Cropped dataset's test set (Up: Only images in 2D, Bottom: Each of the individual images represented by class in 3D space).

The classes that are present in this dataset are shown below:

- **Band Cell**
- **Basophil**
- **Blast Cell**
- **Eosinophils**
- **Lymphocytes**
- **Myelocytes**
- **Metamyelocytes**
- **Monocytes**
- **Neutrophil**
- **Promyelocytes**

The data distribution of the classes of the slides present is shown in Figure 2.9a.

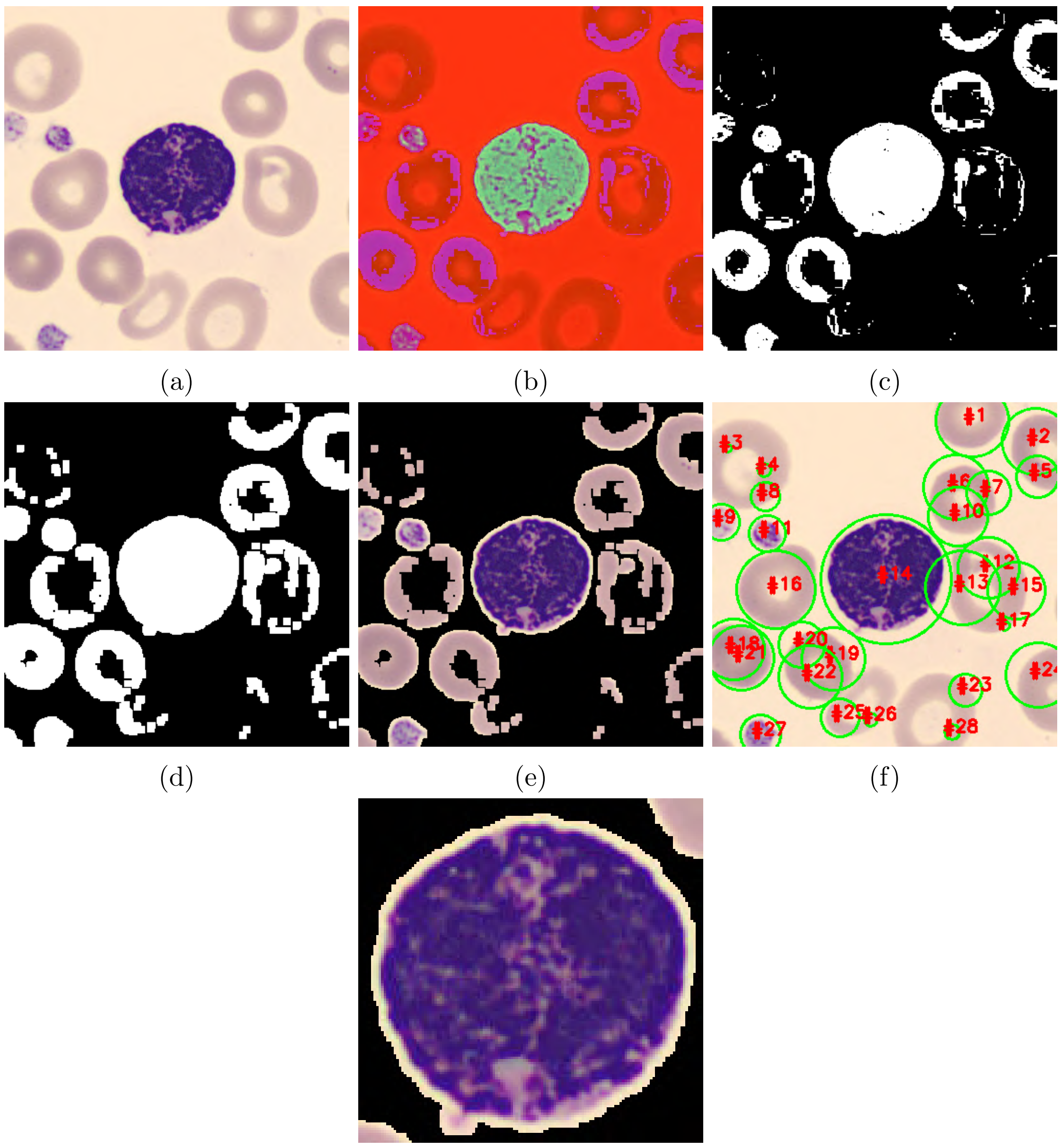


Figure 2.6: The process of automating the cropping of PBC_dataset_normal_DIB dataset. From top left to bottom: (a) A sample image from PBC_dataset_normal_DIB dataset, (b) the image is converted from BGR to HSV, (c) mask obtained after applying the lower and upper threshold of blue color, (d) the dilated mask to make it more prominent, (e) *Bitwise and* with the mask and the original image, (f) Contours obtained by applying watershed algorithm on the mask, (g) cropped out segmented image.

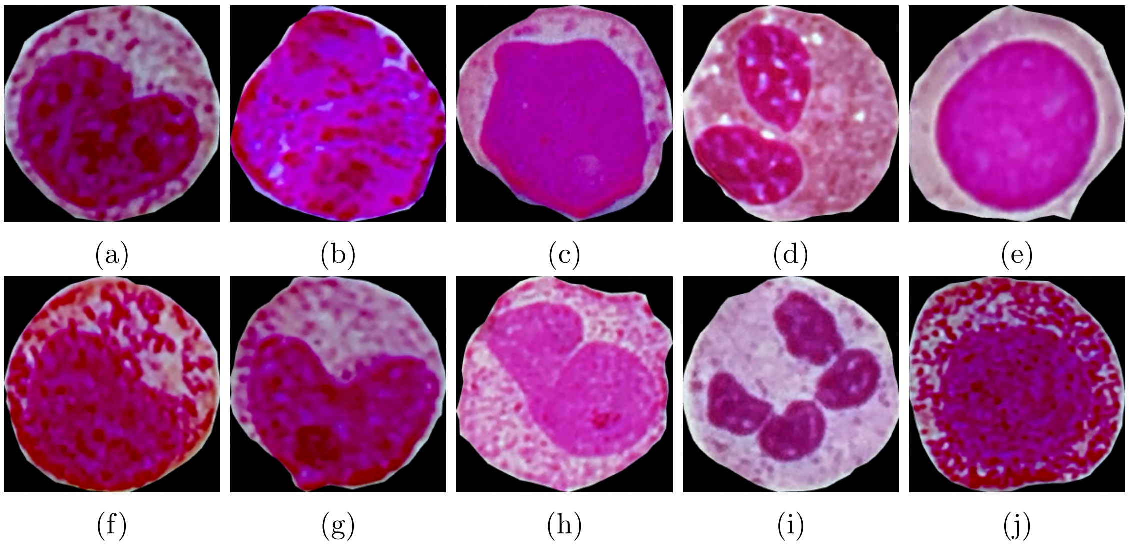


Figure 2.7: Images of slides cropped from each of the classes from whole Smear Slides Cropped Dataset. There are a total of 10 classes, From top left: (a) Band Cell, (b) Basophil, (c) Blast Cell, (d) Eosinophils, (e) Lymphocytes, (f) Myelocytes, (g) Metamyelocytes, (h) Monocytes, (i) Neutrophil, and (j) Promyelocytes. These samples will be used to do Domain Adaptation.

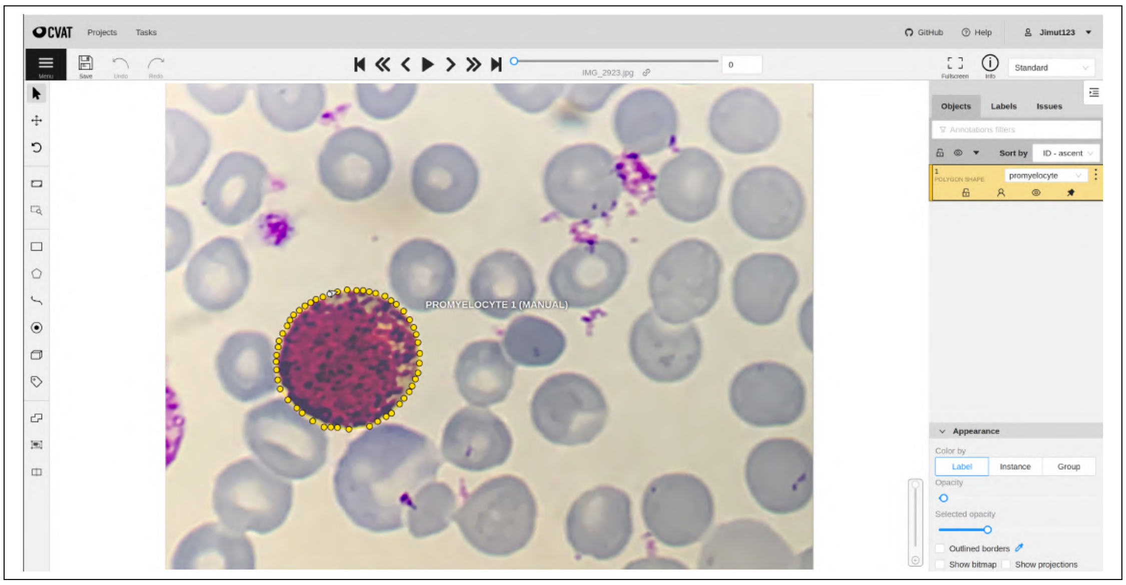
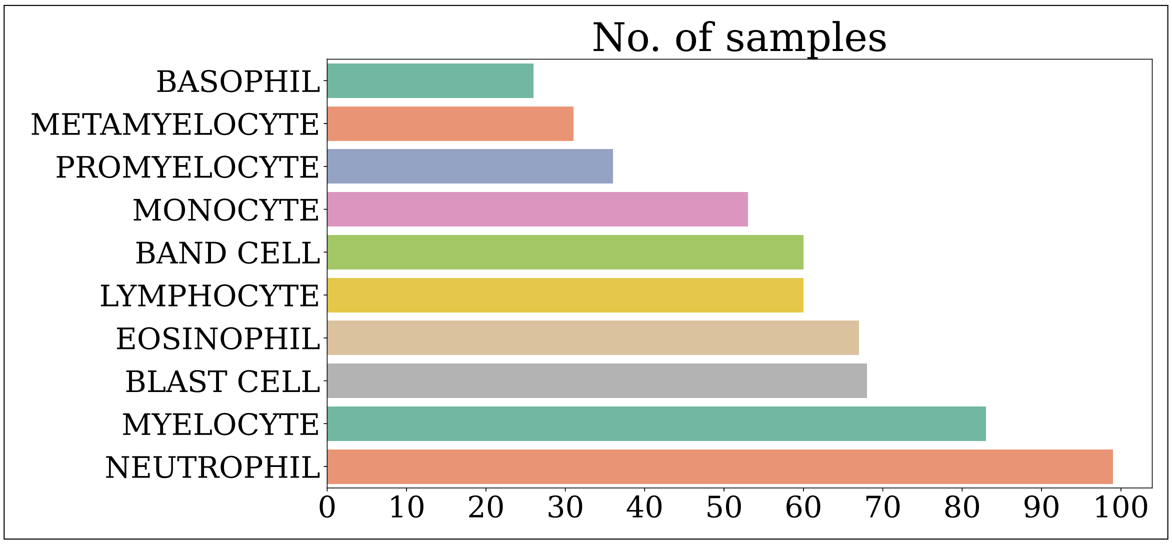
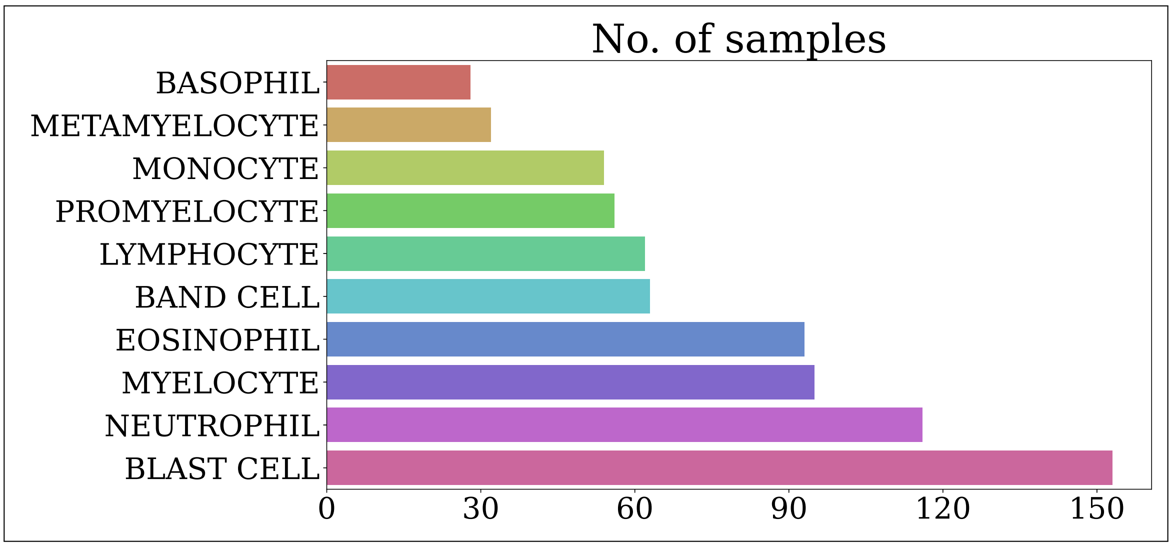


Figure 2.8: A sample of Basophil being labelled using CVAT (Computer Vision Annotation Tool) software to create a segmentation dataset from Smear Slides.



(a) Data distribution of slides of Smear Slides.



(b) Data distribution of individual cells cropped from Smear Slides.

Figure 2.9: Distribution of Smear Slide Datasets, (a) Segmentation dataset (b) Cropped Classification dataset.

Chapter 3

Experimental Methods for Masking

To get the domain adaptation pipeline running we need to firstly segment the cell to detect which class it belongs to. The more clearly it segments the cell, the more helpful it will be to for the detection model to classify the class of the cell. There are various masking methods that have been tested, starting from the classic watershed algorithm, to detecting blobs, to using novel GRAD CAM method for creating the mask for cell. Later we found out that Mask RCNN does a good job in segmenting out the cell. Each of the methods are discussed in the following sections.

3.1 Watershed Algorithm

Watershed algorithm [4] is a classical algorithm used for image segmentation. Using watershed one can use user defined markers, to fill local minima of topographic elevations. It treats input images' pixel's intensity values as topographic elevations. The filling starts from local minima and continues till there is a boundary between two water bodies (here, also referred to as images).

A sample of image taken from Smear slide is used to demonstrate simple example of watershed algorithm as shown in Figure 3.1. We can see that watershed performs quite good job in segmenting the cell from this image. This is not true for all the images, for example, the image shown in Figure 3.2 (above) does a relatively good job in segmenting blast cell, but it doesn't do a good job for segmenting promyelocytes cells as shown in Figure 3.2 (below). So, when we need extreme precision to segment out cell's mask, watershed algorithm will not do a good job.

3.2 DoH (Difference of Hessian), DoG (Difference of Gaussian) and LoG (Laplacian of Gaussian)

Blob detectors are one of the earliest classical methods for detecting interest points. In this section we will describe our own combined method for segmenting cells from PBC_dataset_normal_DIB dataset.

The method presented in Algorithm 12 gives an approximate mask for segmenting PBC dataset's cells. The reason behind this is, the cells are of different shapes and sizes, and blob detectors can capture the red blood cells too which are present in enormous amount. What we did is, the bigger the size of the detected cell, the more the layer of thickness added to the confidence mask. We add a non-linearity to the addition of mask purposefully to gain more weight to the bigger sized cells. After getting the layers of confidences, we threshold and apply erosion to get the most, prominent area of the cell, consequently deleting the less confidence areas.

Algorithm 2: Algorithm for extracting mask from the cells of **PBC_dataset_normal_DIB** dataset.

```
1  im ← individual image of PBC_dataset_normal_DIB dataset ;
2  src ← copy of im ;
3  src ← Gaussian Blur using $(3, 3)$ kernel on src ;
4  gray ← convert im to gray scale ;
5  imgs ← Apply Sobel filter to gray and select the best one using scale
     range in [0,10] and delta range in [0,10] imgs ← Gaussian blur imgs with
     kernel size (3,3) ;
6  blobs_log ← Laplacian of Gaussian with max $\sigma = 80$, num $\sigma = 10$,
     threshold = 0.1;
7  blobs_dog ← Difference of Gaussian with max $\sigma = 80$, threshold = 1;
8  blobs_doh ← Difference of Gaussian with max $\sigma = 80$, threshold = 0.001;
9  c ← compute circles with radius of $\exp^{\log_2(r)}$ and add individal circles to
     the layers of blank image;
10 layer ← threshold to get mask;
11 mask ← apply erosion on layer;
12 repeat for every image of PBC_dataset_normal_DIB dataset;
```

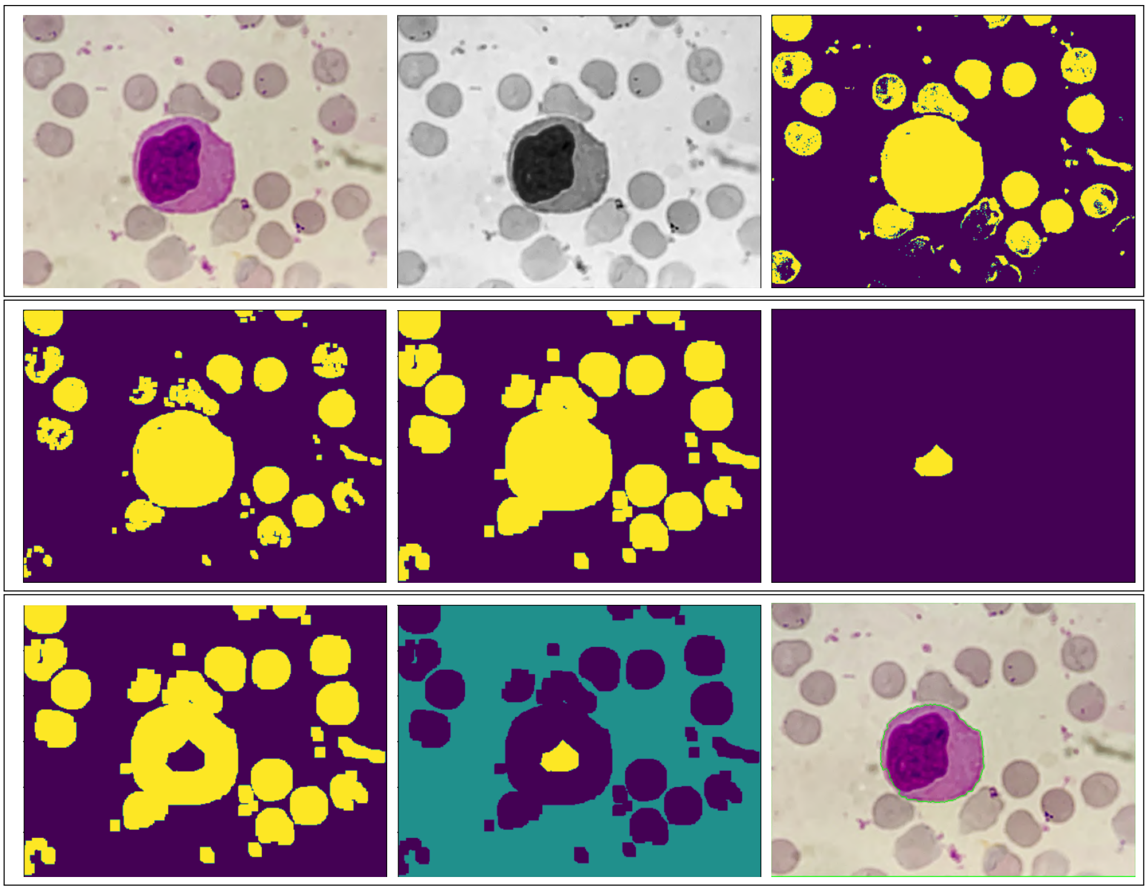


Figure 3.1: The steps involved in performing watershed algorithm to a slide. From top left to bottom right: the original slide image, the gray scaled version of original image, the gray scale image after being applied Otsu's thresholding, performing opening (morphological operation) on the resulted image for 2 iterations, after dilation of the resulting image which forms the background, after applying L2 distance which is sure foreground, the unknown area, the markers for performing watershed algorithm, the resulting image after watershed algorithm.

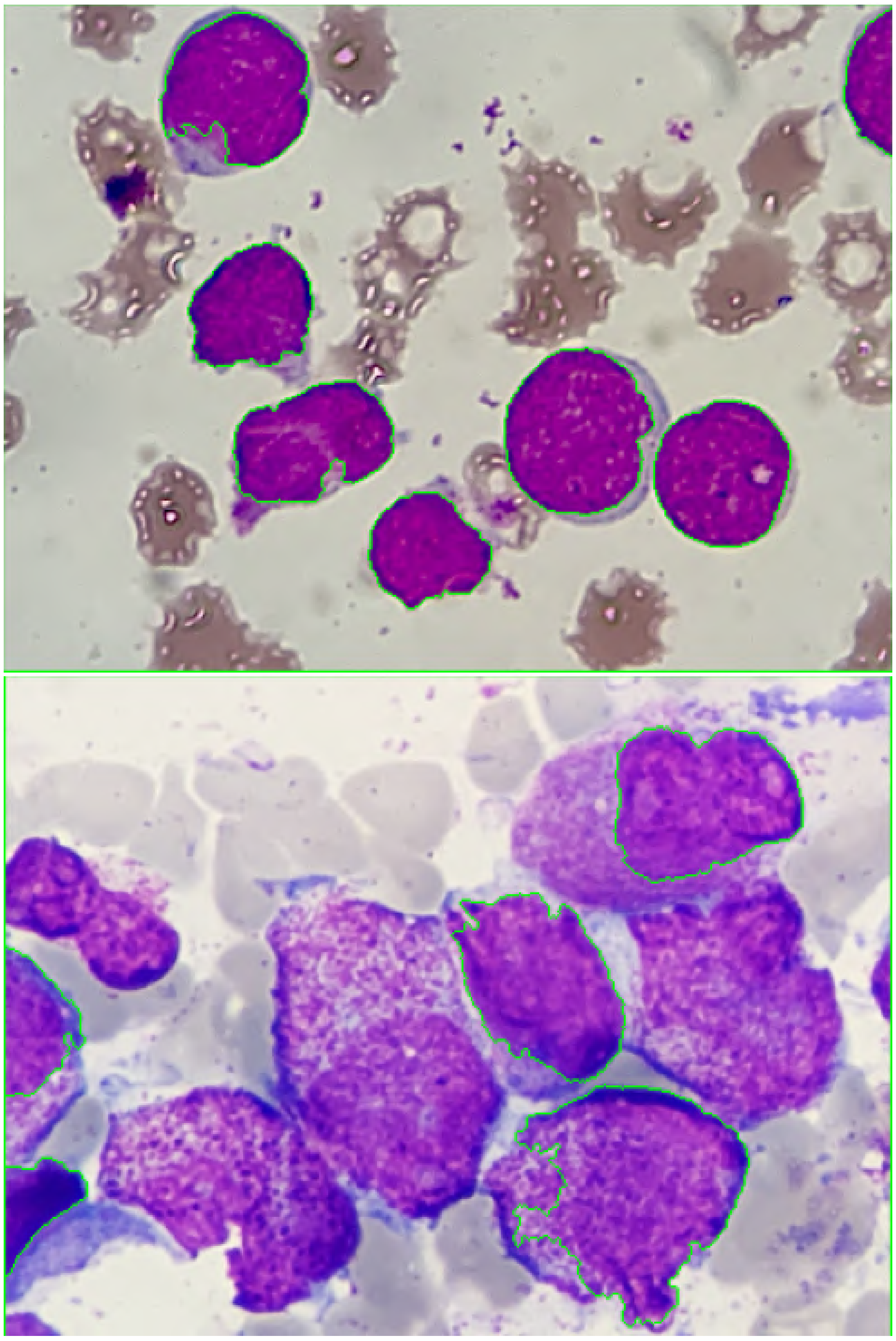


Figure 3.2: The result of applying watershed algorithm on an image of slide containing blast cell, which is able to segment almost all the cells properly (as seen at the top, which is a good example). The figure below shows the application of watershed on a slide containing promyelocytes and it is a bad example, cells are not segmented right by watershed algorithm.

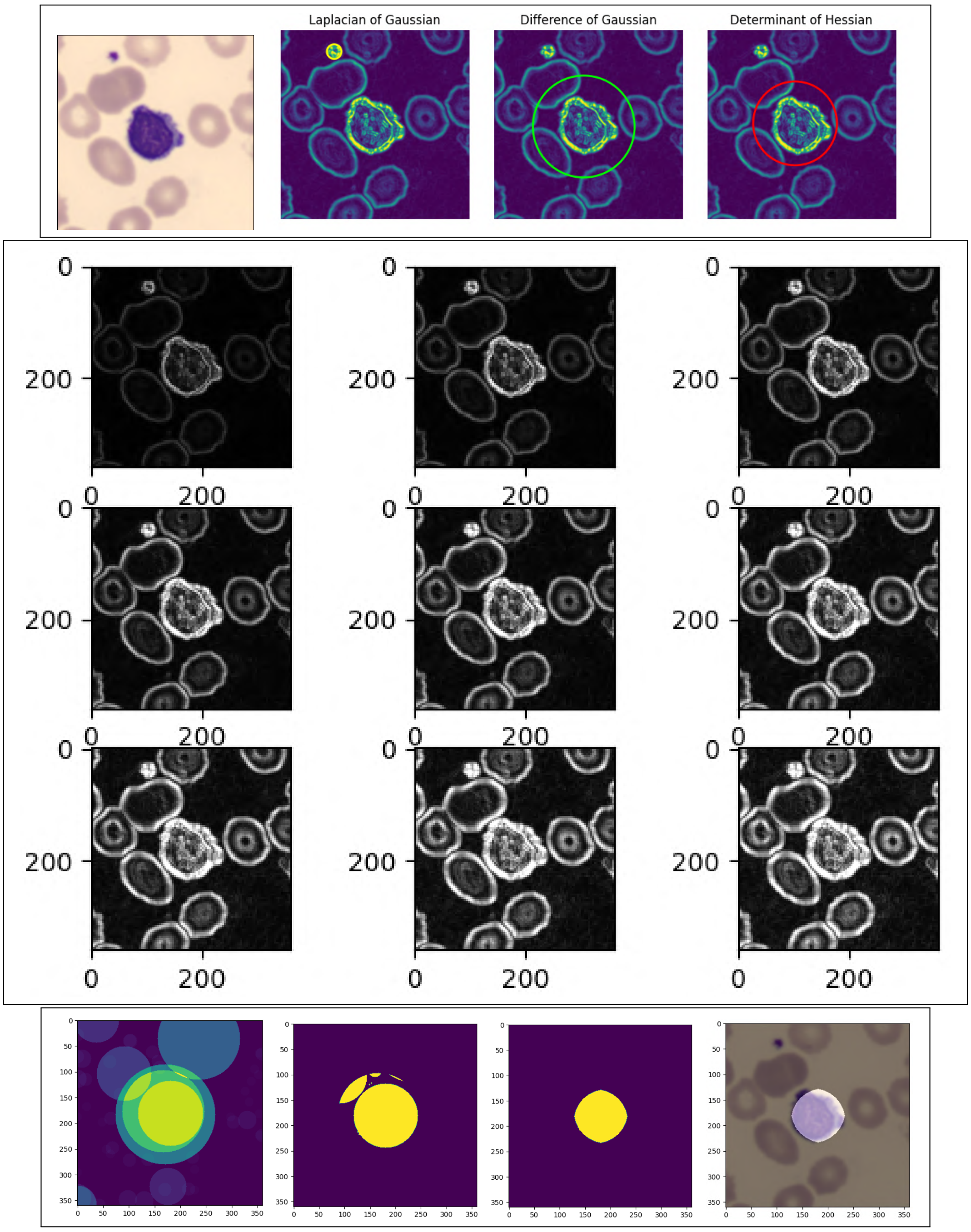


Figure 3.3: From top left to bottom right: A sample of raw image (Lymphocyte) from PBC_dataset_normal_DIB dataset dataset after applying Gaussian smoothing with a filter size of (3,3), using blob detectors of three types, i.e., Laplacian of Gaussian (LoG), Difference of Gaussian (DoG), and Determinant of Hessian (DoH), images after applying Sobel filter of different deltas and sigma, the stacked layers of confidences, applying thresholding on stacked layers of confidence, after repeated erosion on the resulting confidences, the final mask obtained superimposed on the input image.

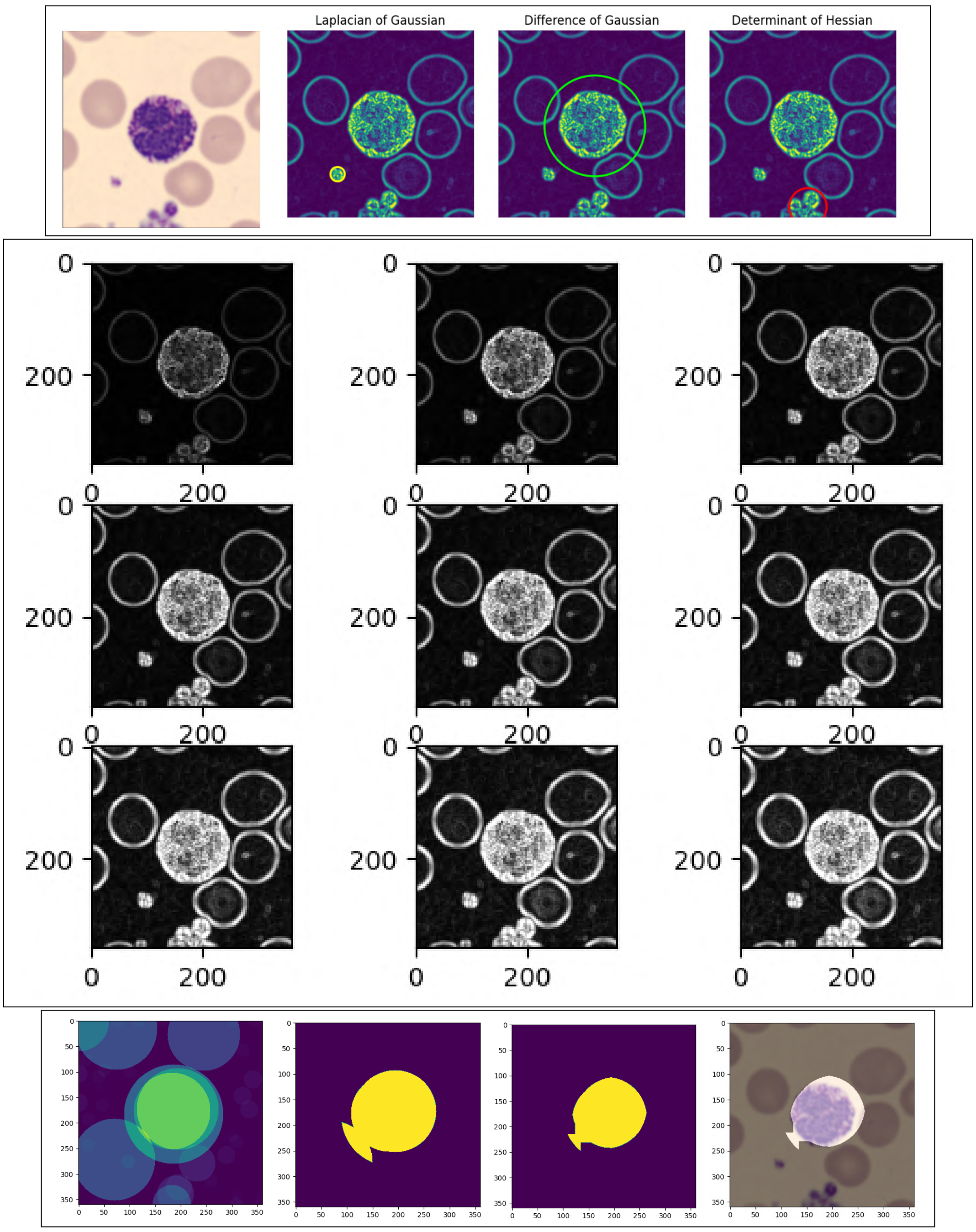


Figure 3.4: From top left to bottom right: A sample of raw image (Basophil) from PBC_dataset_normal_DIB dataset after applying Gaussian smoothing with a filter size of (3,3), using blob detectors of three types, i.e., Laplacian of Gaussian (LoG), Difference of Gaussian (DoG), and Determinant of Hessian (DoH), images after applying Sobel filter of different deltas and sigma, the stacked layers of confidences, applying thresholding on stacked layers of confidence, after repeated erosion on the resulting confidences, the final mask obtained superimposed on the input image.

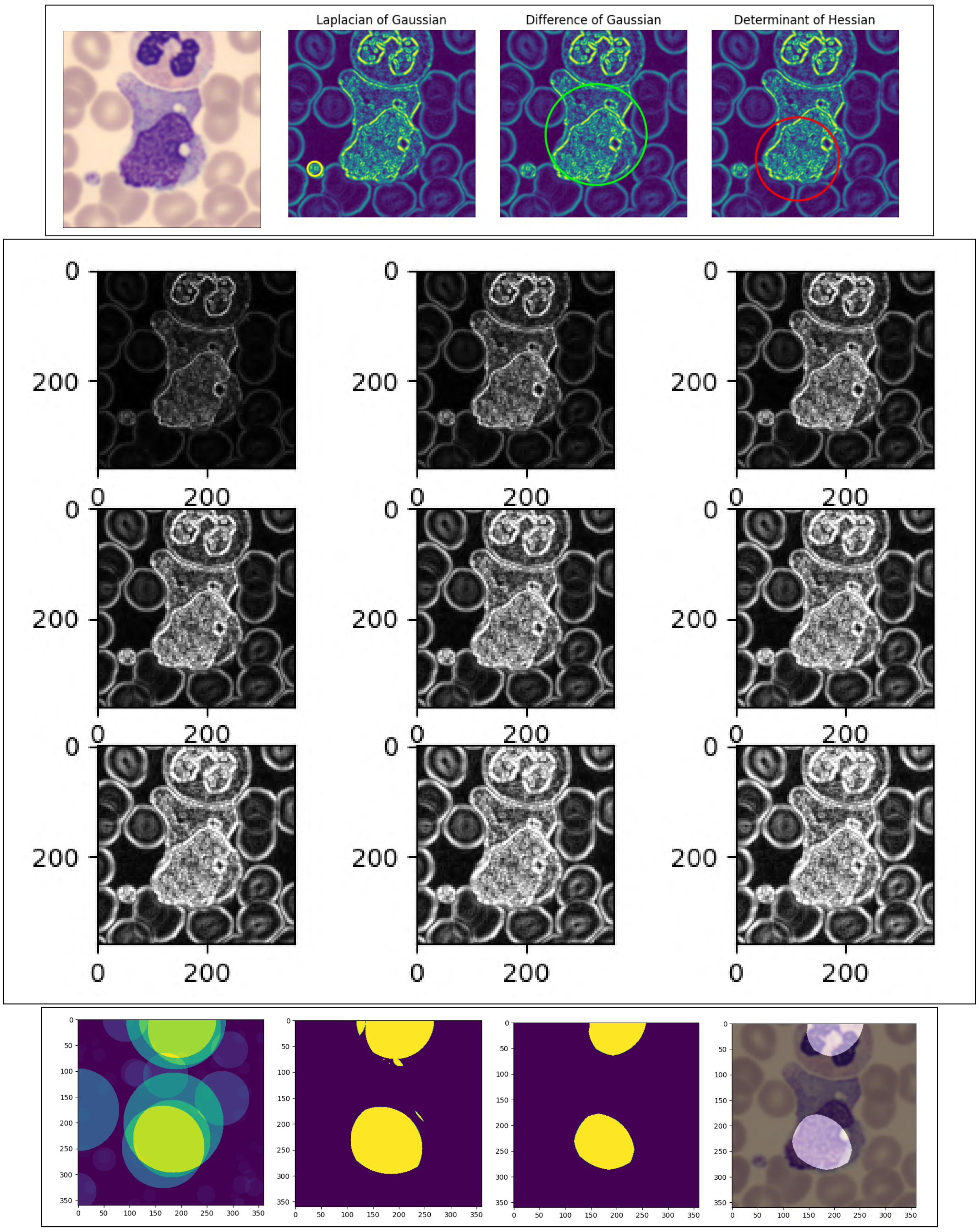


Figure 3.5: From top left to bottom right: A sample of raw image (Monocyte) from PBC_dataset_normal_DIB dataset dataset after applying Gaussian smoothing with a filter size of (3,3), using blob detectors of three types, i.e., Laplacian of Gaussian (LoG), Difference of Gaussian (DoG), and Determinant of Hessian (DoH), images after applying Sobel filter of different deltas and sigma, the stacked layers of confidences, applying thresholding on stacked layers of confidence, after repeated erosion on the resulting confidences, the final mask obtained superimposed on the input image.

Figure 3.3 shows an implementation of the algorithm, the first image is a smoothened version of the original image for which the Sobel filter is applied on as shown in black and white. After applying sobel filter, we apply different blob detectors as shown in the second image from top left. We then get some layers of confidences. The confidences are then threshold to get the actual possible mask of the highest layer of confidences. The confidence is then eroded to get a much better mask, and it is applied on the original image to mask the relevant cell portion. In this example, the algorithm performs relatively well.

Another example is shown in Figure Figure 3.4 which also does relatively well, but the example shown in Figure 3.5 is unable to segment the whole cell region, which shows the incapability of the above method on diverse image samples. Probably we could have tried different innovative techniques to segment the cell, but every time we need to hardcode or change the algorithm to perform well on different samples of the slide. It may even happen that the same method does not perform well on another domain's dataset where the slide is of a different color and texture.

3.3 GRAD CAM based novel method for masking

GRAD-CAM [20] is a method for producing visual explanations from a large class of Convolutional Neural Network models that helps to make them more tansparable and explainable. It was mainly built to understand why neural network models failed by providing visual explanations to the confidence that the model sees. It can be added to any convolutional layers to check the gradients which are concentrated on a certain region which can be traced back to the input image. It was mainly designed to check the transparency of visual question answering systems.

It generates explanations from deep networks via gradient localization. A sample of the explaination is shown in Figure 3.6. It is robust to adversarial perturbations. It is more faithful to underlying models. It helps to achieve model generalization by identifying underlying bias and helps to understand what the model sees. The authors recommend to use the deeper layers for this particular method, since it captures more semantic concepts and is used to get better results.

We have utilized this functionality of GRAD-CAM to get the confidence segmen-

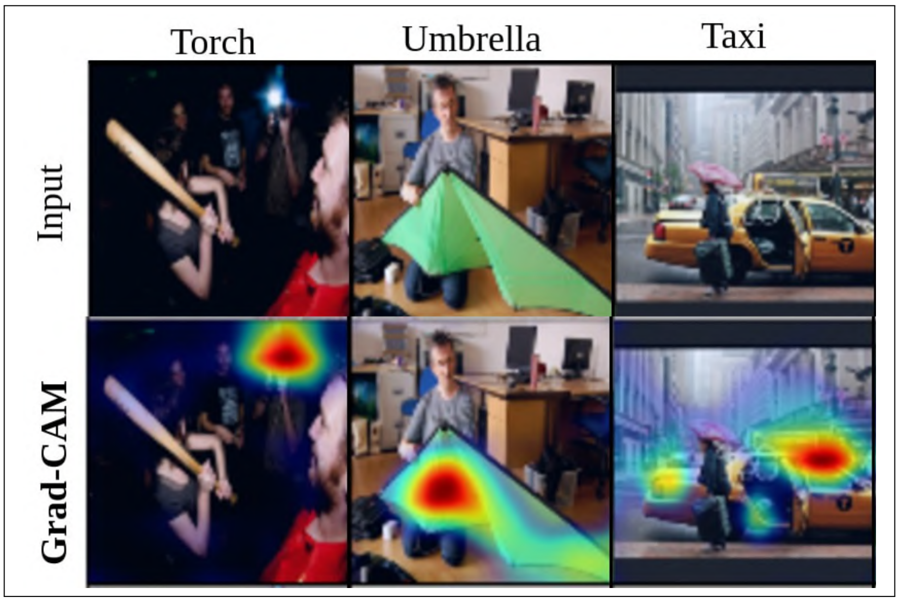


Figure 3.6: Examples of GRAD-CAM in visual question answering systems. When focused on a particular word, the model learns to focus the gradient on a particular part of the picture. (Picture Courtesy: Reproduced with permission from Ramprasaath R. Selvaraju.)

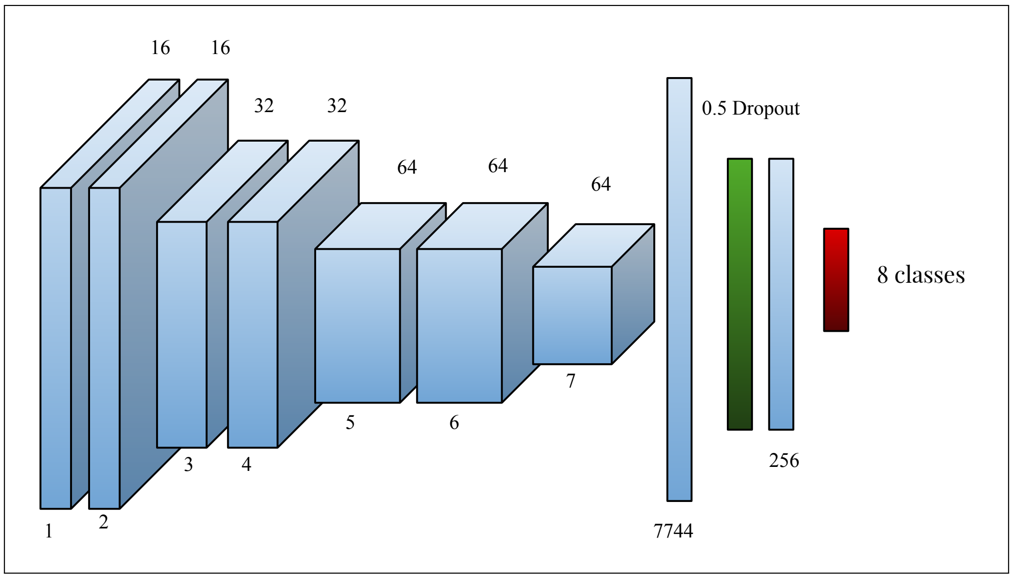


Figure 3.7: A 11-layer Deep CNN model to get the mask from each of the convolution blocks to segment the cells present in the PBC_dataset_normal_DIB dataset.

tation mask from each layer of the proposed network. The proposed model is a simple 11-layer network for deriving the segmentation mask from the PBC_dataset_normal_DIB dataset is shown in Figure 3.7. We get the confidence score from each of the convolutional layers. We add each of the masks and we threshold the mask by the maximum value obtained from the layered masks. After thresholding we apply a dilation via a box kernel of size 5x5.

Hence, a simple classification model will be used to get masks for each of the images of the PBC_dataset_normal_DIB dataset. This is a novel method, since there is very few related works on this like detecting fetus [12], and there is a huge scope of improvement on this proposed method. The working version of the proposed method is shown in Figure 3.8 which does a relatively good job. Similarly another example shown in Figure 3.9 does even better job in segmenting the image.This may not give an exact accurate segmentation mask but we can get an idea of where the cell might be located from just a classification model.

We later found out that taking the confidence score from only layers 3, 4 and 7 is good as well, which also results in saving some computation. The comparison of the two segmentation mask acquiring technique is shown in Figure 3.10. There might be some noise in the captured image, but it might be relevant when there are a lot of images present. This method is just a prototype for converting classification model to segmentation model and hence there is a lot of scope for improvement. Figure 3.11 left shows a good segmented sample, while on the right shows a sample which is not segmented good.

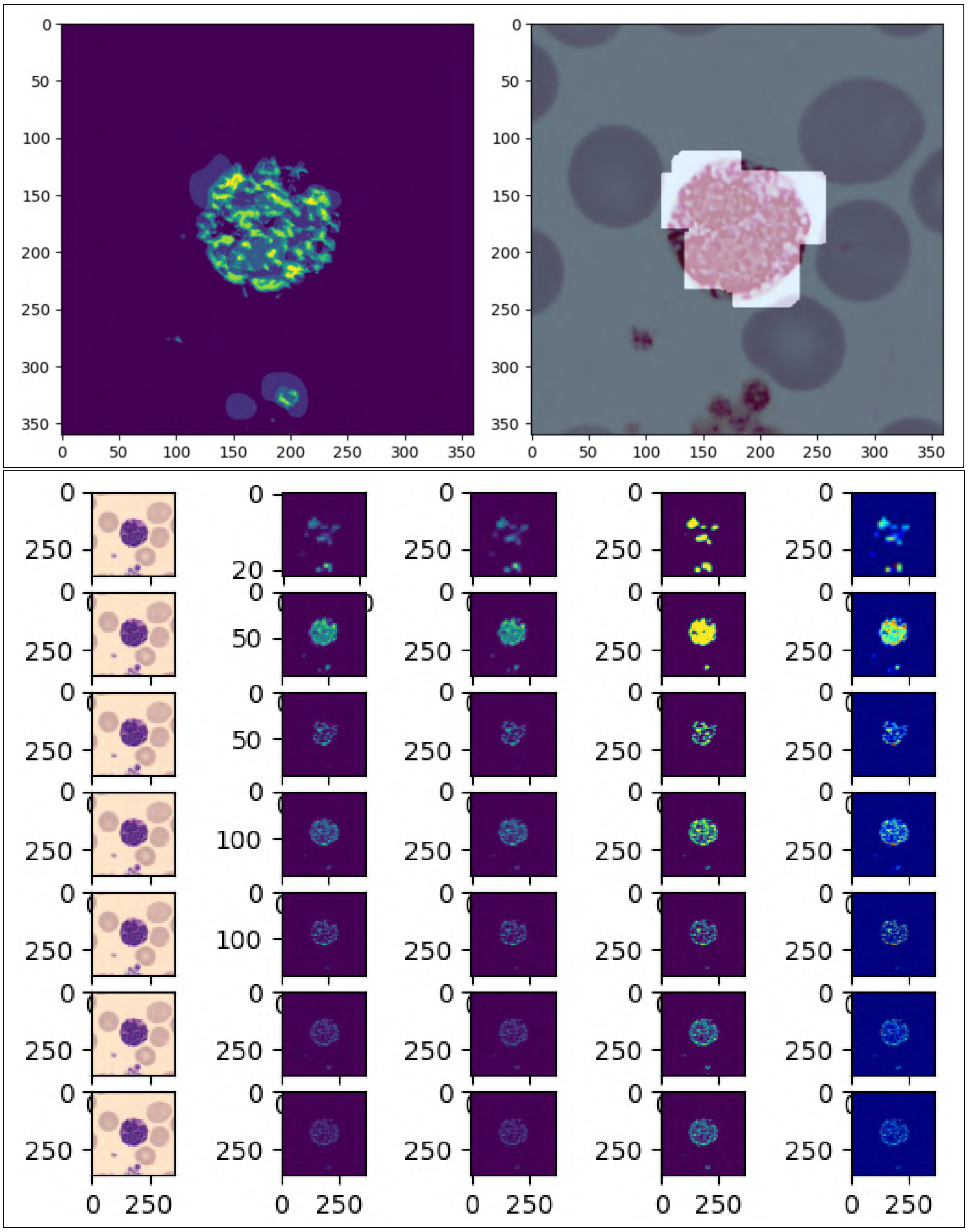


Figure 3.8: The steps involved in creating masks via the PBC_dataset_normal_DIB slide images. The images on bottom shows the individual confidence masks that is created for each convolutional layer of the model, the image on the top left shows the assembled images of confidence masks and the image on the top right is the masked original image with an approximate mask covering the cell.

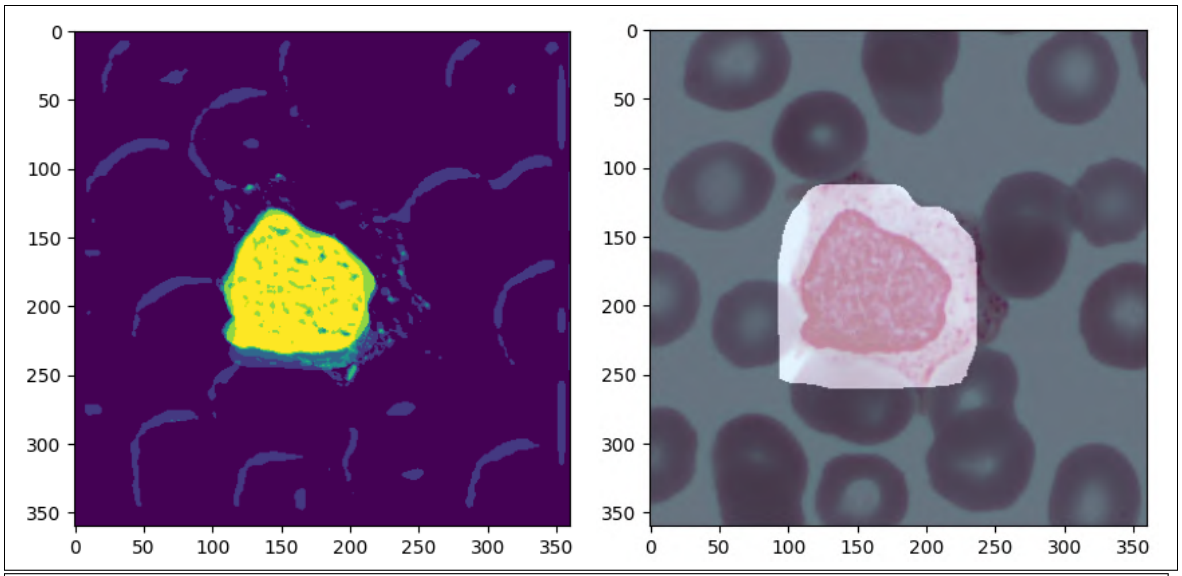
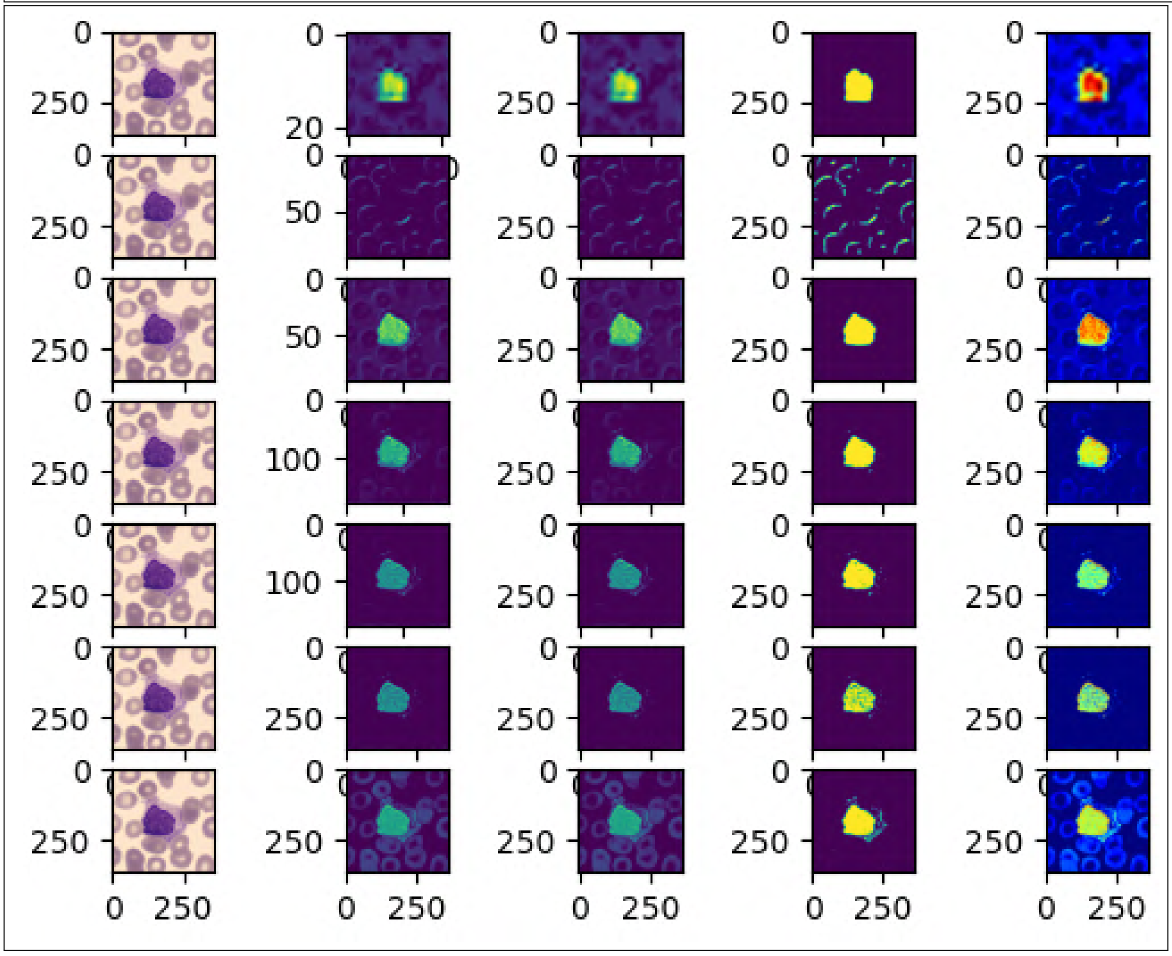



Figure 3.9: The steps involved in creating masks via the PBC_dataset_normal_DIB slide images. The images on bottom shows the individual confidence masks that is created for each convolutional layer of the model, the image on the top left shows the assembled images of confidence masks and the image on the top right is the masked original image with an approximate mask covering the cell.

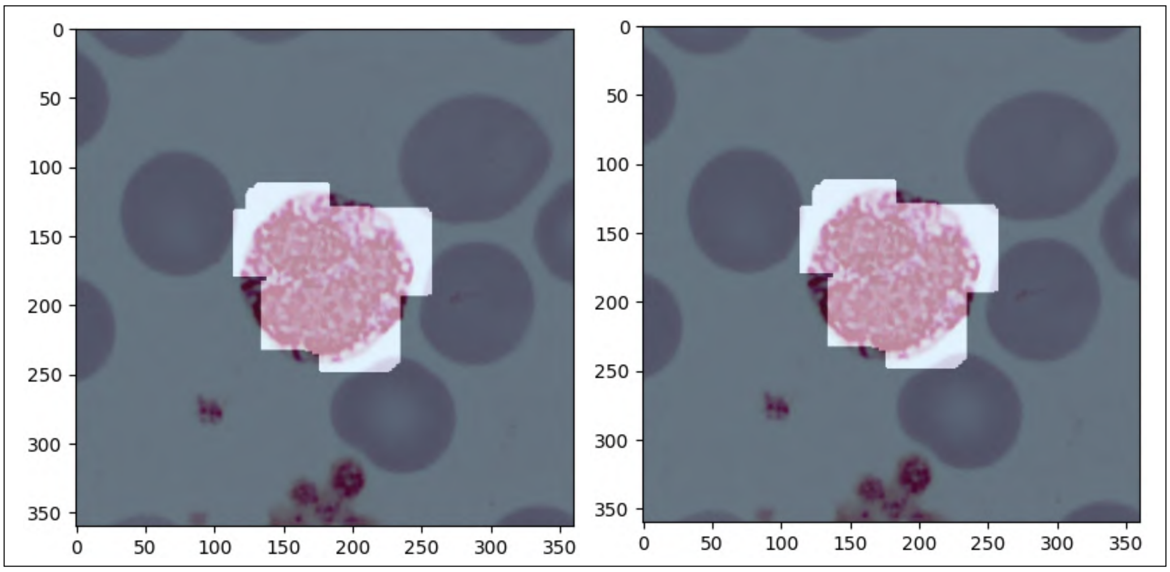


Figure 3.10: Left: From all the layers, Right: From layers 3,4 and 7.

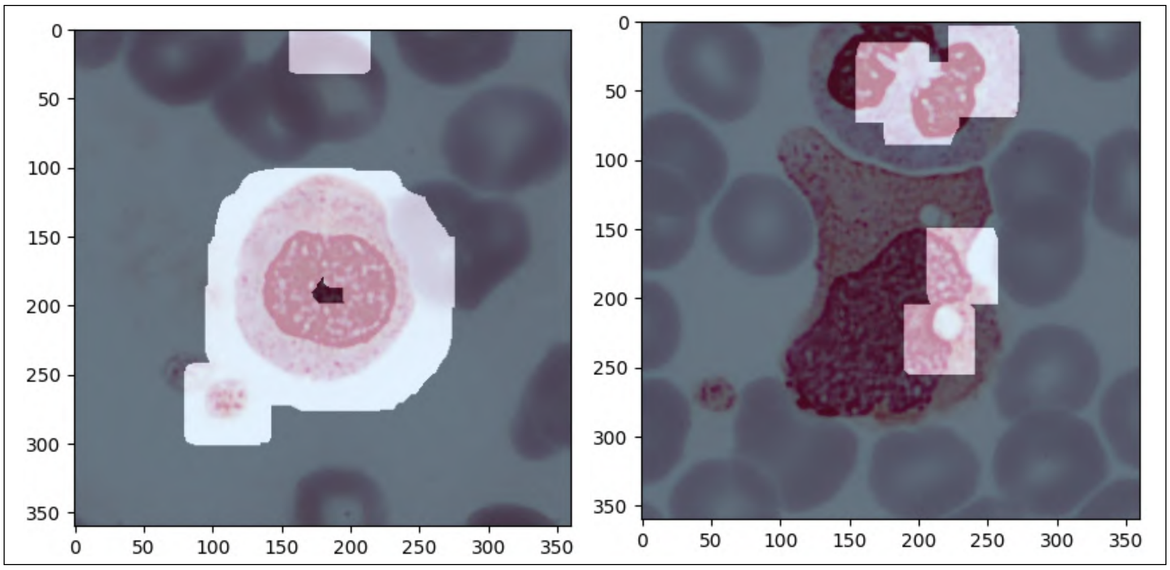


Figure 3.11: Images segmented by taking confidence scores from layers 3, 4 and 7.

Chapter 4

Classification

4.1 Classification models applied on PBC datasets

We have performed classification on PBC_dataset_normal_DIB dataset as well as PBC_dataset_normal_DIB_cropped dataset. Selecting a classifier is a rigorous job since we have to check all the related literature for doing exhaustive search on which model the classifier performs best in the test case. We have created a seed of 42 using NumPy's random function for reproducibility and fair testing of all the models in the given dataset. The original paper [3] was using about 80% as training set and the rest 20% as test set. Among the 80% of training set the authors were using 20% as validation set. So, they were splitting 64-16-20 as training-validation-test set.

The style of architecture that we used for testing the state-of-the-art related literature's backbones (standard model) is shown in Figure 4.1. The architecture is simple, we use the convolutional layers and do a Global Average Pooling at the end of convolutional layers, and add a few dense layers at the end of them. These fully connected layers are 128-32-8, where 8 is the number of classes to be classified through SoftMax activation function.

We have kept this ratio for getting the result using the standard datasets, upon which we have surpassed the state of the art as reported in the original paper. For transparency, we have not changed the seed and we reported the results on 20% of the whole dataset as the test set without data augmentation. The application of various state of the art models on the PBC_dataset_normal_DIB_cropped and

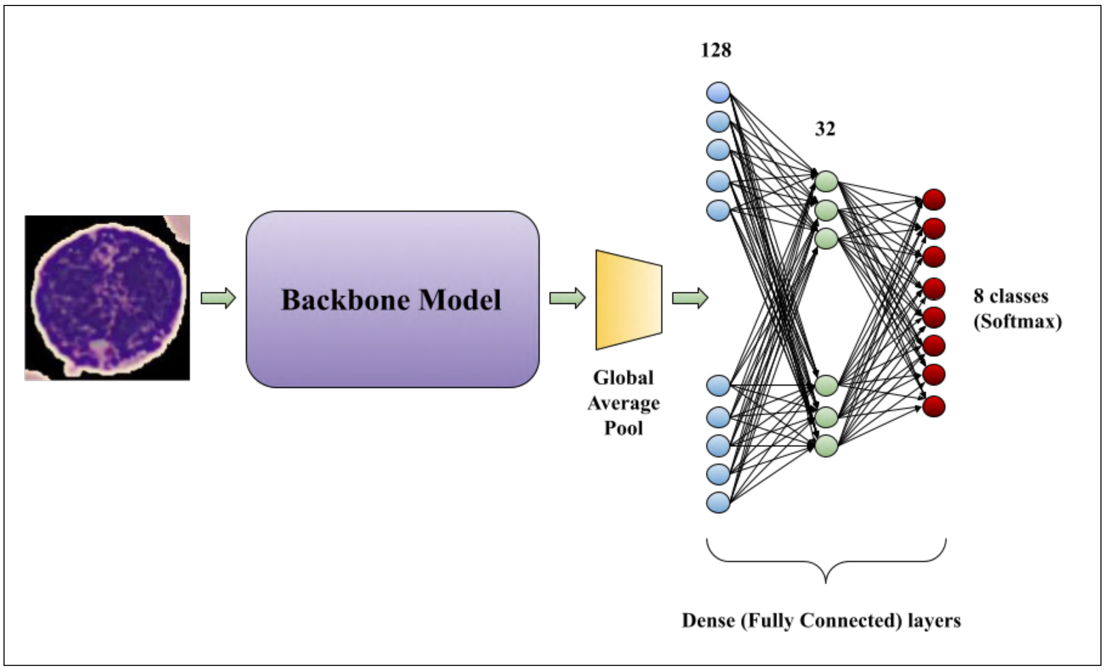


Figure 4.1: Basic backbone that is used in building the classification models, A Global Average Pooling is used after all the Convolutional backbone with Fully Connected layers and at last a output layer of 8 classes with SoftMax as activation.

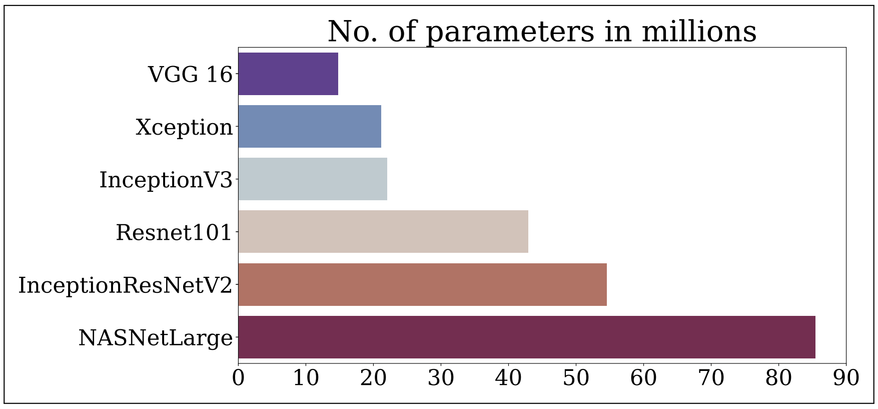


Figure 4.2: The number of parameters in million for different models. VGG16 with 14.78 M, Xception with 21.12 M, InceptionV3 with 22.06 M, Resnet101 with 42.92 M, InceptionResNetV2 with 54.53 M, and NASNetLarge with 85.43 M parameters.

PBC_dataset_normal_DIB datasets and evaluating them on Precision, Recall and F1-score on the test set, are shown in Table 4.1.

Model Name	Dataset	Type	Precision	Recall	F1-score
Inception V3 [22]	PBC_cropped	fully trained	96.01	95.74	95.70
	PBC_cropped	freezed	91.15	91.15	90.97
	PBC_dataset	fully trained	98.73	98.88	98.89
	PBC_dataset	freezed	91.61	91.42	91.28
Inception-ResNetV2 [21]	PBC_cropped	fully trained	98.84	98.99	98.82
	PBC_cropped	freezed	93.73	93.90	93.66
	PBC_dataset	fully trained	**99.02**	**99.07**	**98.93**
	PBC_dataset	freezed	93.60	93.41	93.45
NASNetLarge [26]	PBC_cropped	fully trained	96.63	96.51	96.64
	PBC_cropped	freezed	92.51	92.38	92.39
	PBC_dataset	fully trained	98.84	98.92	98.75
	PBC_dataset	freezed	93.63	93.67	93.78
VGG16 [16]	PBC_cropped	fully trained	98.49	98.29	98.22
	PBC_cropped	freezed	95.03	94.85	95.05
	PBC_dataset	fully trained	98.73	98.78	98.69
	PBC_dataset	freezed	93.60	93.23	93.27
Xception [5]	PBC_cropped	fully trained	98.96	98.81	98.75
	PBC_cropped	freezed	93.59	93.46	93.29
	PBC_dataset	fully trained	98.84	98.81	98.75
	PBC_dataset	freezed	91.51	91.19	91.27
Resnet101 [10]	PBC_cropped	fully trained	98.37	98.56	98.40
	PBC_cropped	freezed	71.05	68.95	68.47
	PBC_dataset	fully trained	98.84	98.92	98.75
	PBC_dataset	freezed	68.87	70.22	68.03

Table 4.1: Application of various state of the art models on PBC_dataset_normal_DIB and PBC_dataset_normal_DIB_cropped datasets with Adam Optimizer and batch size of 16 and learning rate of 1e-4 on test set. (Only NASNetLarge has a batch size of 8)

There are various classifier which have been tried and experimented, and the parameters (in million) of each of them are shown in Figure 4.2. The table 4.1 shows some interesting properties, when the models are trained with just the freezed parameters, i.e., with no fine tuning, the PBC Cropped dataset does a relatively better job in classifying in most of the models. The difference between fine-tuning and freezed is that during freezed, we are only training and updating the weights of the fully

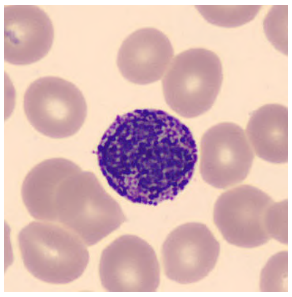


Figure 4.3: Image which is passed to VGG16 for generating features as shown in Figure 4.6.

connected layers, whereas in fine-tuning, we are training the whole network, i.e., not just using ImageNet's weight as convolutional backbone, but retraining to learn specific features according to the problem (i.e., identifying cells for classification). Whereas in fully trained version we can clearly see that almost all the models do a better job in classifying the whole slide. This may happen due to the fact that the size information is lost when we are cropping and resizing the data, also, since we are cropping out the cells, there might be some relevant texture information around the cell which the model finds helpful in classifying the cell. But for the detection part, the cells are almost of same size, we need to crop the masked cell out and then classify it to see which class it belongs to.

In Figure 4.4, we see what sort of features the model learns to respond to. The figure on the left shows the ImageNet weights that are learnt by the initial model [25] which we are using for transfer learning and the figure on the right shows the initial weights fully trained on the PBC_dataset_normal_DIB dataset, hence helping the model to classify specialized data, restricting and shifting the feature space. The way we make these visualizations is simple, we create input images that maximizes the activations of a specific activation on a target layer. Such images represent the visualization of patterns a filter responds to. We do gradient accent to compute the loss for the image which takes the image to a state which respond to the activation function more strongly. These images are selected from **block2_conv2** and **block5_conv3** of each of the models. The feature maps obtained by the first and last convolutional layer of training VGG16 model on an image shown in Figure 4.3 is shown in Figure 4.6.

From Table 4.1, it is clear that InceptionResNetV2 performs better than most of the backbone models in classifying the PBC_dataset_normal_DIB and PBC_cropped datasets, hence giving a Precision of 99.02, a Recall of 99.07 and a F1-score of 98.93 in PBC_dataset_normal_DIB dataset. The confusion matrix of each of the model over the same test set is shown in Figure 4.5. The only mistake the models are making are predicting Lymphocytes as Erythorblast.

The training and validation graphs obtained by all the five models are shown in Figure 4.7 for cropped dataset and Figure 4.8 for full dataset respectively. We have collected the Accuracy, Loss, AUC, Precision and Recall for all the models, and we see that the validation graph might have a bit of kinks due to the rough contour of the higher dimensional loss. This generally happens when there are outliers in the data which does not help in optimizing using mini-batch gradient descent [1]

4.2 Classification models applied on Smear Slides Cropped datasets

Model Name	**Precision**	**Recall**	**F1-score**
Inception V3	77.41	75.57	74.44
Inception-ResNetV2	81.07	76.91	74.93
NASNetLarge	61.79	54.37	51.29
VGG16	73.54	72.72	72.62
Xception	**82.15**	**78.80**	**76.96**
Resnet101	72.94	76.84	74.39

,

Table 4.2: Application of various state of the art models on Smear Slide Cropped dataset with Adam Optimizer and batch size of 16 and learning rate of 1e-4 on test set. (Only NASNetLarge has a batch size of 8)

Since from Table 4.1, we can see that the best performing models are fully trained version of weights that were trained on ImageNet dataset, so we will be saving computations and time for only finding the results for the fined-tuned version of the

[1]https://stats.stackexchange.com/questions/303857/explanation-of-spikes-in-training-loss-vs-iterations-with-adam-optimizer

model. For the preceding experiments, we have kept the similar proportion in creating the train-validation-test split, i.e., using about 80% of the whole dataset as training set and the rest 20% as test set. Among the 80% of training set they were using 20% as validation set. Hence 64-16-20 of the whole datasets as the training-validation-test set. The architecture of the backbone model is same as shown in Figure 4.1.

The results obtained can be seen in Table 4.2. We have ensured the proper and same division of training-validation-test images by applying seed to the shuffled images. From the two tables, i.e., Table 4.1 and Table 4.2, we can clearly see that the Xception Backbone performs better than all of the models, in PBC cropped and Smear Slides Cropped dataset.

Since the dataset have a smaller number of images (762 images), as compared to PBC Dataset (17092 images), for this reason, the validation graphs will have turbulence when converging. The graphs of training and validation metrics are shown in Figure 4.10. The validation metrics do converge since, we are using the state-of-the-art models, which can adapt to difficult datasets, furthermore we are fine-tuning the already trained model on ImageNet dataset, for this reason, the accuracy is quite good, without data augmentation. The confusion matrix is shown in Figure 4.9. From the figure, it looks like NASNetLarge have visually not so good confusion matrix as compared to the other models, also in Table 4.2 we can see the same result.

For the evaluation of performance for the Domain Adaptation pipeline, we use PBC cropped and the Smear Slides Cropped datasets, and Xception performs collectively well in each of the datasets. The results after applying and studying different experiments with the domain adaptation part is described in Chapter 6.

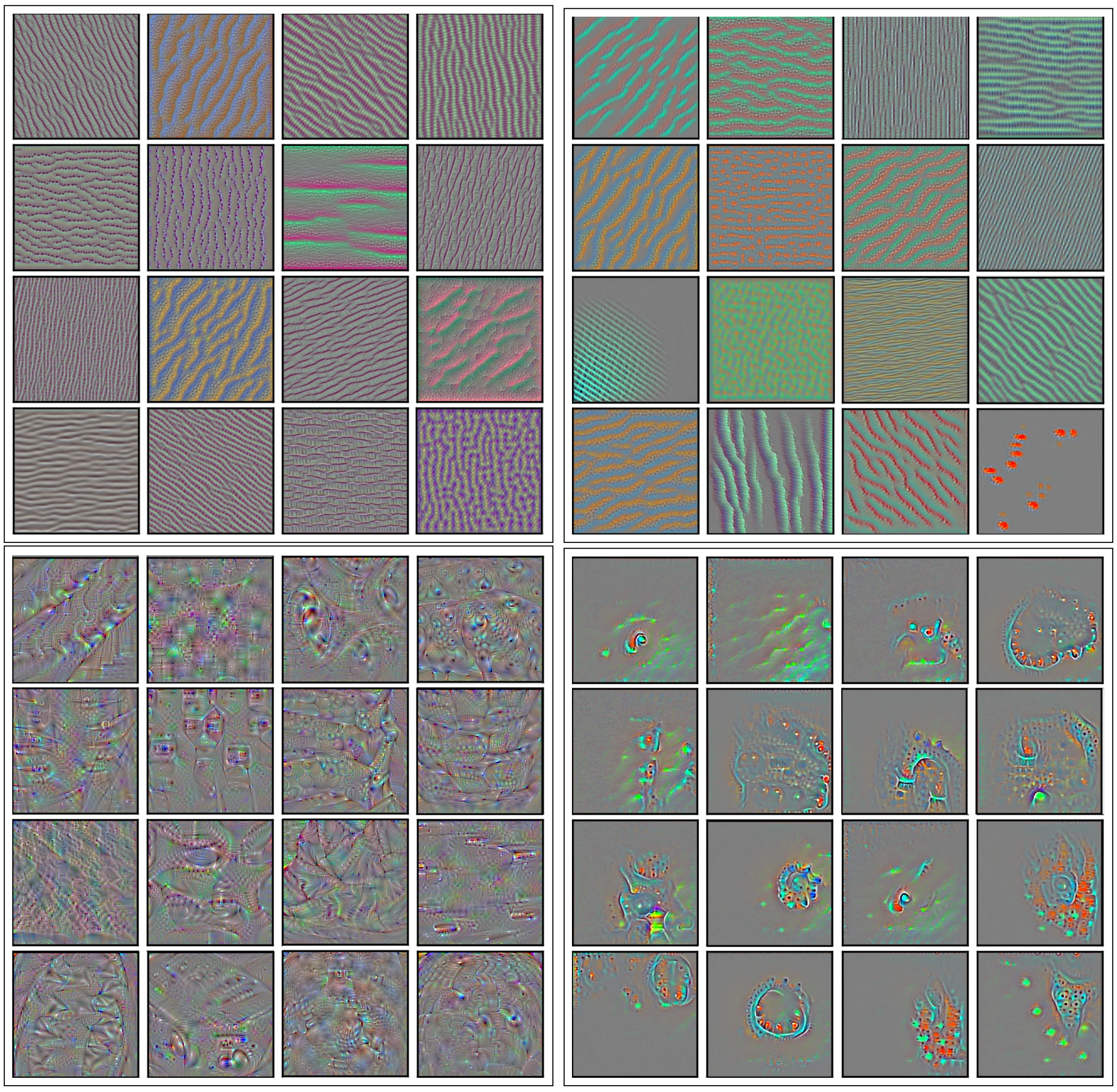


Figure 4.4: An illustration showing the patterns that the selected layers of the VGG16 model learns (top : **block2_conv2** layer and bottom: **block5_conv3** layer), the left figures show the freezed version of the weights, i.e., the model trained on only ImageNet dataset capturing the natural imagery's features, and the right figures shows the fully trained version of the weights, i.e., the features that are modified to identify only cell images, hence restricting to a specialized problem.

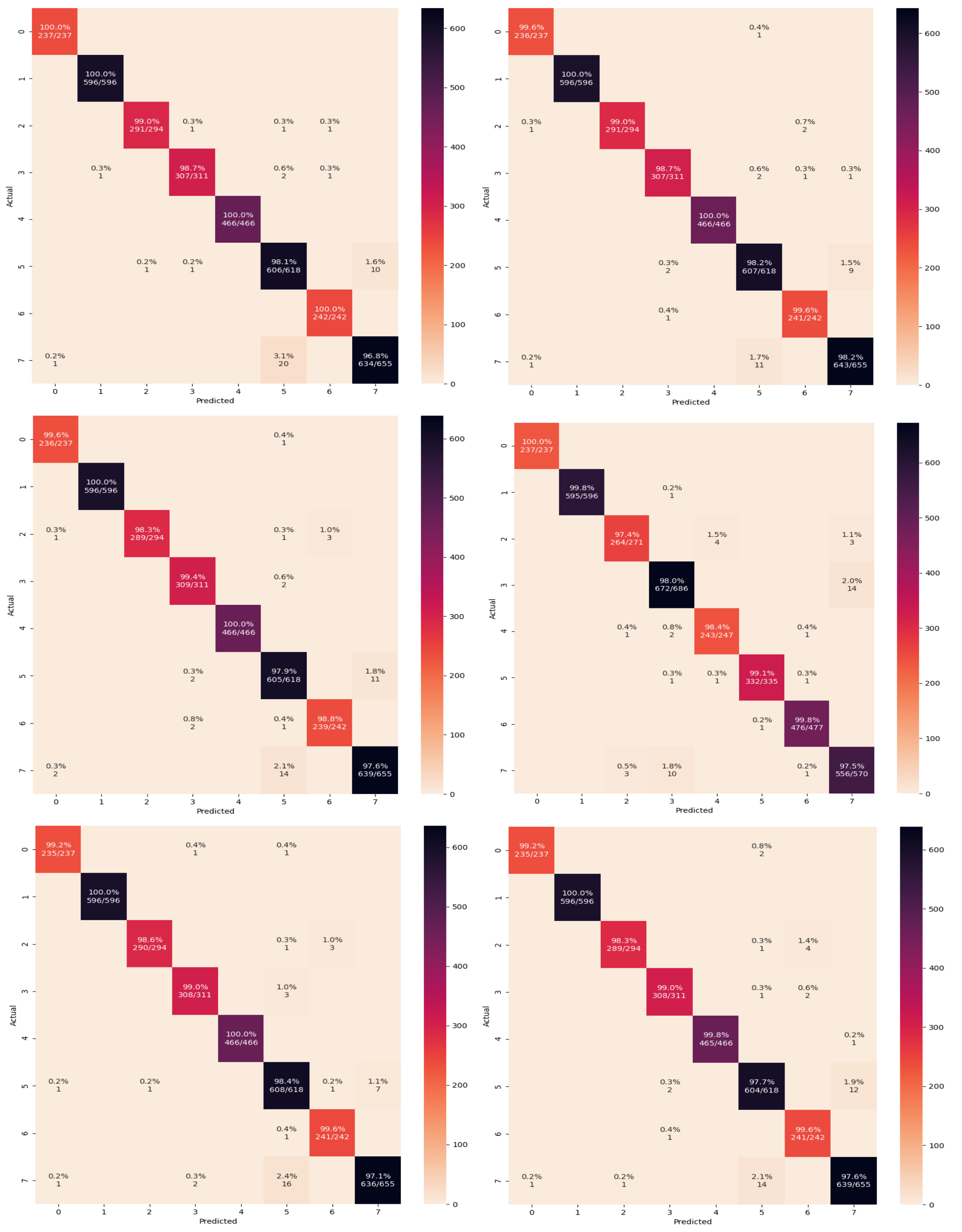


Figure 4.5: Confusion matrix of the best performing models i.e., From top left to bottom right: Inception V3, InceptionResNetV2, NASNetLarge, VGG16, Xception and Resnet101 in each of the respective datasets. Here, the corresponding labels are: 0 (Eosinophil), 1 (Neutrophil), 2 (Monocyte), 3 (IG - Inactive Granulocytes), 4 (Basophil), 5 (Erythroblast), 6 (Platelet), 7 (Lymphocyte). The names were used as numbers due to lack of space.

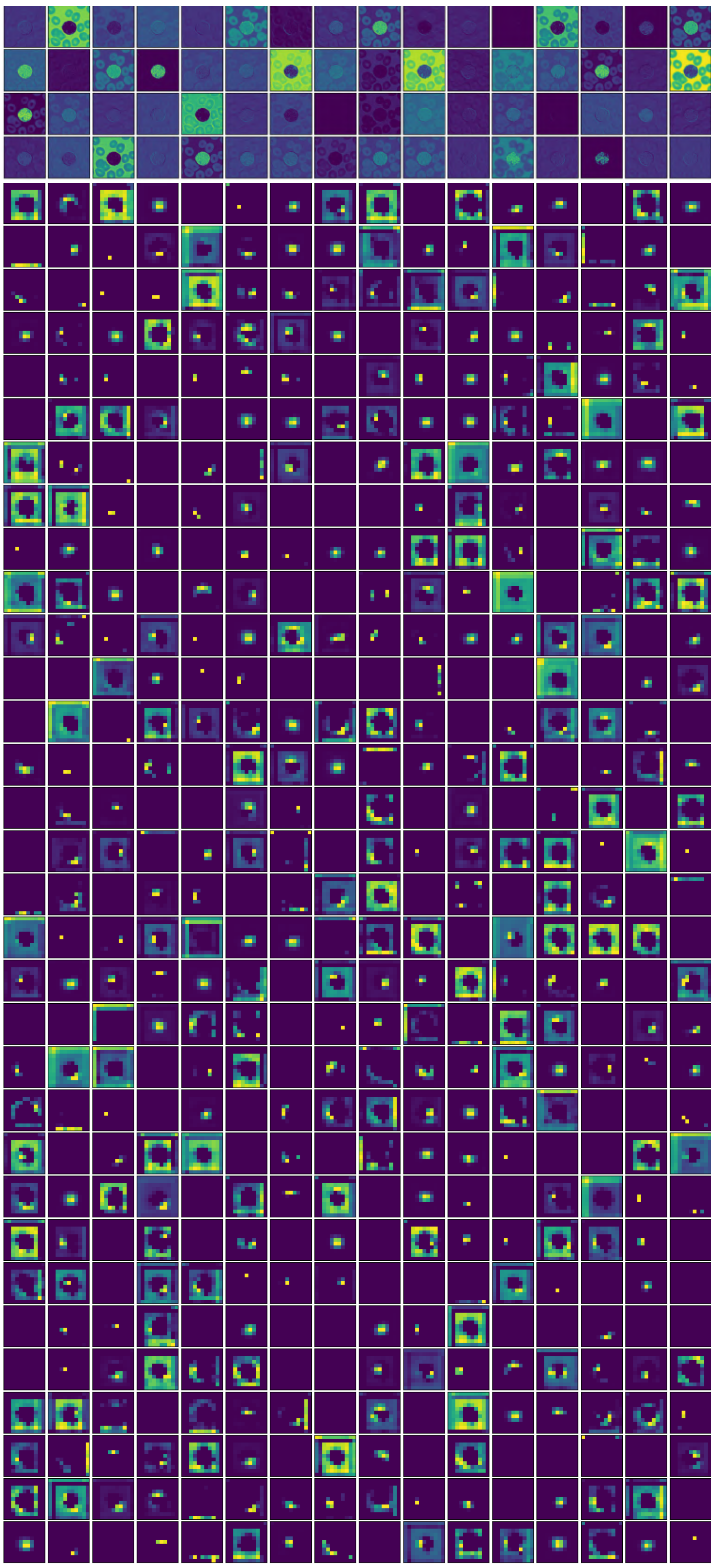


Figure 4.6: The feature maps that were generated by the layer 1 (64 filters, up) and layer 18 (512 filters, bottom) of VGG 16 architecture provided by Keras.

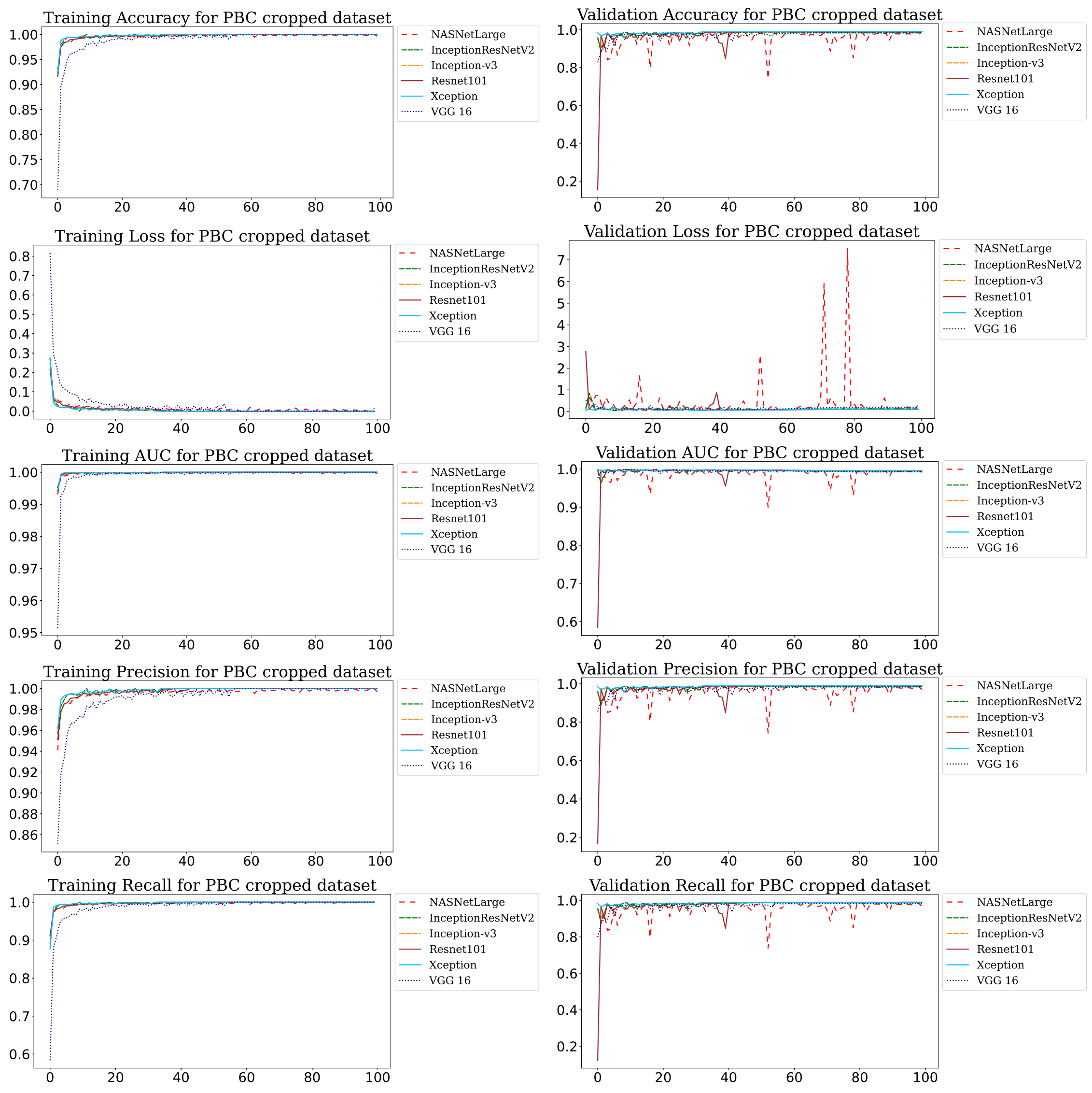


Figure 4.7: The graphs obtained by training and validation of PBC cropped dataset by the application of all the five models, i.e., VGG16, Xception, InceptionV3, Resnet101, InceptionResNetV2, NASNetLarge (by doing fine-tuning). From top to bottom, training and validation metrics on Accuracy, Loss, AUC (Area Under Curve), Precision and Recall.

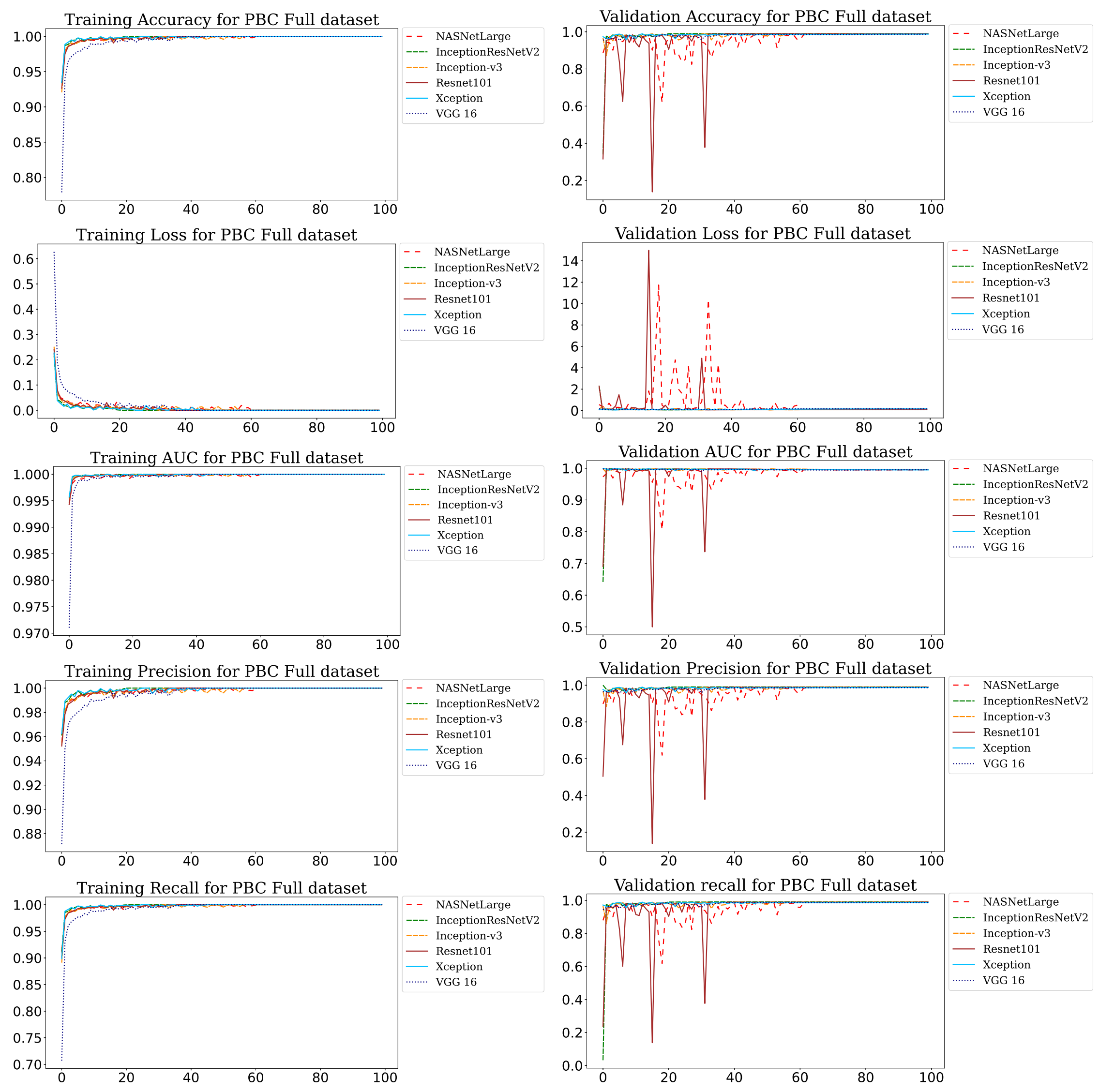


Figure 4.8: The graphs obtained by training and validation of PBC full dataset by the application of all the five models, i.e., VGG16, Xception, InceptionV3, Resnet101, InceptionResNetV2, NASNetLarge (by doing fine-tuning). From top to bottom, training and validation metrics on Accuracy, Loss, AUC (Area Under Curve), Precision and Recall.

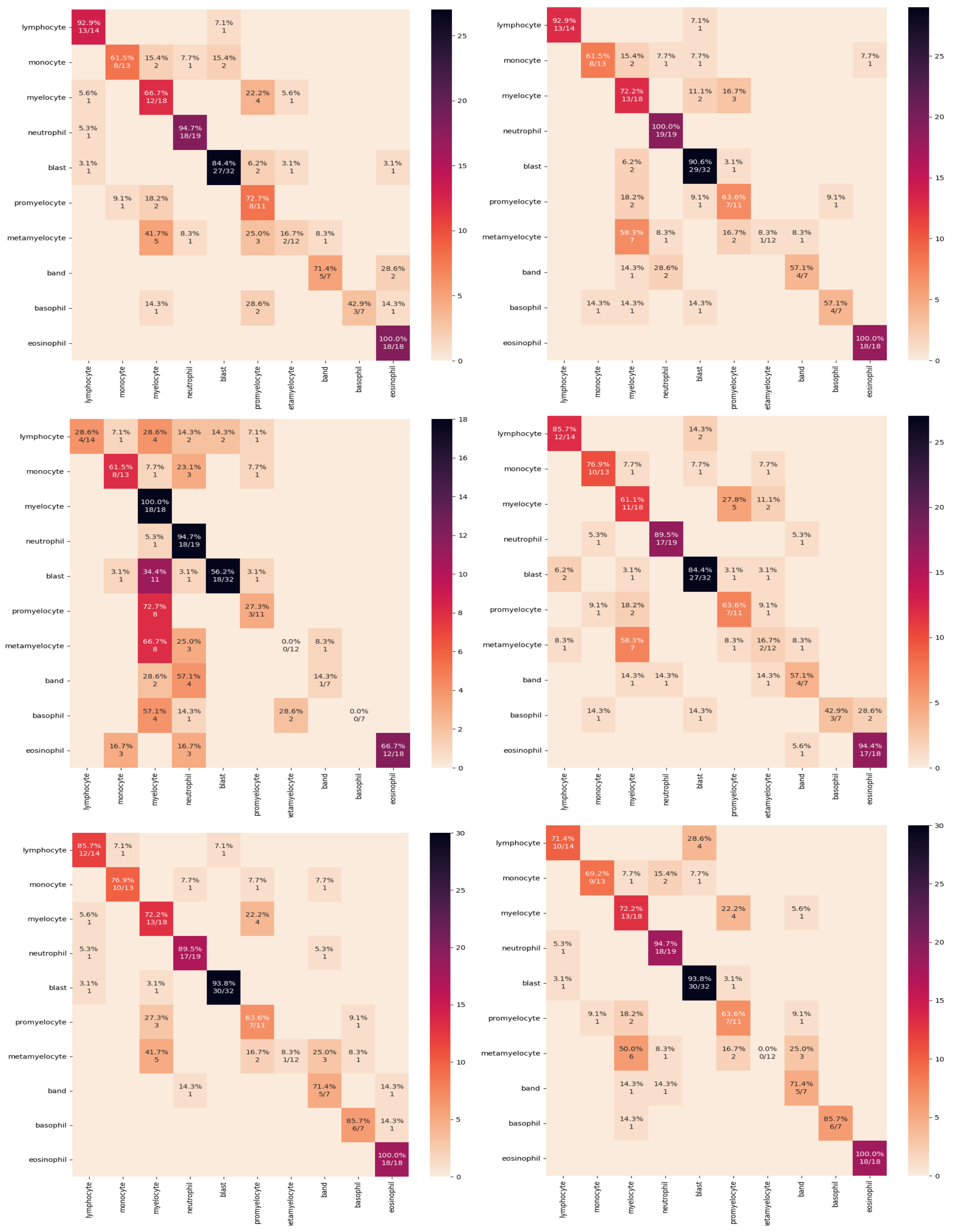


Figure 4.9: Confusion matrix of the fully trained models i.e., From top left to bottom right: Inception V3, InceptionResNetV2, NASNetLarge, VGG16, Xception and Resnet101 in each of the respective datasets.

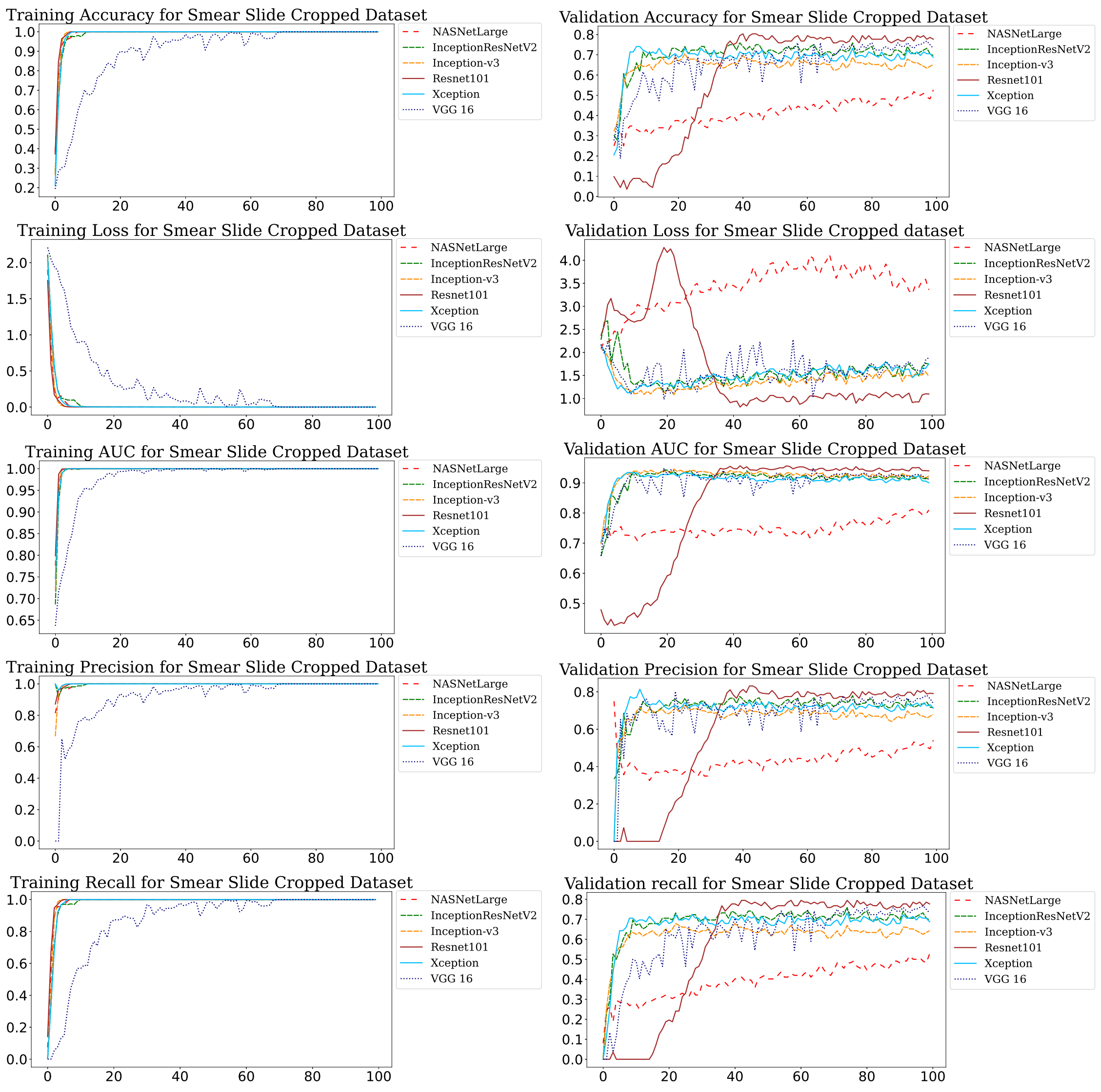


Figure 4.10: The graphs obtained by training and validation of Smear 10 Cropped (10) dataset by the application of all the five models, i.e., VGG16, Xception, InceptionV3, Resnet101, InceptionResNetV2, NASNetLarge (by doing fine-tuning). From top to bottom, training and validation metrics on Accuracy, Loss, AUC (Area Under Curve), Precision and Recall.

Chapter 5

Mask RCNN

5.1 What is Instance segmentation?

Instance segmentation deals with recognizing and understanding what is in the image in pixel level. It deals with assigning different colors to different instances of the same object. In figure 5.1 (left), different instances of balloons are assigned different colors. This is a hard task since objects might be occluded and it is necessary to find different parts of the same object. Here, in Figure 5.1 (right), a chair is occluded but for a good instance segmentation algorithm, it is necessary to recognize this object correctly and assign the necessary color.

Mask RCNN [9] [1] returns object's bounding boxes, classes present, and precise masks of the objects for an input image, hence doing instance segmentation. It uses a convolutional backbone, i.e., FPN (Feature Pyramid Network), for preserving features at different scales. It is based upon a Faster RCNN [18] backbone, so it uses its architecture for getting the features from a given image. The architecture uses RNP (Region Proposal Networks) which contains bounding boxes, also known as anchors, which helps to detect objects faster without searching the whole image. The main contribution of the network is it uses ROI Align to align the features at different scales using bilinear interpolation method, which helps to remove location misalignment caused due to ROI pooling, hence, significantly increasing performance in getting segmentation masks near to the ground truth. The architecture of Mask RCNN is shown in Figure 5.2.

[1]**Collaborators: Aniket Bhattacharyea (aniketmail669@gmail.com)**

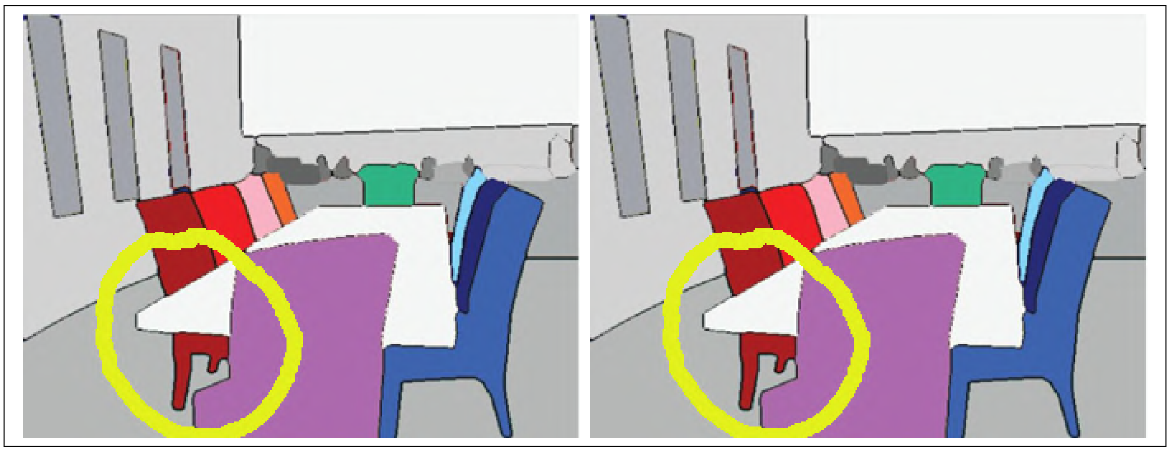


Figure 5.1: Left: Instance segmentation of balloons, where each balloon is uniquely identified and assigned different colors. Right: Instance segmentation deals with assigning same color to even occluded parts of the same object, which might not be connected directly.

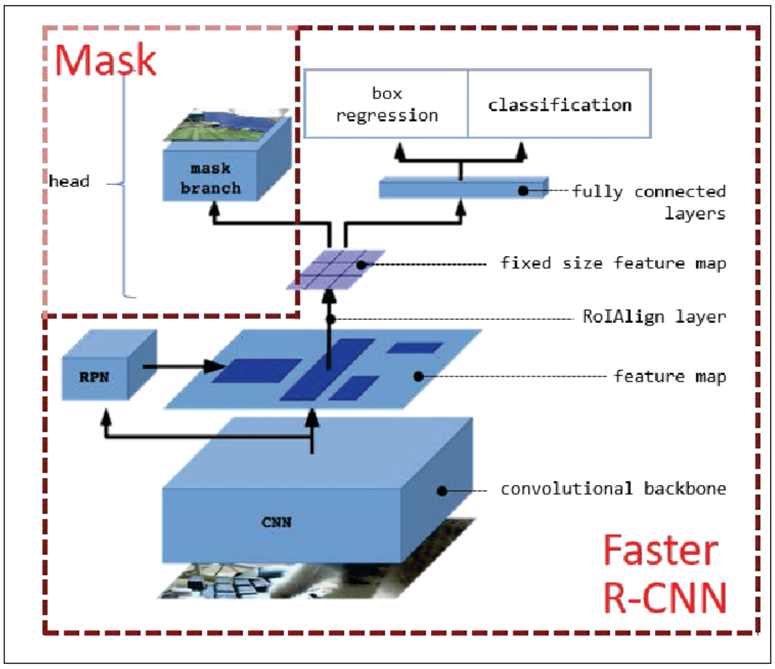


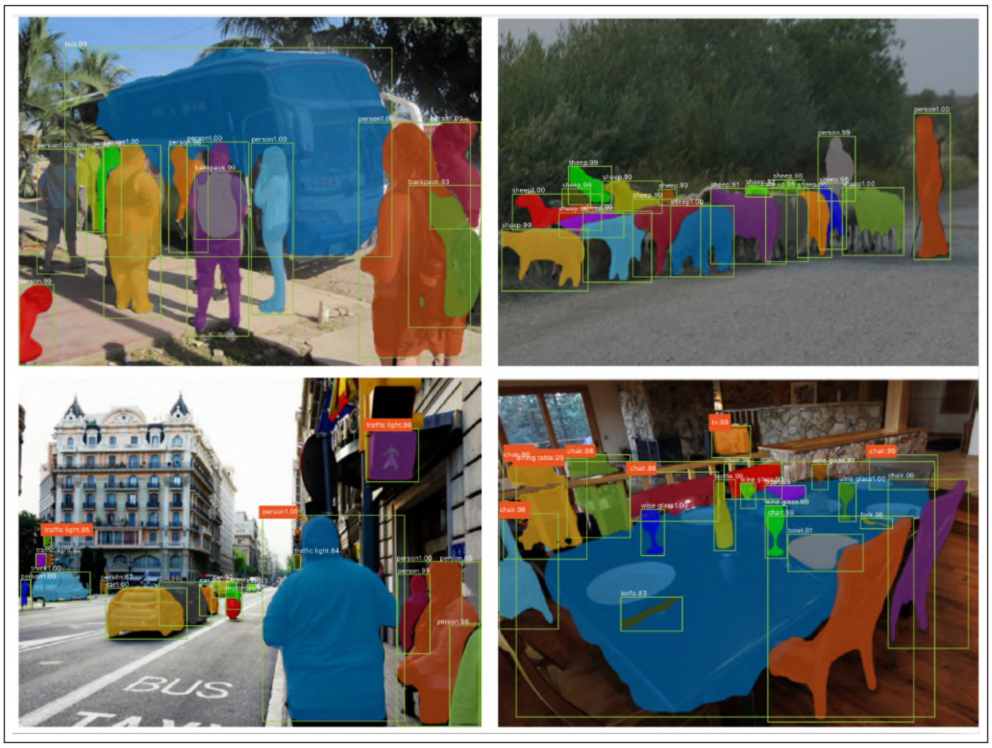
Figure 5.2: The architecture of Mask RCNN using Faster RCNN as backbone to collect the important features that will be needed by the detector to create near accurate segmentation masks.

He et. al [9] have applied Mask RCNN to COCO test set and the segmentation mask obtained are shown in Figure 5.3 (top). Similarly, when Mask RCNN is applied to City Scape dataset, the Mask RCNN module recognizes every detail accurately. This shows the capability of Mask RCNN in segmenting natural imagery. We will be using this architecture as the starting of pipeline, by training it on Smear Slide dataset to get near accurate segmentation mask. This will be the starting point of the pipeline to get the cell which will be passed to the domain adaptation pipeline.

5.2 Training Mask RCNN - Initial Configurations

We have used the Matterport's implementation of MaskRCNN [2] in Keras (Python3). Brief architecture of Mask RCNN may be seen in Figure 5.4. The architecture uses ResNet 101 backbone for getting the features, which is in turn trained on the MS COCO (Microsoft Common Objects in Context) [14] dataset. The dataset was divided into 80-10-10, i.e., 80 train, 10 validation and 10 test to perform training.

The configurations that were used in the Training of Mask RCNN pipeline were, 3 images per GPU, we were using Quadro GV100 GPU with 32 GB RAM which could easily handle that. There were 10 classes and 1 Background, so total of 11 classes per pixel were detected. Steps per epochs were set to 500, which would result in prolonged training, whereas validation steps were set to 40. A minimum confidence of 0.7 was set to detect an instance of the object present in the slide. 128 Regions of Interest were used for the training per image. 200 instances of Ground truth can be used for training, whereas the detected instances were set to 100. In the training of the Mask RCNN, the annotation files in *.xml* were parsed for each of the classes and were set to training internally. For the proper training of the Network, 200 epochs were set for training the head of the network, whereas extra 200 epochs were set for training the whole dataset. This is done to initialize the weights properly by doing part wise training. While the Network's head were training, the pretrained Convolutional layers were not updating the weights, whereas when the whole network was training, everything was being updated. A learning rate of 0.001 was used (present as default in the package), which was subclassed and overridden from the Mask RCNN package. It also used a momentum of 0.9 during training.



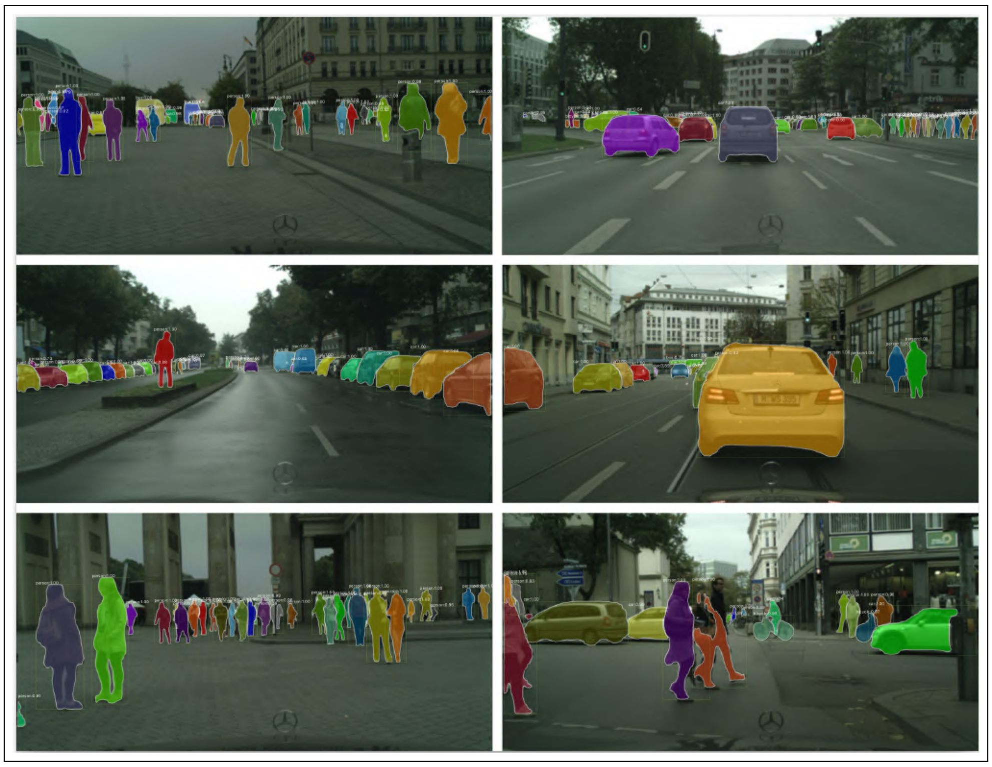


Figure 5.3: Top: Mask RCNN applied on COCO test set. Bottom: Mask RCNN applied on CityScape dataset. (Picture Courtesy: Reproduced with permission from Kaiming He.)

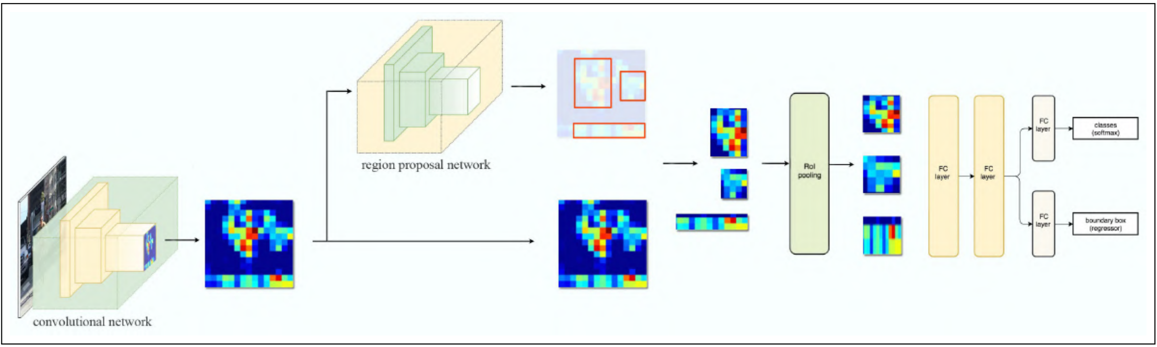


Figure 5.4: Mask RCNN architecture in brief. The architecture starts with a set of Convolutional Networks in the left, with a Region Proposal Network in the middle, where the features are made same sized with ROI pooling before passing to a set of Fully Connected Layers (network heads) which gives bounding boxes (using Regressor) and classes (using Softmax) as outputs.

The default optimizer of SGD (Stochastic Gradient Descent) [19], was used with all these settings. A separate annotation file was created to seperate these files for passing into Mask RCNN for training.

The losses can be formulated as $L_{total} = L_{cls} + L_{box} + L_{mask}$ .The graph of training loss by training the heads of the network is shown in Figure 5.5. The result of training loss for the whole network is shown in Figure 5.6. The graph might be a little misleading, but the losses while training the heads converge from a higher value to a lower value, while in the case of training the whole network, the losses are already converged, so it tries to oscillate a bit keeping everything almost same throughout the training. The networks make micro adjustments during this training.

5.3 Results of Mask RCNN on Test Dataset

After the training we have saved the model in **mask_rcnn_blood_final_model.h5** file. This will help us to run the model anywhere without training it again, we have also built a code which can point to any directory containing only test files to be masked by the model, and by doing this it will return a **.JSON** file which will contain all the detected mask (in polygon), class name, coordinates of bounding box, image name, height and width for the particular examples present in the folder.

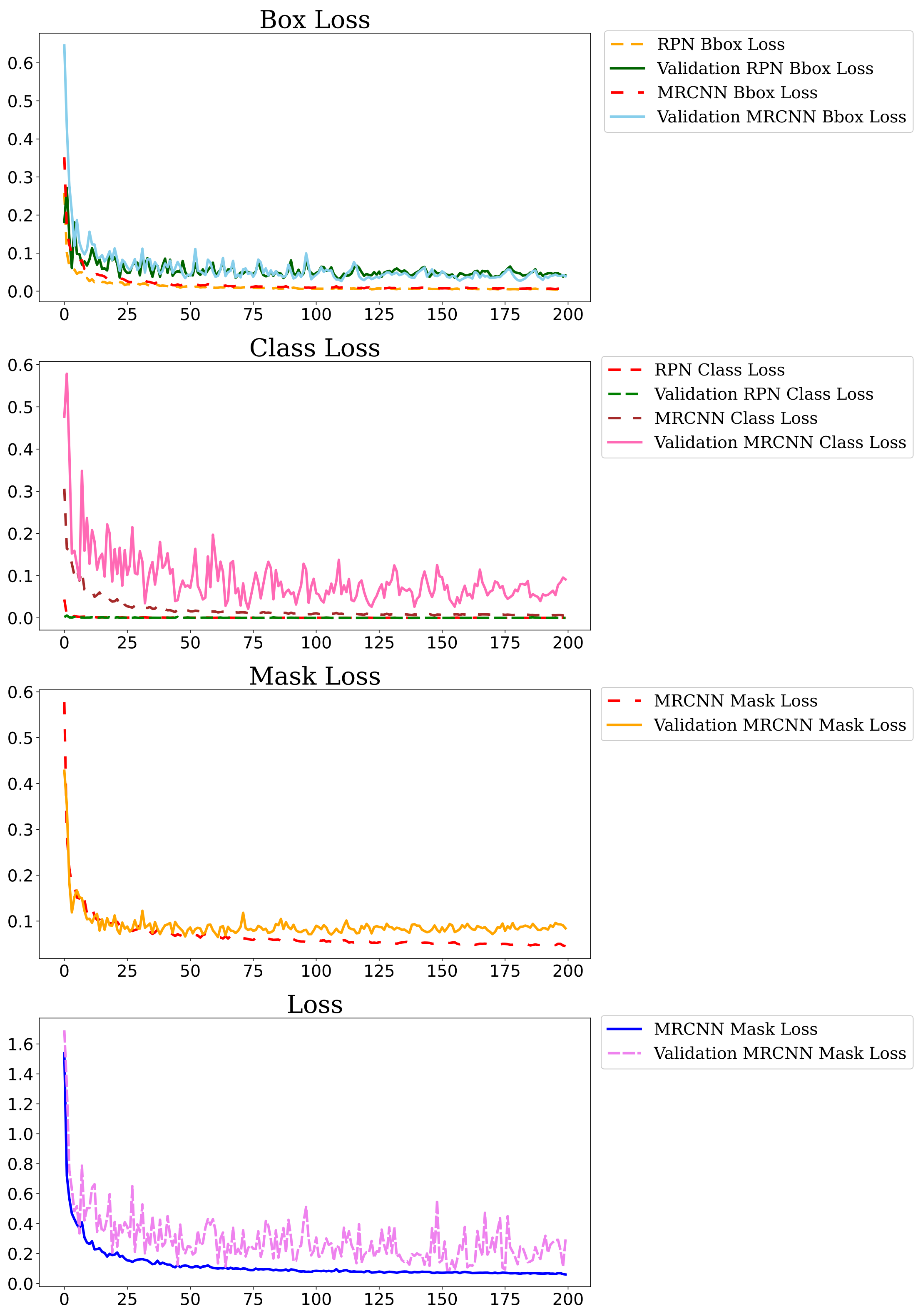


Figure 5.5: Training and Validation Losses for **Network Heads** of Mask RCNN. From top: Box Loss (L_{box}), Class Loss (L_{cls}), Mask Loss (L_{mask}), and Total Loss (L_{total}).

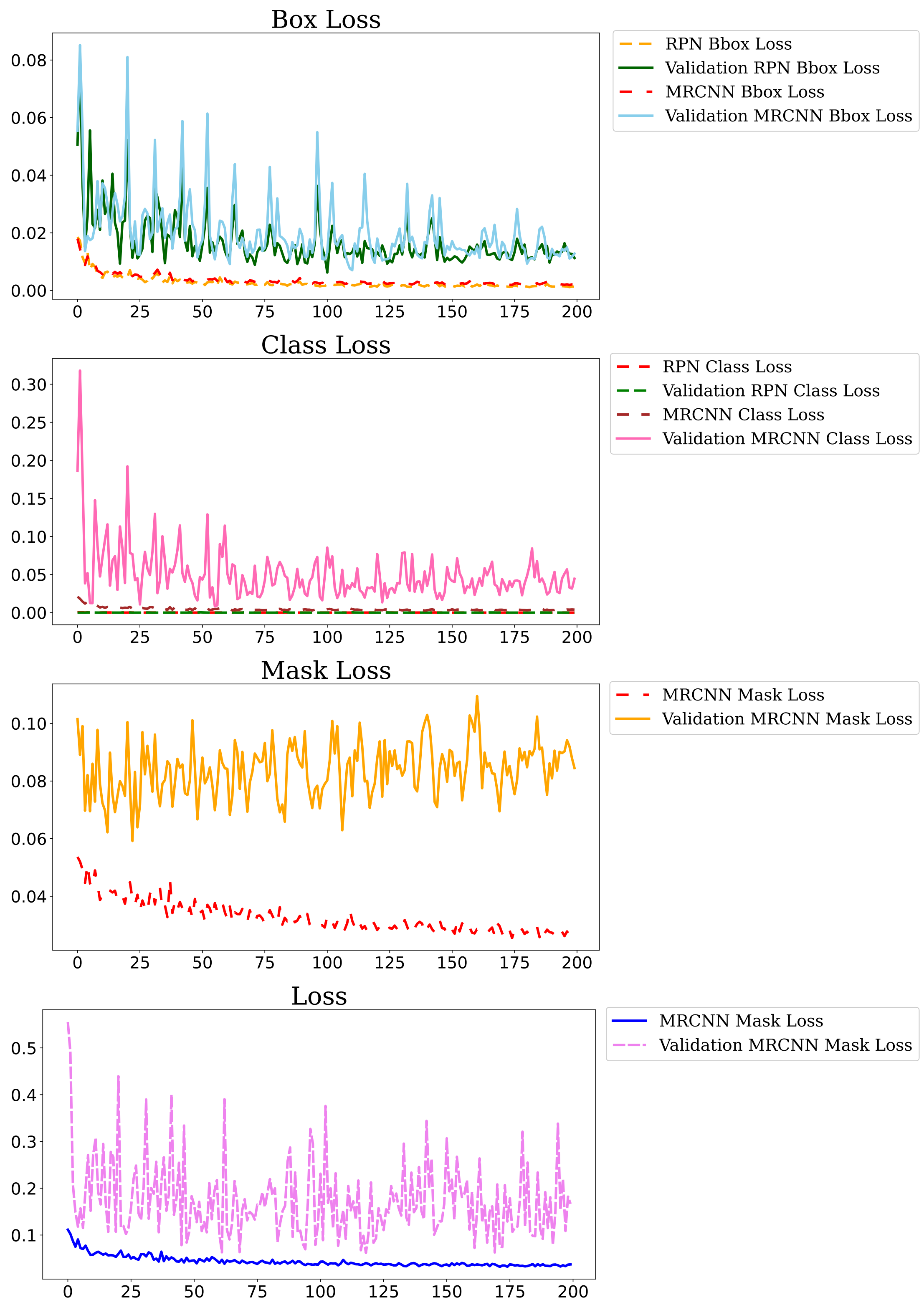


Figure 5.6: Training and Validation Losses for the **Whole Network** of Mask RCNN. From top: Box Loss (L_{box}), Class Loss (L_{cls}), Mask Loss (L_{mask}), and Total Loss (L_{total}).

A sample of JSON file generated is shown below, the JSON is contained in the ”data” key.

```
{"data":
    {"0":
            {
                "filename":"IMG_4302.jpg", "height":3024, "width":4032,
                "masks":
                [
                    {
                        "class_name":"neutrophil",
                        "score":0.9632735252,
                        "bounding_box": { "x1":22, "x2":740, "y1":436,"y2":1169},
                        "vertices":[[397.0,1142.5], ... , [396.0,1141.5],
                        [397.0,1142.5]]
                    },
                    {
                        "class_name":"neutrophil", "score":0.9132735252,
                        "bounding_box": {"x1":223, "x2":940, "y1":936, "y2":1769},
                    "vertices":[[223.6,940.5], ... ,
                    [224.3,939.4], [223.6,940.5]]
                    }
                ]
                },
      "1":
            {
                ...
            }
        ...
    }
}


----------------------------------------------------------------
A sample of JSON file generated using the Mask RCNN custom scipt
```

Some of the examples of cell images detected by Mask RCNN is shown in Figure

5.7, 5.8, 5.9 and 5.10. The top image shows the Ground Truth of the instance, visualized using our custom script, which is clearer than Mask RCNN's script, and similarly, we have parsed the .JSON file and collected the visualized images in the folder. There files are taken from the test set, which comprises about 10% of the whole dataset.

5.4 Results of Mask RCNN on Mixed Unseen (without GT) images

In general, the Peripheral Blood Smear slides will have mixed cells containing different variety of cells clumped together in one smear slide. The training of Mask RCNN was done in slides which contained only single class instances per images (mostly). In the following section we will see how the model behaves when there are a lot of cells present. This is achieved via keeping the new slides in the folder and pointing the script for generating the JSON file to that folder. The JSON file would be present in the folder containing the file. After that, we have collected the JSON file and checked the mask generated by our own script. The results are shown in Figure 5.11, 5.12, 5.13 and 5.14 respectively. From the images we can see that Mask RCNN does a relatively better job in training about 400 or so images and testing on unseen and different type of images.

From the results obtained, we can certainly say that the more the amount of data supplied, the better the model will be able to segment and detect the images present. The Mask Generated is quite good, given there are some confusing fluid present in the images.

5.5 Future Scope - Automated Masking of Smear Slides Dataset

We have certainly seen that Mask RCNN does a great job in masking out the cells from unseen data. This motivates us to automate the most labour intensive task

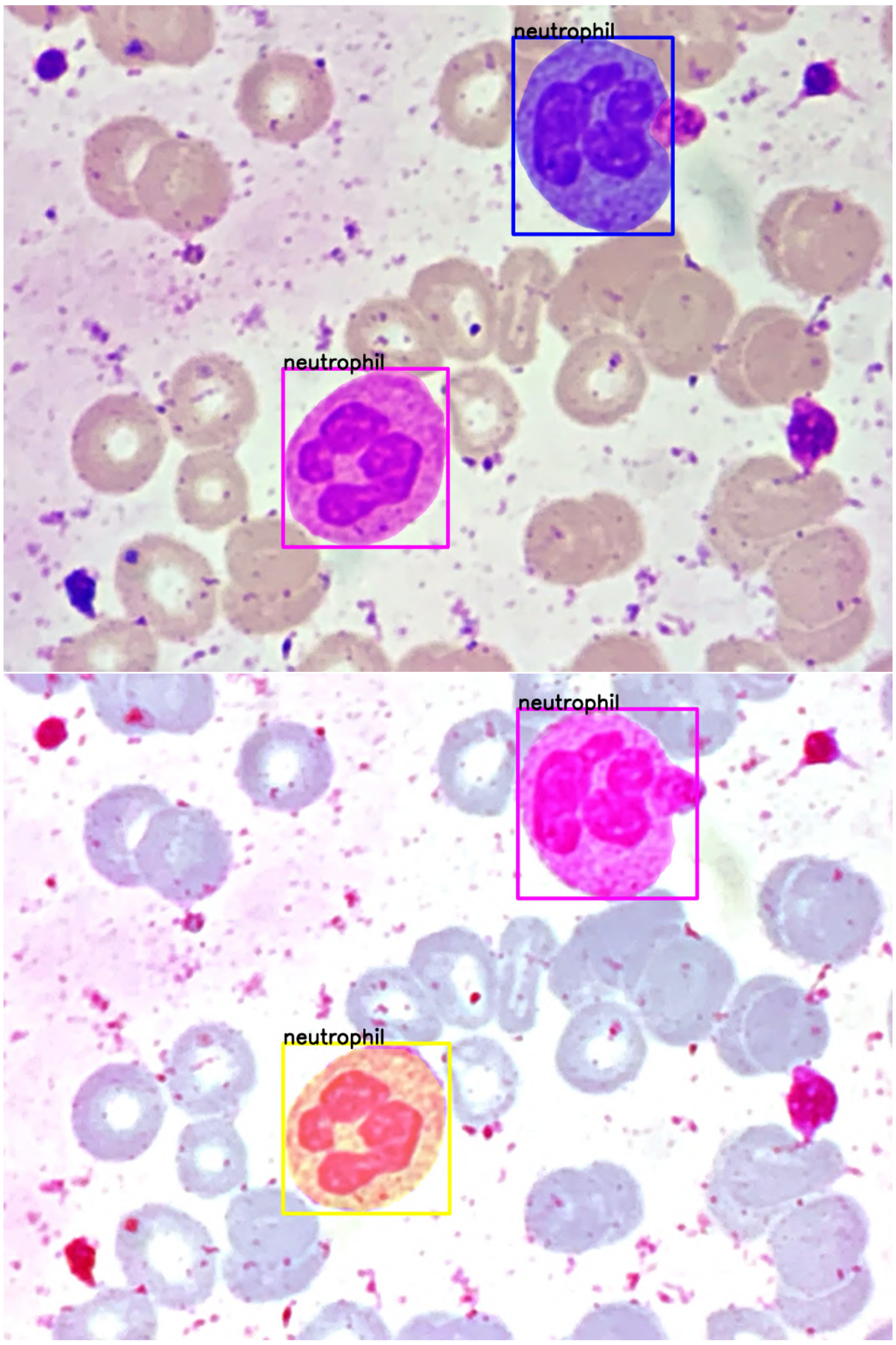


Figure 5.7: Top: The Ground truth containing an instance of Neutrophil, Bottom: Detected and masked Neutrophil via Mask RCNN.

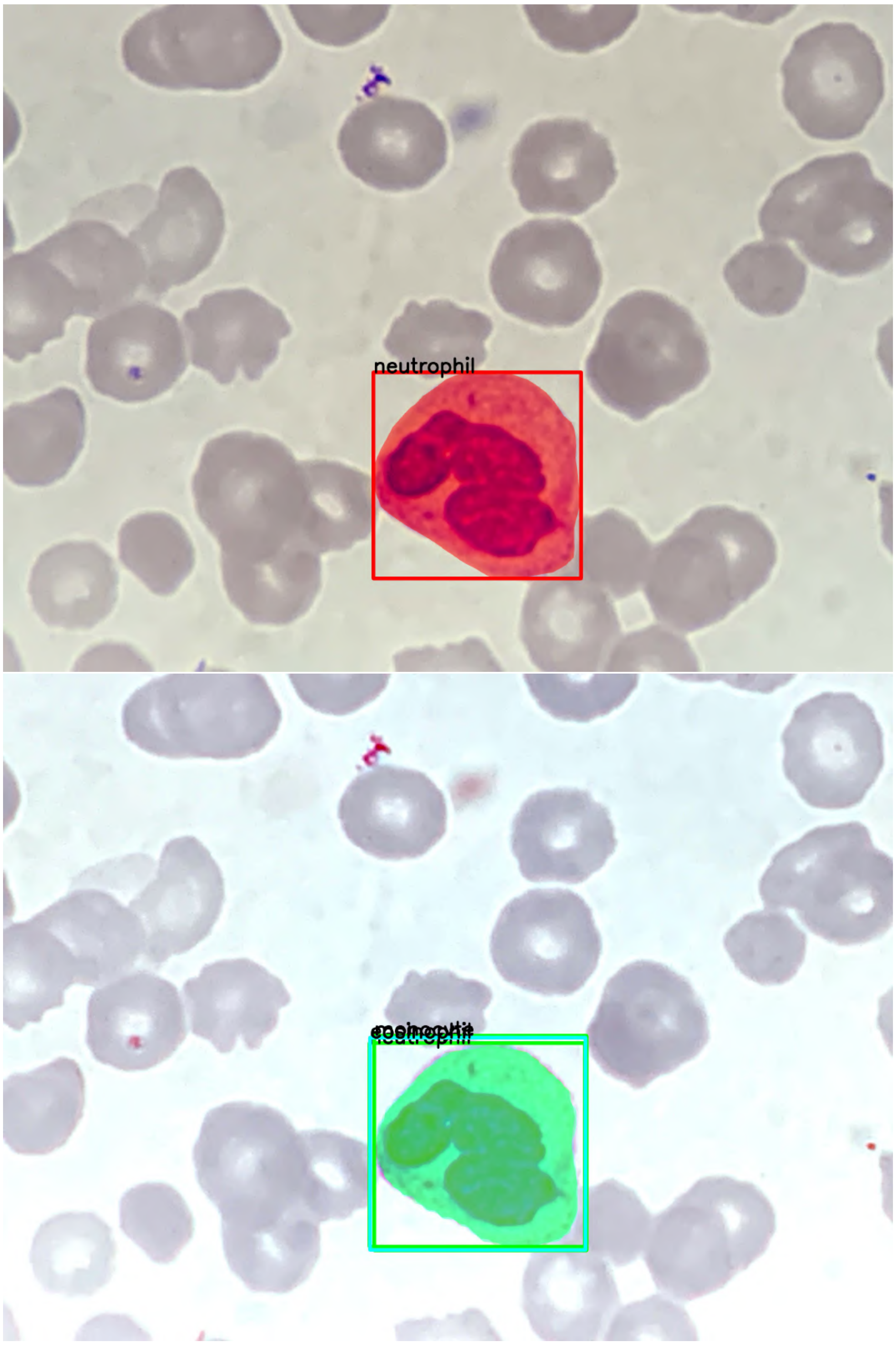


Figure 5.8: Top: The Ground truth containing an instance of Neutrophil, Bottom: Detected and masked Neutrophil via Mask RCNN.

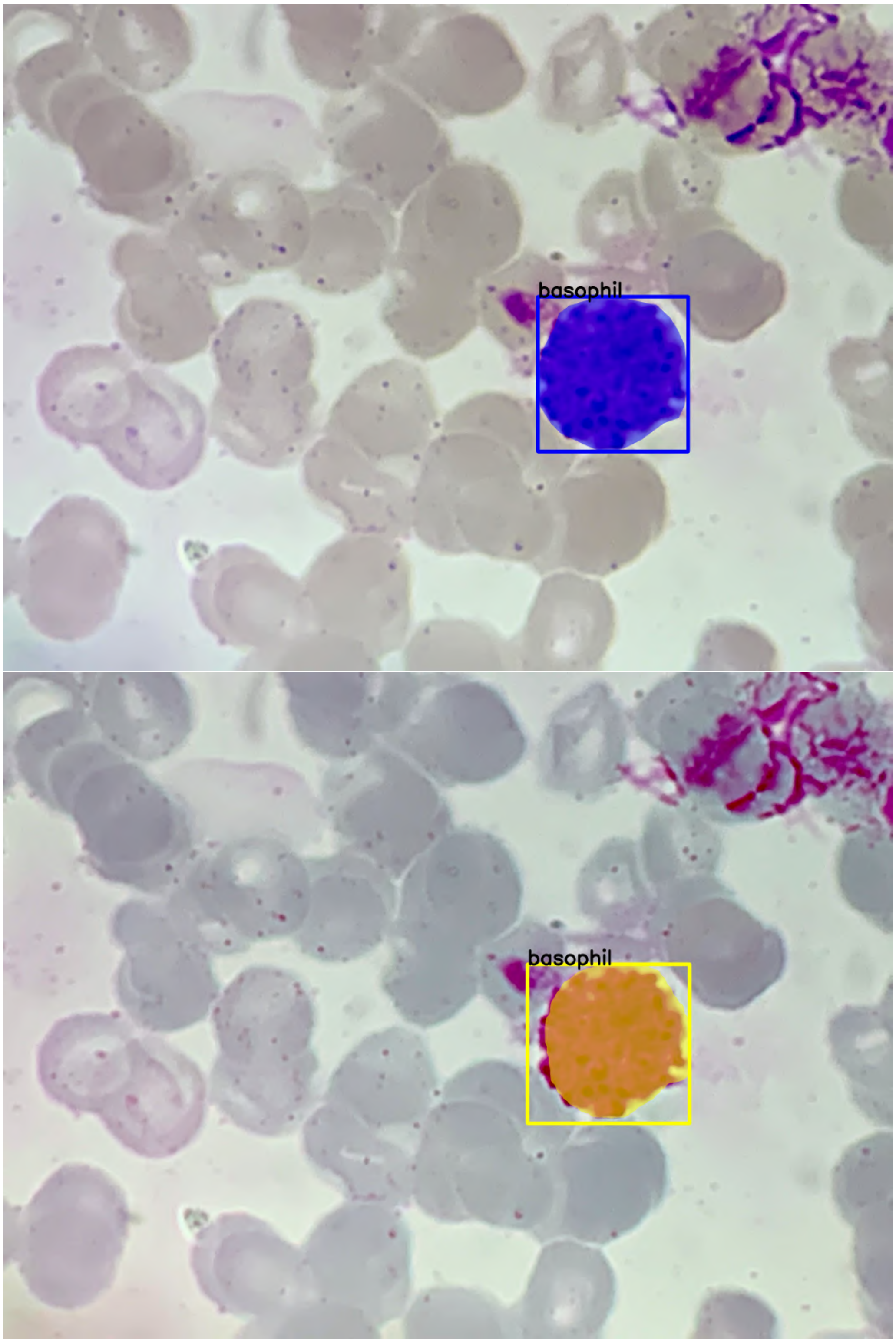


Figure 5.9: Top: The Ground truth containing an instance of Basophil, Bottom: Detected and masked Basophil via Mask RCNN.

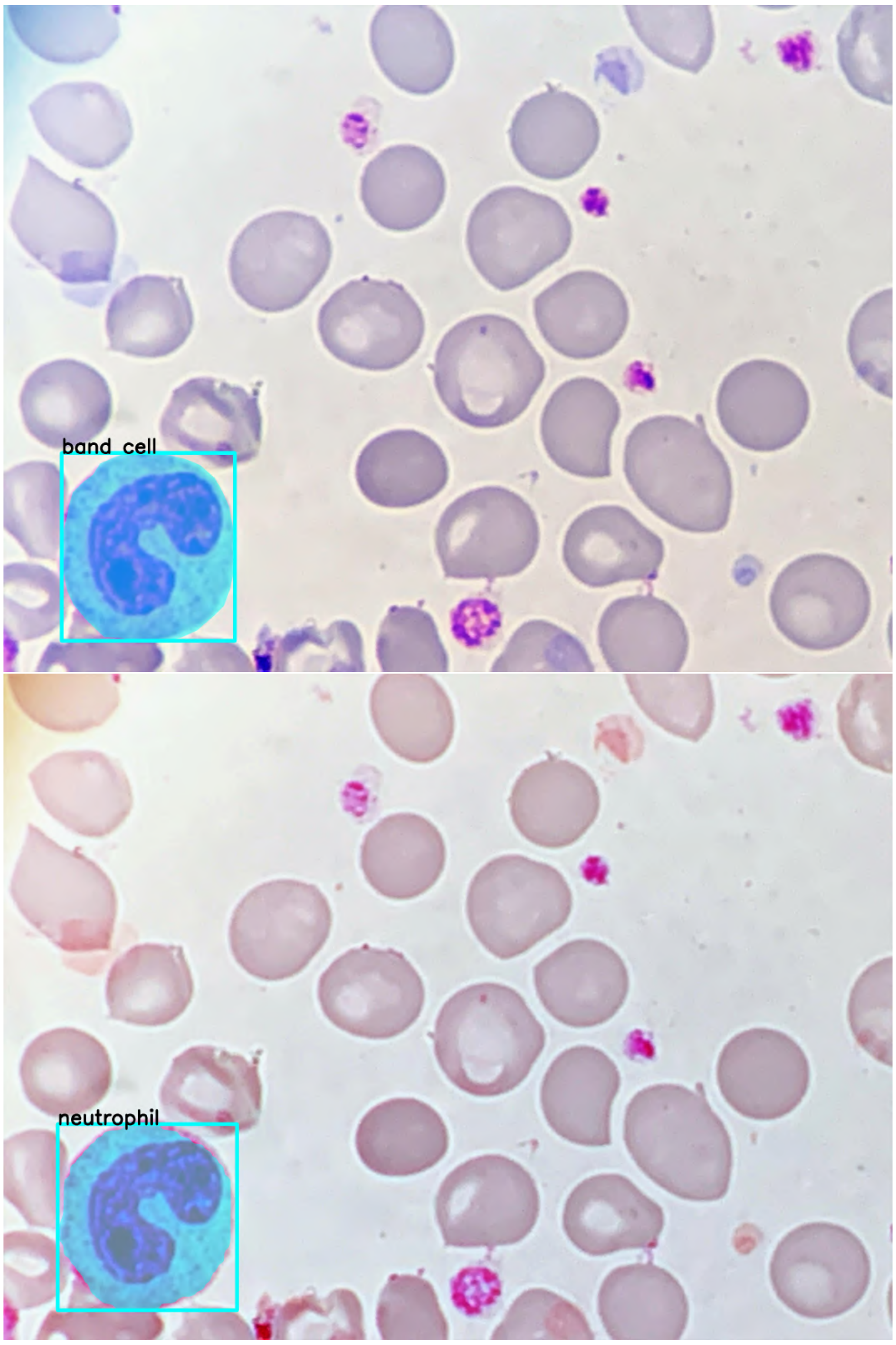


Figure 5.10: Top: The Ground truth containing an instance of Band Cell, Bottom: Detected and masked Neutrophil Cell via Mask RCNN (a wrongly detected sample).

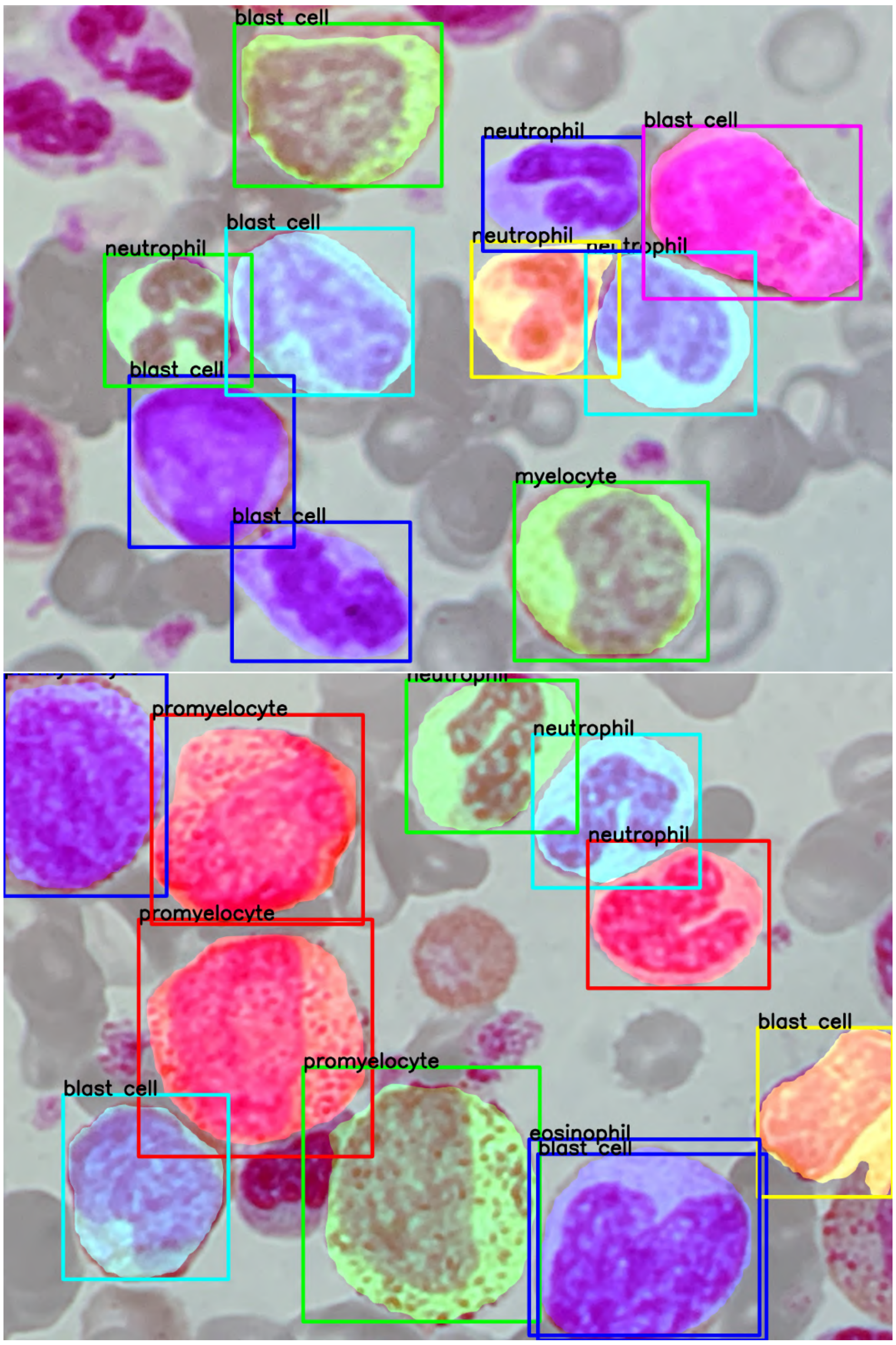


Figure 5.11: Sample of Unseen mixed slide taken by testing Mask RCNN.

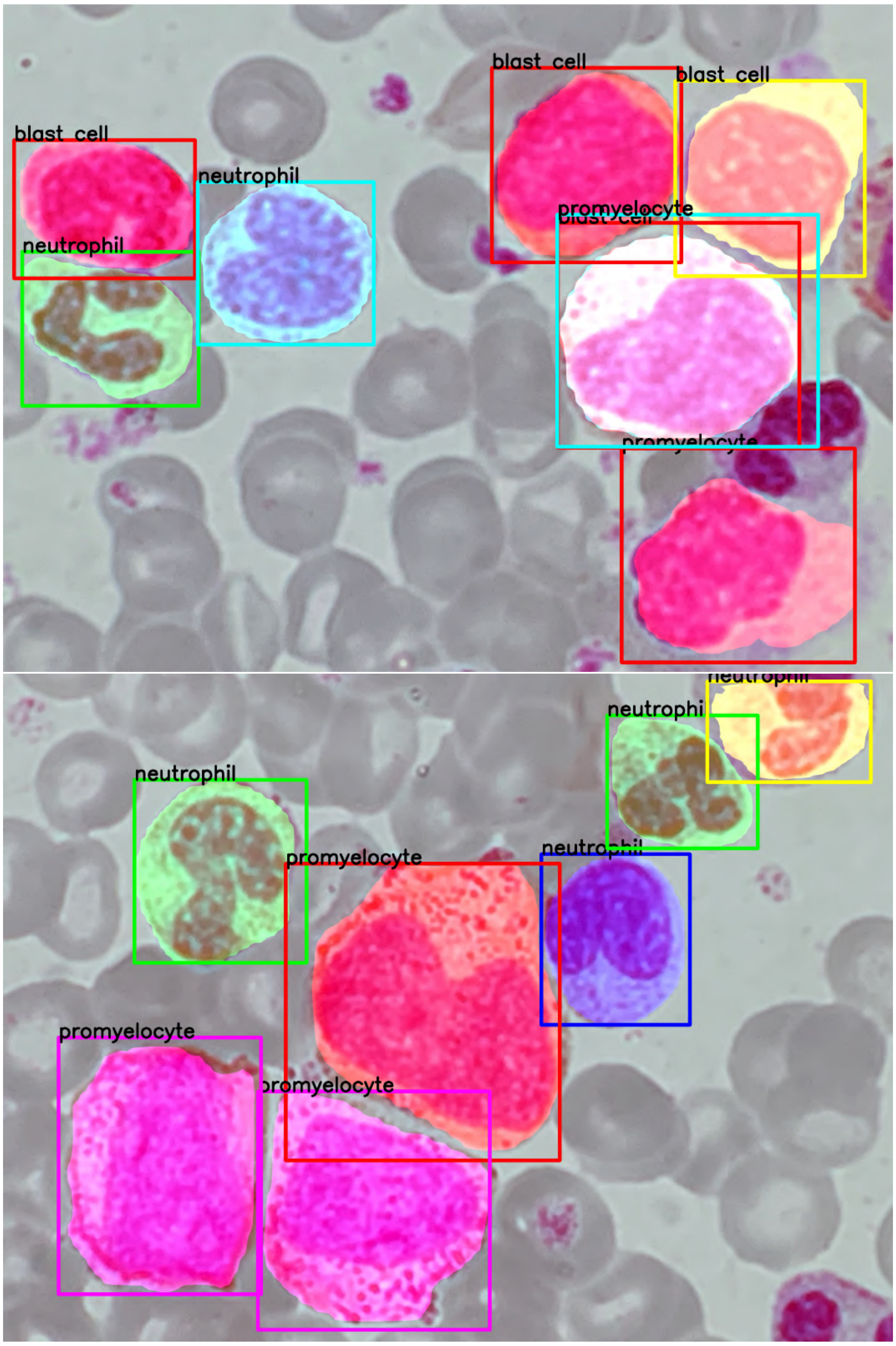


Figure 5.12: Sample of Unseen mixed slide taken by testing Mask RCNN.

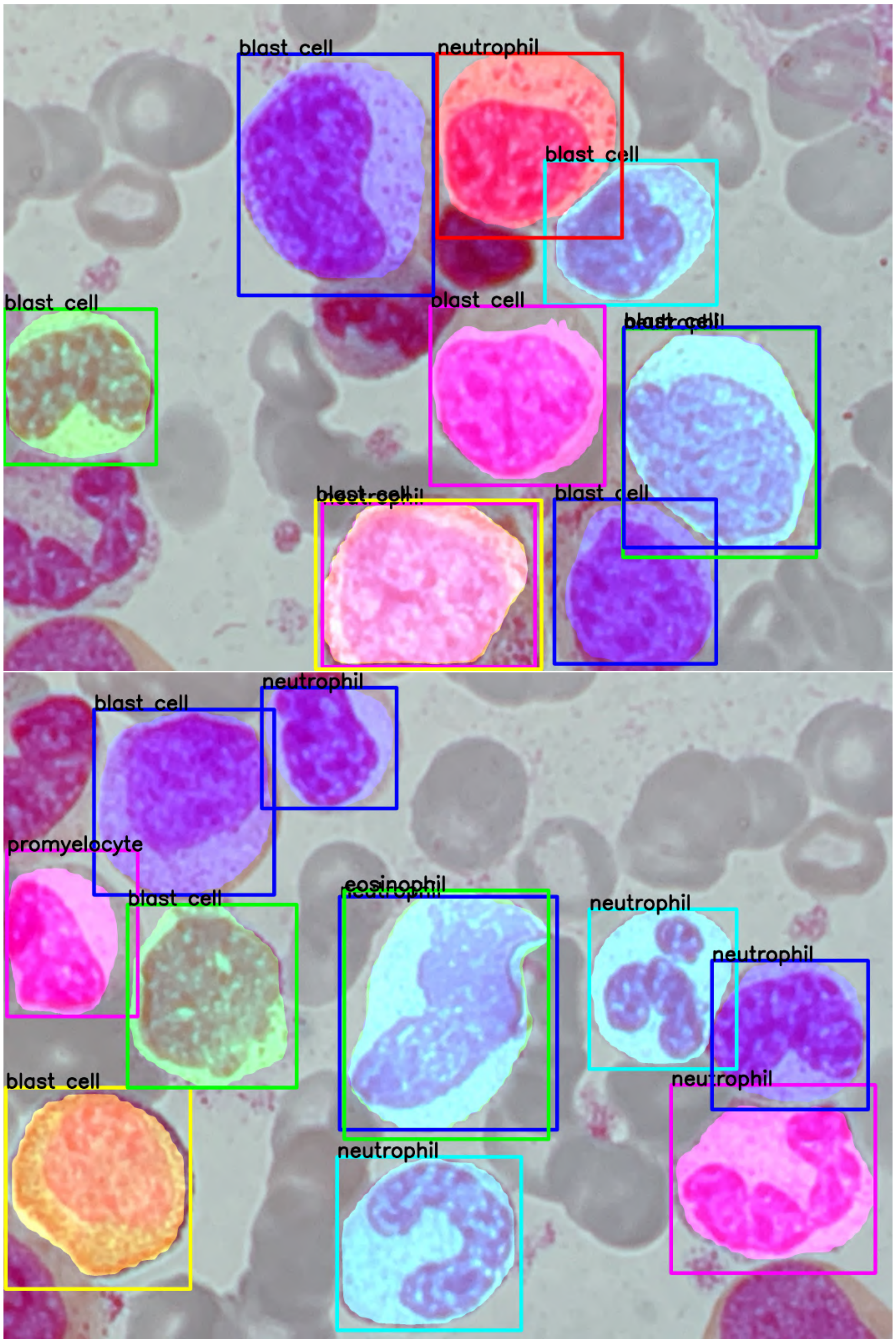


Figure 5.13: Sample of Unseen mixed slide taken by testing Mask RCNN.

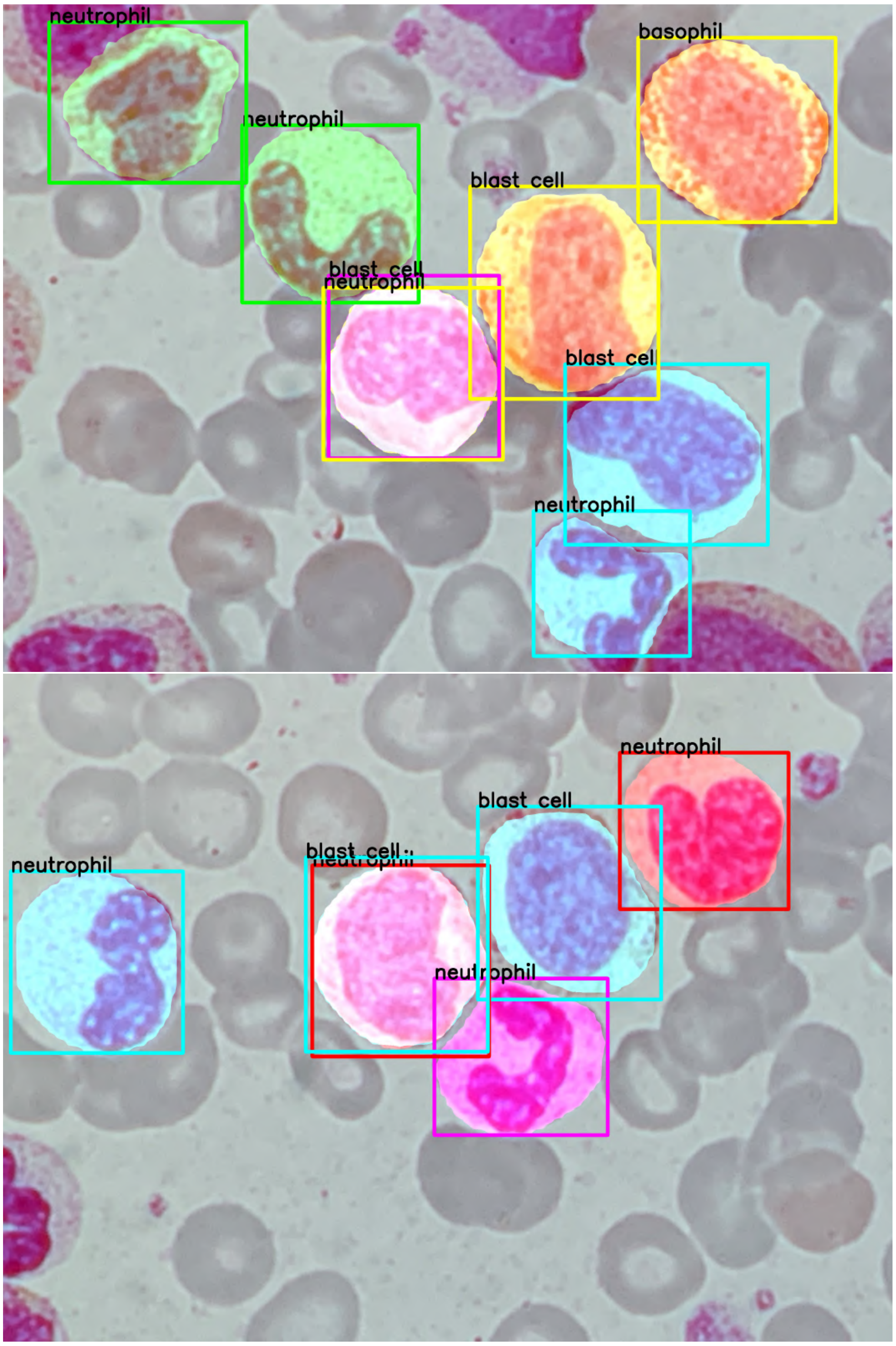


Figure 5.14: Sample of Unseen mixed slide taken by testing Mask RCNN.

of generating polygons for the cells. The new tool will be used to generate polygon and dump it into the existing annotation *.xml* file. We will be able to upload it to CVAT for fine tuning the already automated annotated file and it will significantly reduce the time and cost for annotation.

Chapter 6

Domain Adaptation

6.1 Domain Adaptation as a part of Transfer Learning

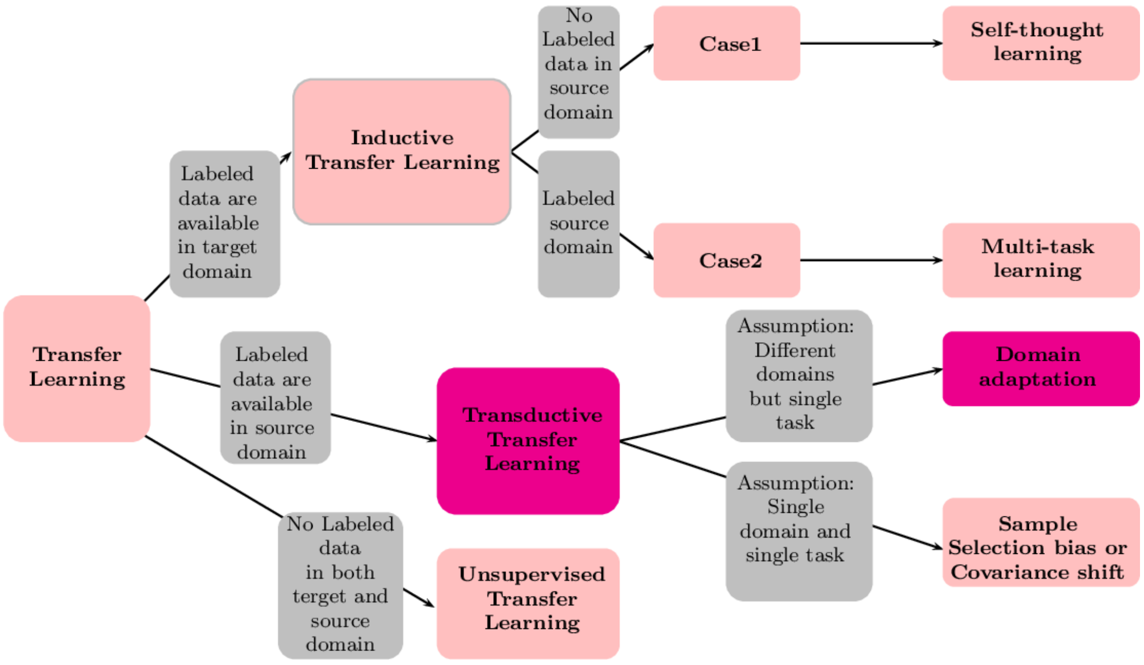

Transfer Learning
Labeled data are available in target domain
Inductive Transfer Learning
No Labeled data in source domain
Labeled source domain
Case1
Case2
Self-thought learning
Multi-task learning
Labeled data are available in source domain
Transductive Transfer Learning
Assumption: Different domains but single task
Assumption: Single domain and single task
Domain adaptation
Sample Selection bias or Covariance shift
No Labeled data in both terget and source domain
Unsupervised Transfer Learning

Figure 6.1: Flowchart showing the hierarchies of Transfer Learning [6].

Transfer learning is the future of AI.[1] It is becoming increasing popular in retraining deep networks by using predefined weights which was trained in bigger datasets

[1]https://ruder.io/transfer-learning/

to capture state of the art results. That knowledge is then transferred to some other problems where the data is less. A schematic representation of Transfer Learning is shown in Figure 6.1. Domain adaptation is a small part of this tree.

There has been a long challenge of training images in an unsupervised way. There are a lot of unlabeled data in the internet but very few labelled data which needs annotators (human resources) to label them. Unsupervised domain adaptation by backpropagation [7] is a way to utilize some of the labelled data to learn features from some of the unlabeled data in such a way that we get domain invariance features from both the dataset. We will use this style of architecture to train our own model which learns to classify images from both the domain, i.e., here PBC and Smear Slides.

New methods are also implemented, for example leveraging the power of adversarial networks [23] to outperforms classification models on standard datasets. These type of networks captures the distribution of classes in both the domains, and is best at its tasks. Domain adaptation can also be applied to segmentation problem [17], but one dataset needs to have proper segmentation mask and in large numbers. In our case, we have masks but the number of data slides is low, and for the PBC dataset, the mask may contain variety of noises, which will be ineffective in training good networks for medical dataset where critical information is necessary. There have been works for performing detection [11] by training models via domain adaptation. There are also few works in medical sectors which uses domain adaptation, a good survey can be found in [8].

6.2 Domain Adaptation Pipeline

Since, we have the Mask RCNN and the PBC Extraction algorithm working pretty well, we will now need to run the domain adaptation pipeline. The pipeline is shown in Figure 6.2. There are two stages of the extraction process, one to crop out cells from the PBC dataset to create the PBC Cropped dataset, another to crop out the masked images from the Smear Slides, this will create the Smear Slides Cropped dataset. The pipeline shown in bottom will also be used to crop out the cells present in individual smear slides and then perform classification on them.

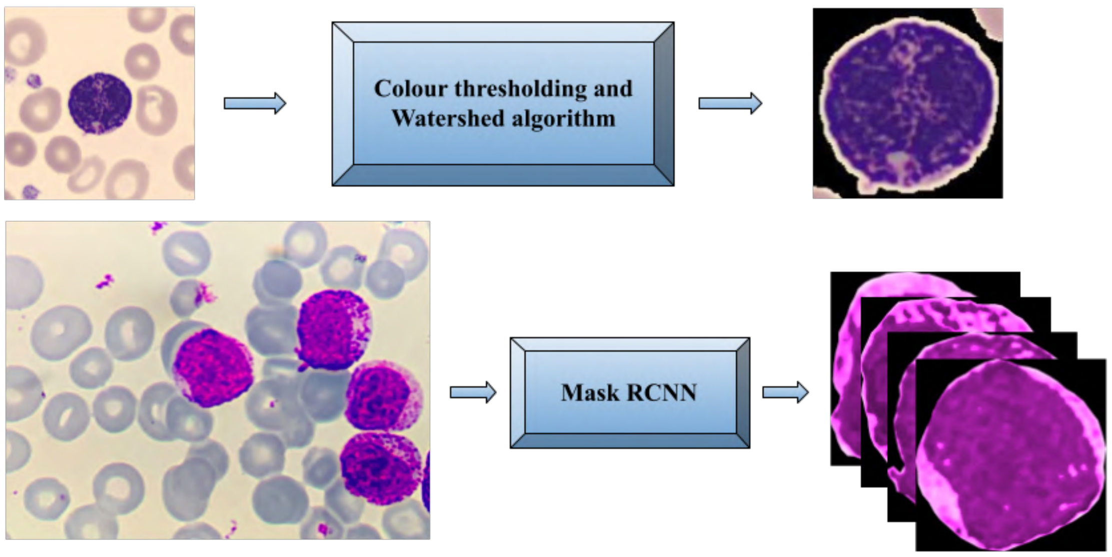


Figure 6.2: Domain Adaptation pipeline in general. We crop out the PBC Slides by Watershed algorithm, and for the Smear Slides, we use Mask RCNN to get out the mask. These two will comprise the dataset from two different domains.

6.3 Experiments on the common classes

Before performing Domain Adaptation, we need to show how the model performs for 8 classes, which have not been tested before. For this purpose, we have shown the result for the 8 classes which are common. For comparing the results of Domain Adaptation, we need to show how the Xception model performs on Smear Slides Cropped 8 classes by training on PBC Cropped DA 8 classes dataset. Similarly, we have to show the reverse, i.e., how the model performs on PBC Cropped DA 8 classes by training on Smear Slides Cropped 8 classes. We also have to show how the model performs when the dataset is mixed.

6.3.1 Xception Trained on PBC cropped DA 8 classes and testing on Smear Slides Cropped 8 Clases DA

We train and fine-tune Xception model on PBC cropped dataset with 8 classes, test on the same dataset and also test it on Smear Slides Cropped 8 Classes. The confusion matrix obtained is shown in Figure 6.3. The Precision, Recall and F1-score obtained by testing on PBC Cropped dataset are **94.78 %**, **94.77 %** and **94.79**

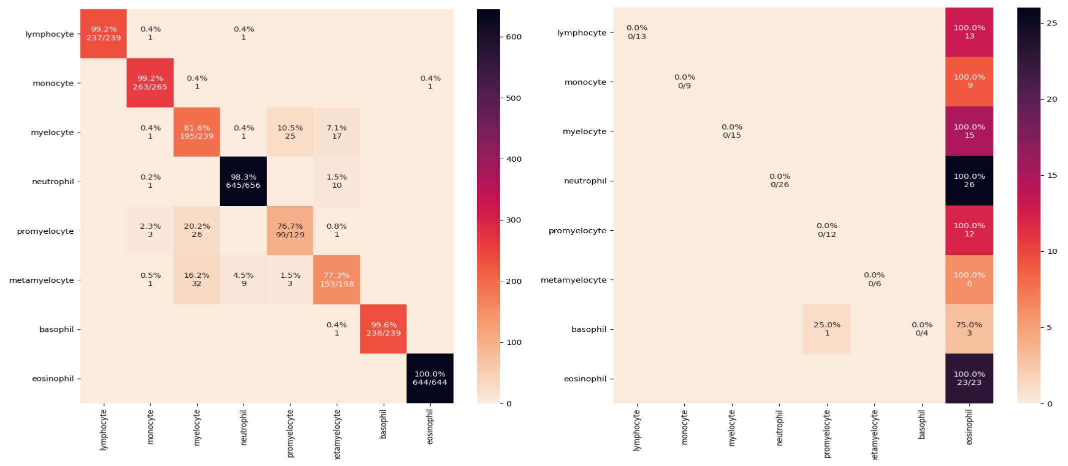


Figure 6.3: Confusion matrix obtained by training the fine-tuned model on PBC 8 DA dataset, evaluating on the test dataset and also evaluating on the Smear Slides 8 DA dataset without data augmentation.

%, which is a bit less, due to the crowding of confusing classes which were not concentrated on 8 super classes, hence decreasing the evaluation metrics. Similarly, the Precision, Recall and F1-score obtained by testing on Smear Slides Cropped dataset are **4.47 %**, **21.29 %** and **7.45 %** which is very less, since there is a domain shift in the dataset, even though it might have same classes.

6.3.2 Xception Trained on Smear Slides Cropped 8 Clases DA and testing on PBC cropped DA 8 classes

Here, we train and fine-tune Xception model on Smear Slides Cropped 8 Classes, test on the same dataset and also test it on PBC cropped dataset with 8 classes. The confusion matrix obtained is shown in Figure 6.4. The Precision, Recall and F1-score obtained by testing on Smear Slides Cropped dataset are **82.64 %**, **82.25 %**, and **81.74 %**, which is a better on the other hand, due to less crowding of confusing classes which were not concentrated on 10 classes, hence increasing the evaluation metrics a bit. Similarly, the Precision, Recall and F1-score obtained by testing on PBC cropped dataset with 8 classes are **53.34 %**, **24.36 %** and **28.29 %** which is better than the previous result when we were evaluating Smear Slides dataset by training on PBC 8 DA Cropped dataset. This is probably due to the fact that the Smear Slides dataset have more context rich features which can help

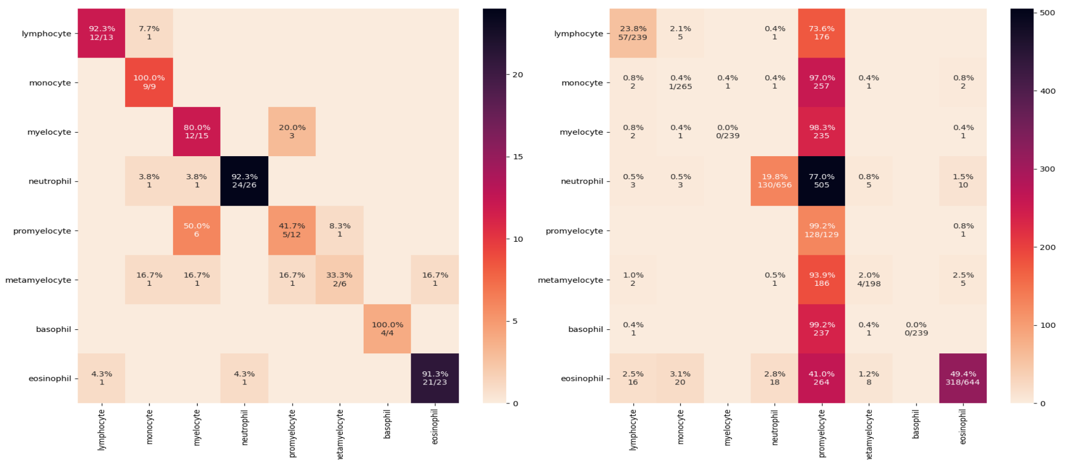


Figure 6.4: Confusion matrix obtained by training the fine-tuned model on Smear Slides 8 dataset, evaluating on the test dataset and also evaluating on the PBC 8 DA dataset without data augmentation.

to generalize the other dataset better.

6.3.3 Combining both the datasets and their common classes using Xception Model

By Combining both the datasets, i.e., common classes and fine tuning the Xception model, we get a Precision of **92.41 %**, Recall of **92.16 %** and F1-score of **92.19 %**. The confusion matrix obtained by the experiment is shown in Figure 6.5. It looks pretty good, since the model learns to generalize the datasets by treating it as one dataset. This is different from performing domain adaptation since we are just merging the datasets without learning the proper distribution of the data, in this case, it might not generalize well for those samples which were unseen.

6.4 Domain Adaptation - Model and Formulation

Let us consider data samples $x \in X$. Here, X is some input space, and $y \in Y$ are certain labels output, from label space. There are two distribution, i.e., Source, $S(x, y)$ and Target, $T(x, y)$, both are complex and unknown. Here, S is shifted from target distribution T by some domain shift. The goal is to able to predict y, given

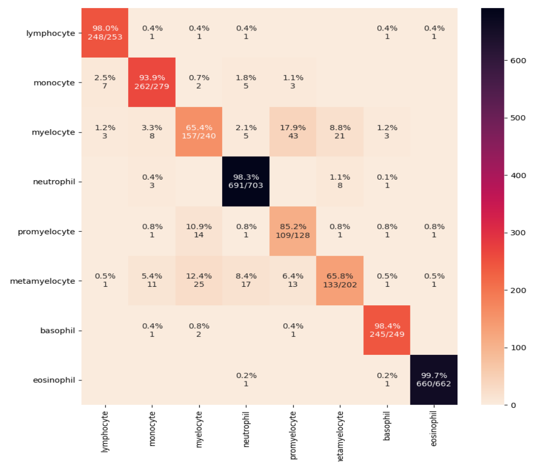


Figure 6.5: Confusion matrix obtained by training the fine-tuned model on mixing Smear Slides 8 dataset and PBC 8 DA Cropped dataset, and evaluating on the test dataset without data augmentation.

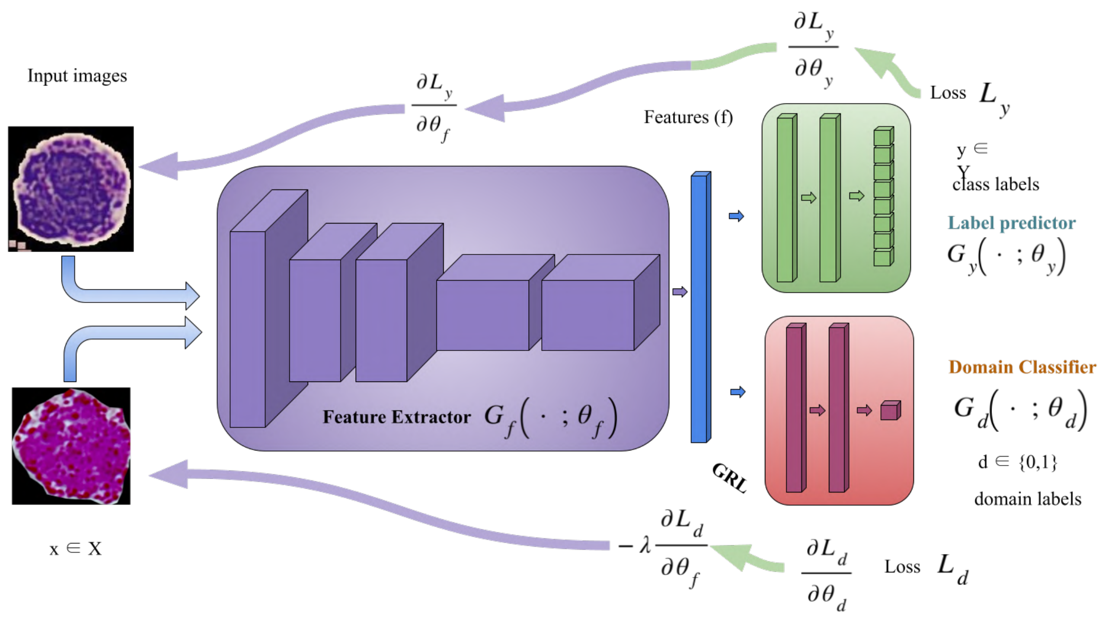
input x for the target distribution. During training we have access to large number of training samples $\{x_1, x_2, ...x_N\}$ from both the Source and Target distribution.

Here, we have an extra parameter, i.e., domain d

$$d_i = \begin{cases} 0, & \text{if } x_i \in \text{Source domain, } y_i \in \text{Y are known} \\ 1, & \text{if } x_i \in \text{Target domain, } y_i \in \text{Y are necessarily not known} \end{cases}$$

The task is to predict target domain's input samples in test time. A feed forward neural network architecture may be used to predict $y \in Y$, and its domain label $d_i \in \{0, 1\}$. A $\mathbb{D}$ dimensional feature vector ($f \in R^{\mathbb{D}}$) is acquired by passing the input x through G_f, which is also a Convolutional Deep Neural architecture for feature extraction.

During the training, we aim to minimize the label prediction loss on the annotated part, i.e., source part, of the training set and hence both the label predictor and feature extractor are thus optimized, in order to minimize the empirical loss of source domain samples. This ensures the discriminativeness of the features f, and the overall good prediction performance of the combination of feature extractor and label predictor on the source domain.



Input images
$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
Loss L_y
Features (f)
y ∈ Y
class labels
Label predictor
$G_y(\cdot\,;\theta_y)$
Feature Extractor $G_f(\cdot\,;\theta_f)$
Domain Classifier
$G_d(\cdot\,;\theta_d)$
d ∈ {0,1}
domain labels
GRL
x ∈ X
$-\lambda\frac{\partial L_d}{\partial \theta_f}$
$\frac{\partial L_d}{\partial \theta_d}$
Loss L_d

Figure 6.6: The Domain Adaptation model by passing input images from two different domains in random order. The feature extractor is taken from the best performing model for classification, and for both the cases, it is the Xception model. The features (f) are passed to the label predictor (which predicts the classes of the data) and the domain classifier (which predicts the domain of the data). The domain classifier tends to get more confused over time, while the label predictor gets better over time, which helps to learn domain invariant features.

We want to make the two features f domain invariant at the same time, i.e., make the distributions, $S(f) = \{G_f(x;\theta_f)|x \in S(x)\}$ and $T(f) = \{G_f(x;\theta_f)|x \in T(x)\}$ to be as close to each other. This will make the label predictor accuracy on the target domain to be same as source domain according to the co-variate shift assumption. Measuring the dissimilarity of distributions are non-trivial, given f is high-dimensional and the distribution themselves are constantly changing as learning progresses.

The loss of the Domain Classifier is maximized at the training time in order to obtain domain invariant features θ_f. We minimize the loss for label predictor and maximize the loss for the domain classifier.

$$\begin{aligned}
E(\theta_f, \theta_y, \theta_d) = \sum_{\substack{i=1..N\\d_i=0}} L_y\left(G_y(G_f(x_i;\theta_f);\theta_y), y_i\right) - \\
\lambda \sum_{i=1..N} L_d\left(G_d(G_f(x_i;\theta_f);\theta_d), y_i\right) = \\
= \sum_{\substack{i=1..N\\d_i=0}} L_y^i(\theta_f, \theta_y) - \lambda \sum_{i=1..N} L_d^i(\theta_f, \theta_d)
\end{aligned} \tag{6.1}$$

Here, since the label predictor $L_y(.,.)$ is a classifier, hence the loss is multinomial. Similarly, $L_d(.,.)$, the domain classifier loss is logistic. L_y^i and L_d^i are the corresponding loss functions that are evaluated at the i^{th} training example. In, general we are seeking the parameters $\hat{\theta}_f$, $\hat{\theta}_y$, $\hat{\theta}_d$ that deliver a saddle point of the function :

$$(\hat{\theta}_f, \hat{\theta}_y) = \arg\min_{\theta_f,\theta_y} E(\theta_f, \theta_y, \hat{\theta}_d) \tag{6.2}$$

$$\hat{\theta}_d = \arg\max_{\theta_d} E(\hat{\theta}_f, \hat{\theta}_y, \theta_d)\,. \tag{6.3}$$

At the saddle point, θ_d of the domain classifier minimize the domain classification loss, while the parameters θ_y minimize the label predictor loss. The features learnt by the label predictor are discriminative while maximizing the domain classification loss learning domain invariant features. The overall architecture of this process can be seen in Figure 6.6.

Standard optimizer such as Stochastic Gradient Descent can be used as optimizer. The stationary saddle point is a part of the following stochastic updates, where μ may be considered as the learning rate, which may vary over time:

$$\theta_f \longleftarrow \theta_f - \mu\left(\frac{\partial L_y^i}{\partial \theta_f} - \lambda \frac{\partial L_d^i}{\partial \theta_f}\right) \tag{6.4}$$

$$\theta_y \longleftarrow \theta_y - \mu \frac{\partial L_y^i}{\partial \theta_y} \tag{6.5}$$

$$\theta_d \longleftarrow \theta_d - \mu \frac{\partial L_d^i}{\partial \theta_d} \tag{6.6}$$

The reduction in 6.4 can be achieved by introducing a special Gradient Reversal Layer (GRL) as shown in Figure 6.6. Here, λ is a meta parameter which is not updated by back-propagation. During forward propagation Gradient Reversal Layer (GRL) acts as an identity transform, during back propagation, GRL takes the gradient from the subsequent level, multiplies it by -λ and passes it to the preceding layer. GRL is inserted between the feature extractor and the domain classifier, as shown in Figure 6.6. Hence the total loss can be formulated as 6.7

$$\tilde{E}(\theta_f, \theta_y, \theta_d) = \sum_{\substack{i=1..N \\ d_i=0}} L_y \Big(G_y(G_f(x_i; \theta_f); \theta_y), y_i \Big) + \\ \sum_{i=1..N} L_d \Big(G_d(R_\lambda(G_f(x_i; \theta_f)); \theta_d), y_i \Big) \tag{6.7}$$

After learning the label predictor, $y(x) = G_y(G_f(x; \theta_f); \theta_y)$ can be used to predict inputs from both the domains.

6.5 Domain Adaptation using Xception Model

Since we have about 13050 images in total from PBC Cropped 8 Classes DA dataset, and about 544 images from the 8 classes, so we need to do data augmentation for performing Domain Adaptation for both the classes.

6.5.1 Augmenting the Smear Slides Cropped 8 Classes Data

Since medical images are prospected under heavy constraints [2], so there is very little noise and variances in the data. There are various types of augmentations that could be applied to images, firstly, we could apply some background to the images, since the PBC Cropped dataset have some amount of background in them. The background could also be of stain, since some slide images are cluttered in stain, which could be mistaken as a part of the cell. The textures that were similar were taken as background for heavy data augmentation as shown in Figure 6.8. After applying background, we could zoom out the image a bit, from 0.5 to 1, which is got by a random vector. The images could also flip, i.e., horizontally, vertically, and

[2]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7104701/

both with a probability $\frac{1}{3}$. We have also applied color transformation i.e., cooling and heating effect on the image which suits the color of images present in the slide. The cooler represents a more purple stain while the hotter version represents a more yellowish stain. The color transformation is applied with a probability of $\frac{1}{6}$.

Applying repetitive Gaussian blur, we can fade the image, which can simulate out of focus objects, which is not so common in medical images, but we still want them to be robust to those perturbations. We can also rotate an image to a certain angle with a probability of $\frac{1}{4}$ in the range [0,90], [90,180], [100,270] and [270,360]. We did also change the contrast and brightness. All the above mentioned augmentations are shown in Figure 6.7, and they are applied to a single image in the illustration.

Augmentation is necessary in performing Domain Adaptation, since, we need to have same number of data samples from each of the classes while training. A sample of image is applied to a series of random augmentations with a vector of size 11 for example (1,0,1,1,1,1,0,0,1,0,1), where each of the values decides whether an augmentation will happen (1) or not (0). A series of data augmentations can be shown in Figure 6.9.

6.6 Domain Adaptation Source : Smear and Target : PBC Cropped

This is performing unsupervised domain adaptation on smear slides as source and PBC slides as target. This generally means that with the help of Smear's label data we will be learning the distribution to classify the PBC's data. We get a Precision, Recall and F1-score at the test dataset **86.71 %**, **85.67 %** and **84.89 %** respectively at the Smear Test dataset. This is a bit more than doing the normal classification, on the 8 classes using the Xception model. This is hapenning basically due to the fact that the model is able to learn the distribution somehow and approximating on the unseen distribution of the multinomial multivariate data distribution, by including extra information from both the datasets. The confusion metrics obtained can be shown in Figure 6.10, and the graph of loss and accuracy of the source and domains are shown in Figure 6.11.

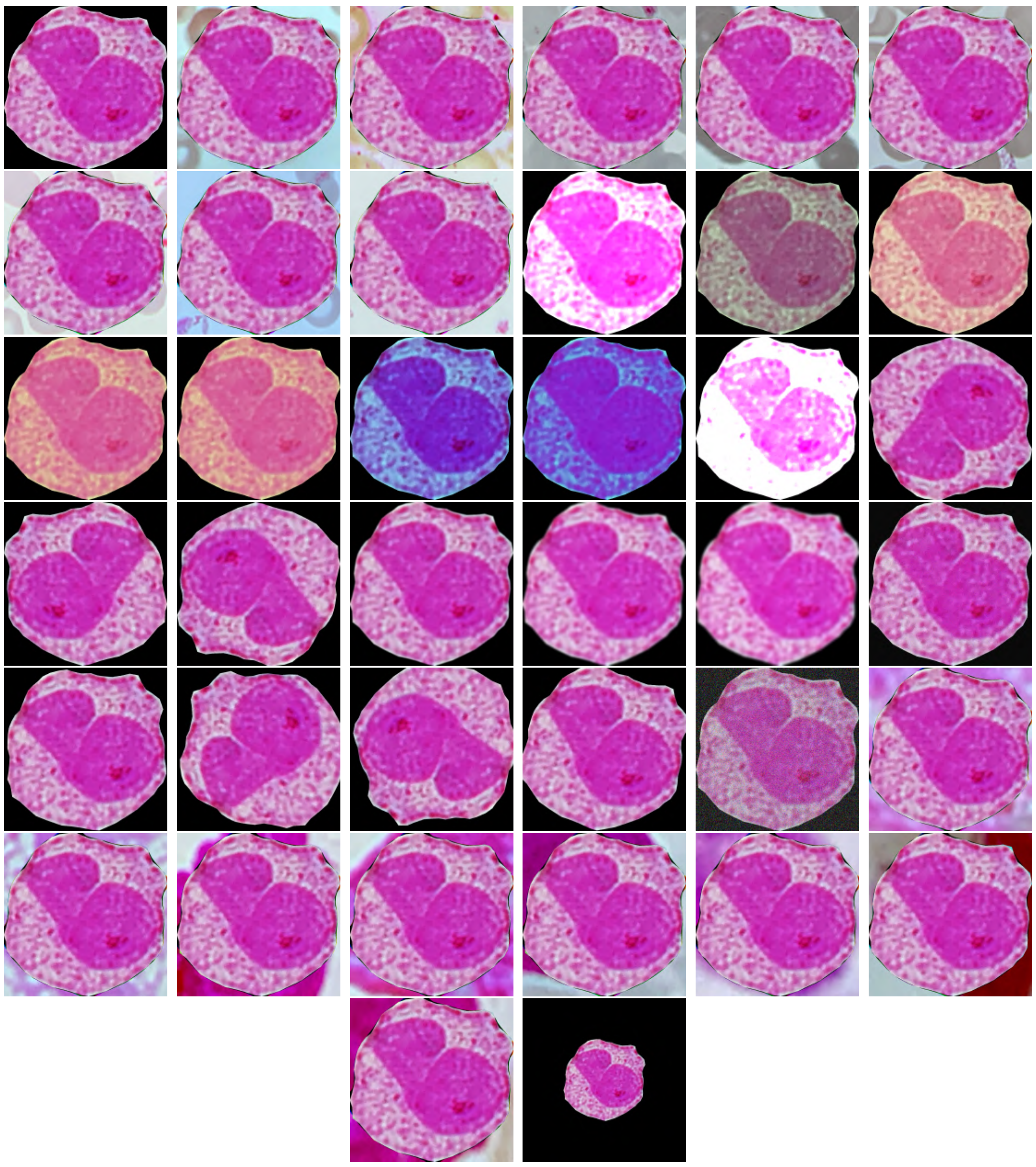


Figure 6.7: Augmentations applied on a single image of Monocyte cell. From top left: (1) Monocyte Cell, (2-9) Applying background on the cell, (10) Image with maximum brightness, (11-16) Applying different color variant to the image, (17) Image with maximum contrast, (18-20) different flips, (21-23) Gaussian blur repetitive with kernel sizes 5,11,17, (24-27) Rotating images, (28) Salt and Pepper noise, (29-36) Applying stain background, (37) Maximum zoom out of image. The transformations are applied to the original image, and in the augmentation module, these are applied recursively on a single image at random.

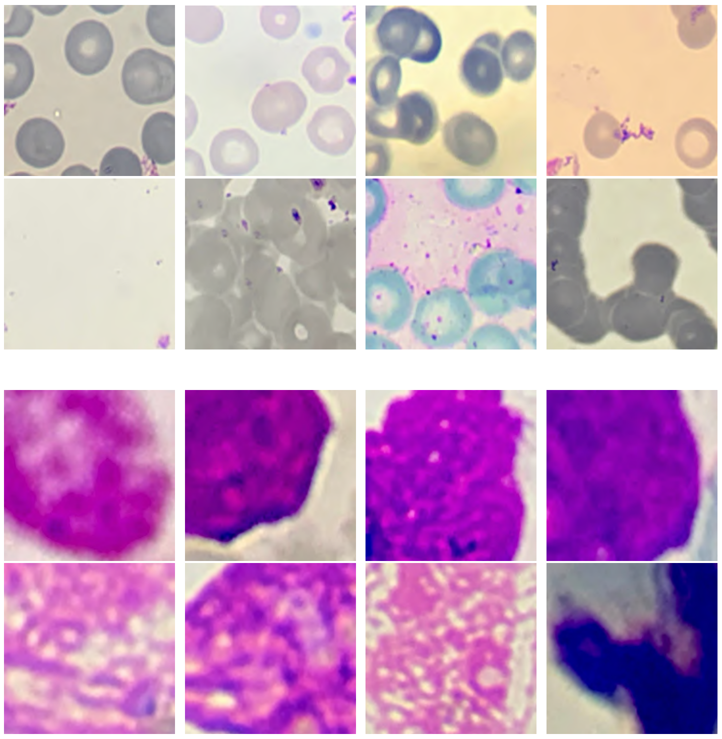


Figure 6.8: Top: Textures of normal background images, Bottom: Textures of stain background.

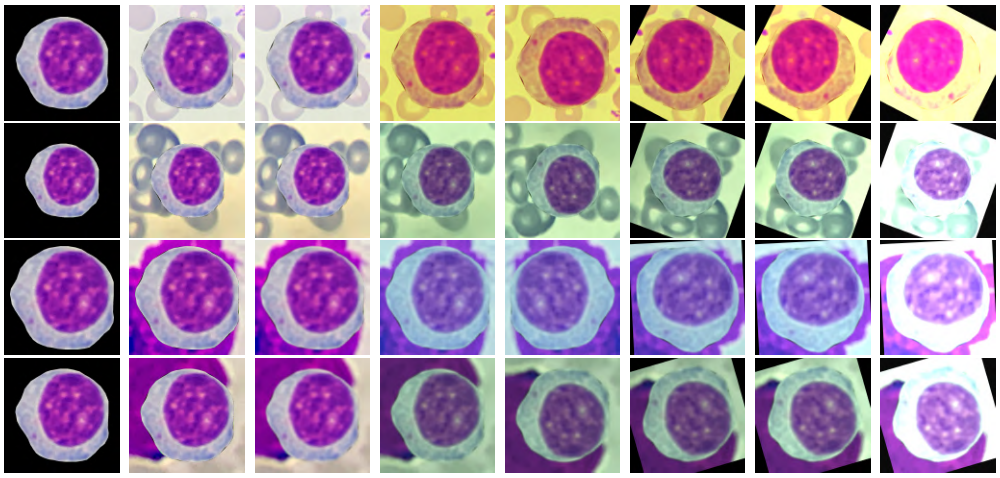


Figure 6.9: Series of random augmentations applied to a cell in the top left recursively. These types of augmentation are common while creating the training set of the Domain Adaptation pipeline.

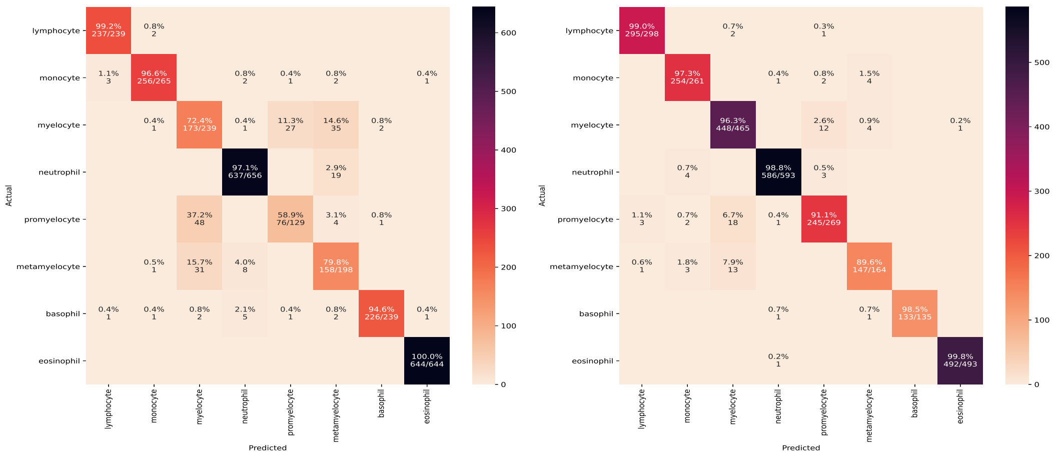


Figure 6.10: Confusion matrix by performing domain adaptation on Source as Smear and Target as PBC dataset. (Left: Smear, Right: PBC)

The Precision, Recall and F1-score obtained by testing on PBC Cropped dataset are **93.11 %**, **93.21 %** and **93.37 %**. This is a bit lower because we are learning features from Smear dataset, to classify PBC cropped dataset in unsupervised way, which is lowering the metrics.

6.7 Domain Adaptation Source : PBC and Target : Smear

Here, by training on the PBC data, we get to train unlabeled Smear data. The Precision, Recall and F1-score at the test dataset for PBC 8 Cropped **95.64 %**, **95.77 %** and **95.89 %** respectively. This is the highest metrics obtained on the 8 Cropped datasets, since the learning of extra features helps in classifying better for the PBC 8 Cropped dataset. Similarly, when tested on Smear Slides 8 Cropped dataset, we get a Precision, Recall and F1-score of **78.89 %**, **78.31 %** and **77.11 %**, which is lesser since, the features are learned in an unsupervised way. The confusion metrics are shown in Figure 6.12 and the graph of training for domain adaptation is shown in Figure 6.13.

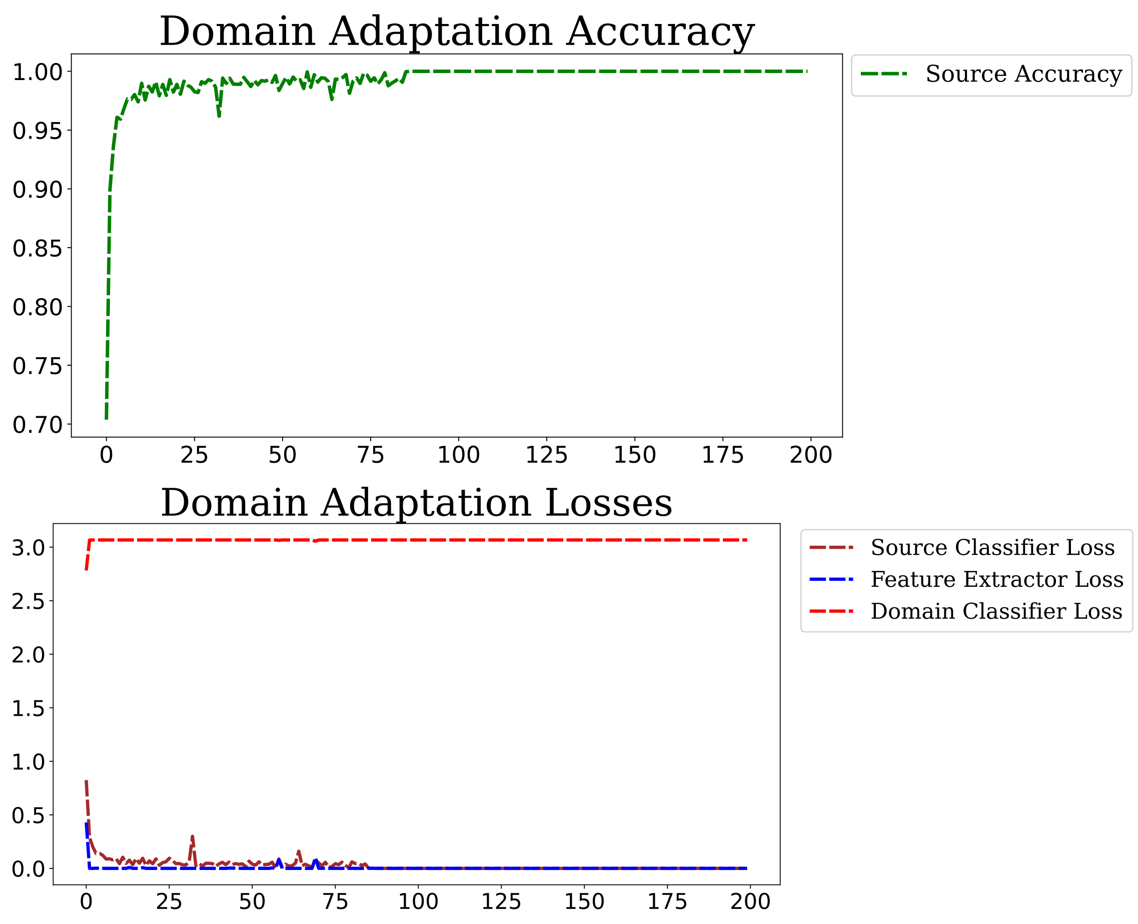


Figure 6.11: Top: Accuracy of Source Domain Images, and Bottom: Losses of Source Classifier (Categorical Crossentropy), Related Feature Extractor and Domain Loss (Binary Crossentropy).

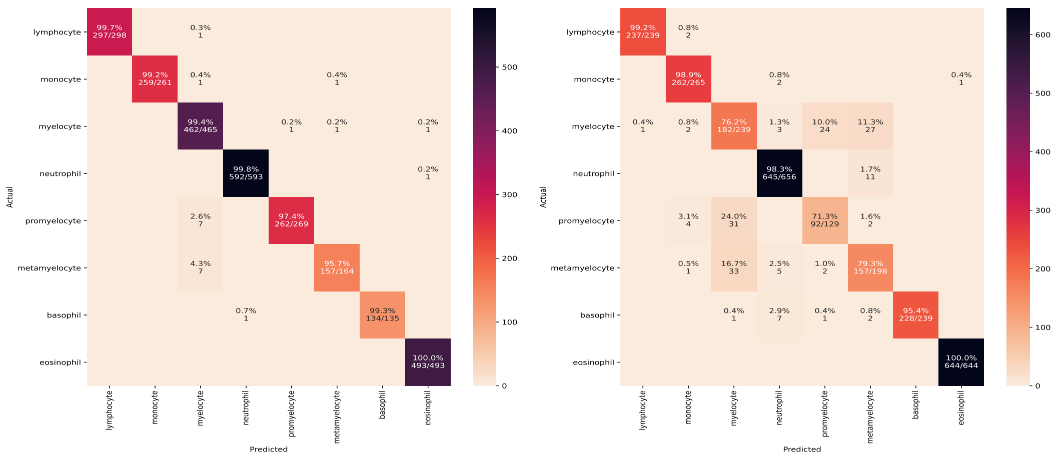
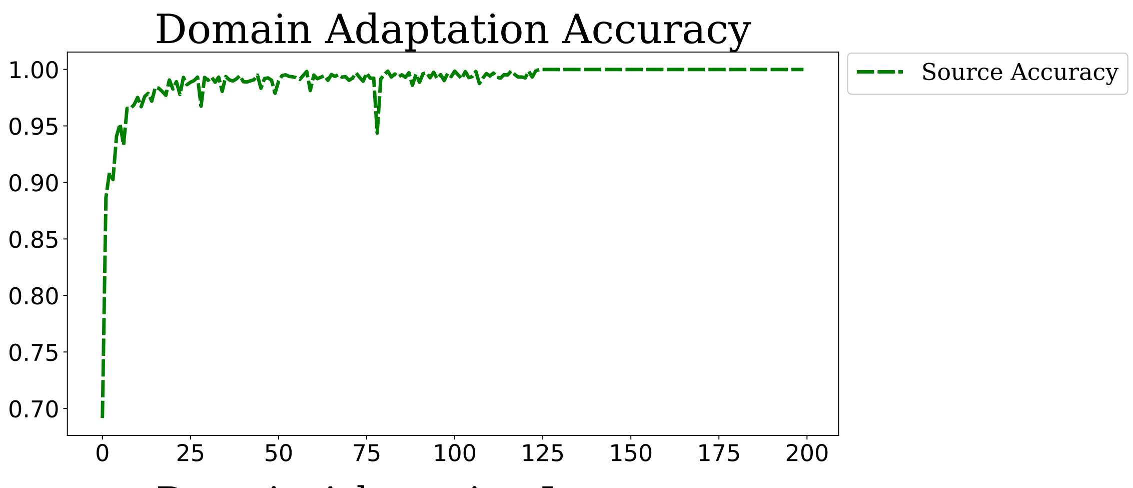


Figure 6.12: Confusion matrix by performing domain adaptation on Source as PBC and Target as Smear dataset. (Left: PBC, Right: Smear)



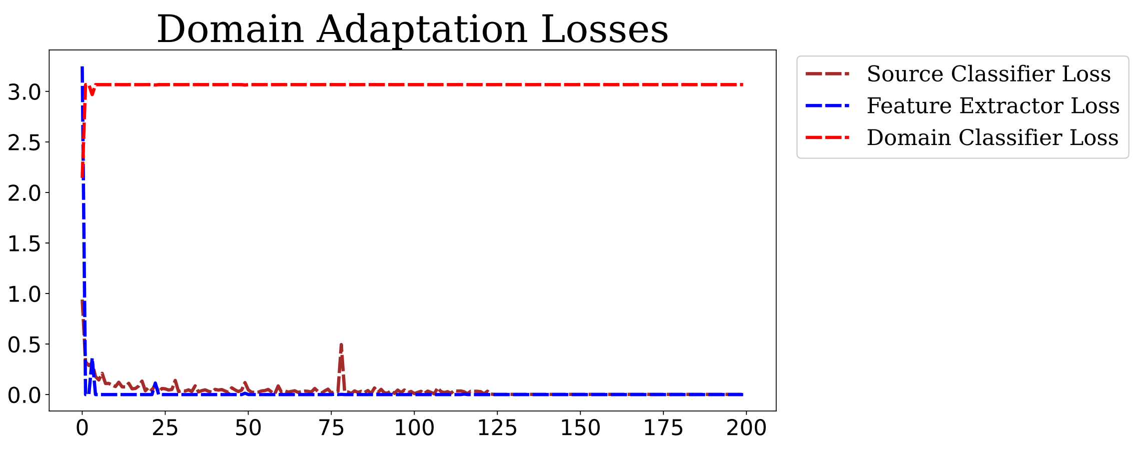


Figure 6.13: Top: Accuracy of Source Domain Images, and Bottom: Losses of Source Classifier (Categorical Crossentropy), related Feature Extractor and Domain Loss (Binary Crossentropy).

6.8 Results Summary for the Cropped Common Classes Dataset

The best results is obtained when we are performing domain adaptation to learn the extra features which was not learnt before as shown in Table 6.1. We will be using this model to classify the 8 common classes after classifying from the final model. The full pipeline is shown in Figure 6.14, which comprises of the Mask RCNN part, the classifier part and the Domain Adaptation module. The Mask RCNN part extracts the exact mask of the cells, the classifier part detects which class the cells belong to for all the 10 classes, and the domain adaptation module refines the classes for 8 class detection.

The final results of the test dataset is shown in Figure 6.15 and Figure 6.15. The general distribution of smear slides remains cluttered and mixed classes all together, which can be seen in Figure 6.17 and Figure 6.17. The upper part of the bounding box has the class detection of Mask RCNN and the lower part has the detection by Domain Adaptation pipeline. These two combinations can give almost near accurate results for the classes of cells detected. Of course, more data will help.

Remarks	Precision (in %)	Recall (in %)	F1-score (in %)
Trained on PBC 8 Cropped	94.78	94.77	94.79
Above model when used to classify Smear Slides 8 Cropped	4.47	21.29	7.45
Trained on Smear Slides 8 Cropped	82.64	82.25	81.74
Above model when used to classify PBC 8 Cropped	53.34	24.36	28.29
Model trained on mixed and classifying PBC 8 Cropped	95.21	95.45	95.37
Model trained on mixed and classifying Smear Slides 8 Cropped	83.98	83.73	84.34
Domain Adaptation Source: PBC Cropped Target: Smear Slides 8 Cropped, Testing on Smear Slides Cropped	78.89	78.31	77.11
Domain Adaptation Source: PBC Cropped Target: Smear Slides 8 Cropped, Testing on PBC 8 Cropped	95.64	95.77	95.89
Domain Adaptation Source: Smear Slides Cropped Target: PBC 8 Cropped, Testing on Smear Slides 8 Cropped	86.71	85.67	84.89
Domain Adaptation Source: Smear Slides 8 Cropped Target: PBC 8 Cropped, Testing on PBC 8 Cropped	93.11	93.21	93.37

Table 6.1: The final results summarized for the 8 Common Classes dataset, and fine-tuning on Xception model using a learning rate of 1e-04 and Adam Optimizer.

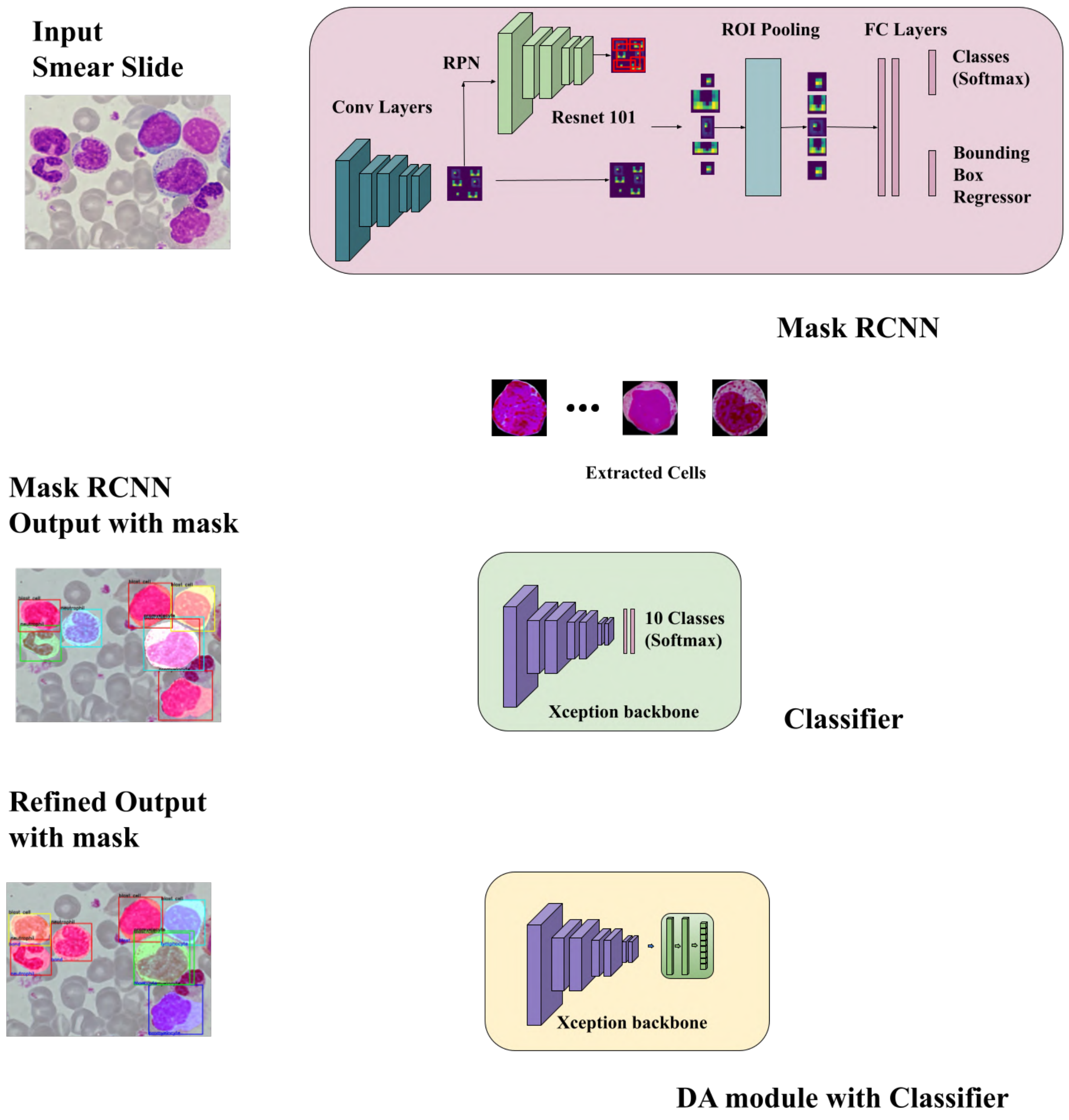


Figure 6.14: Full pipeline including Mask RCNN, Classifier and Domain Adaptation module for accuractely segmenting and classifying cells.

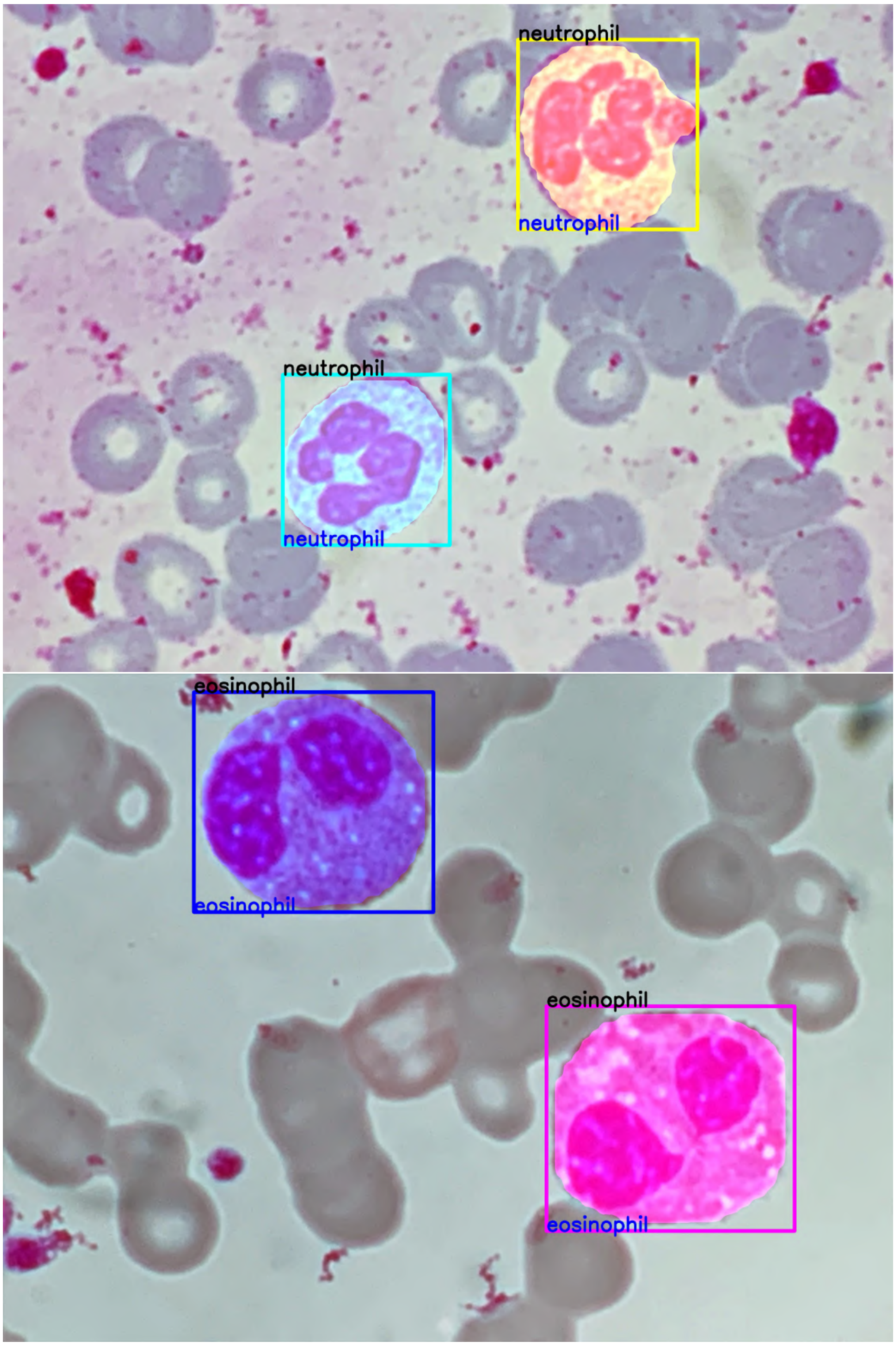


Figure 6.15: The results of test dataset with domain adaptation pipeline.

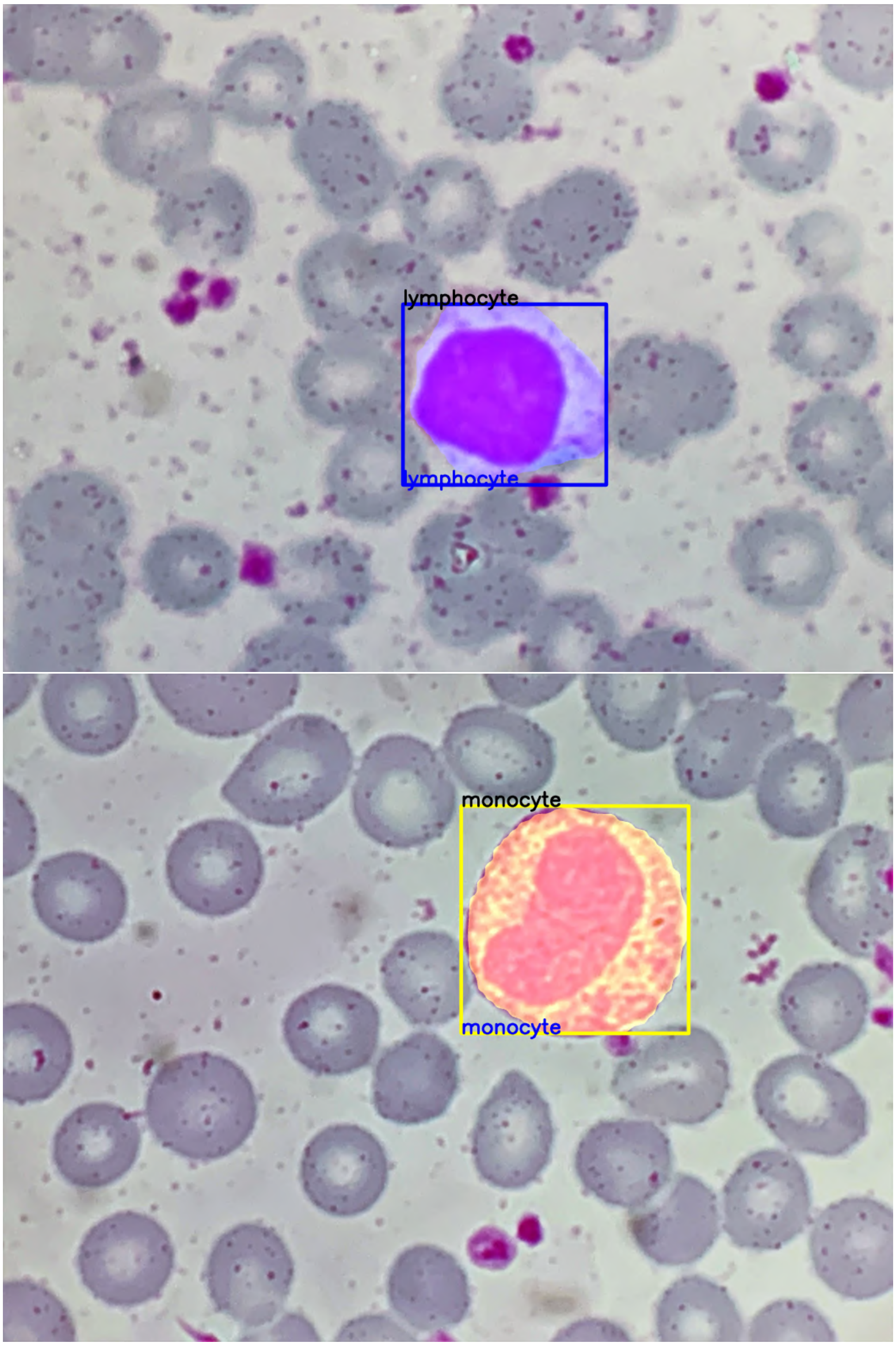


Figure 6.16: The results of test dataset with domain adaptation pipeline.

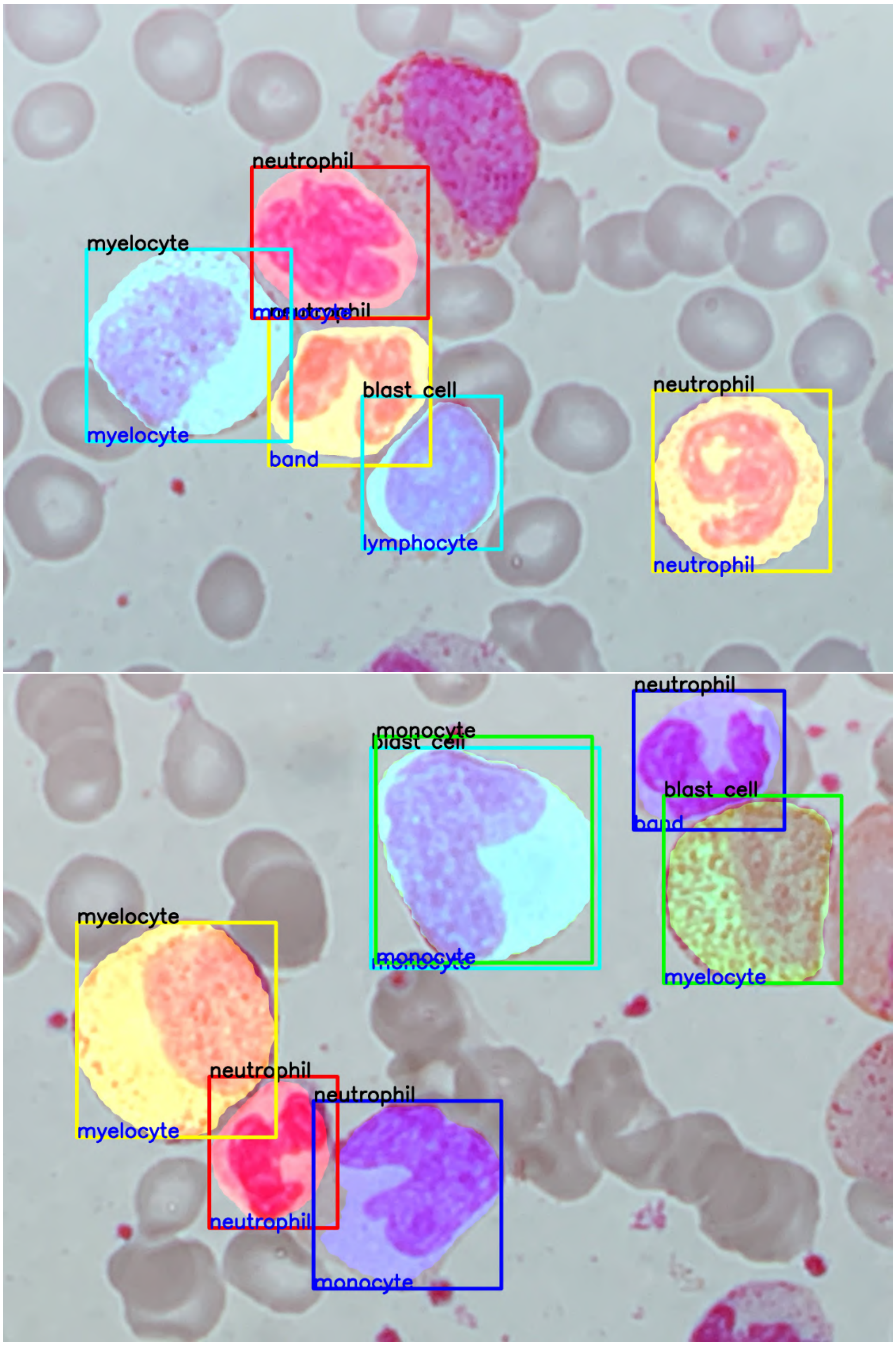


Figure 6.17: The results of unseen mixed classes with domain adaptation pipeline.

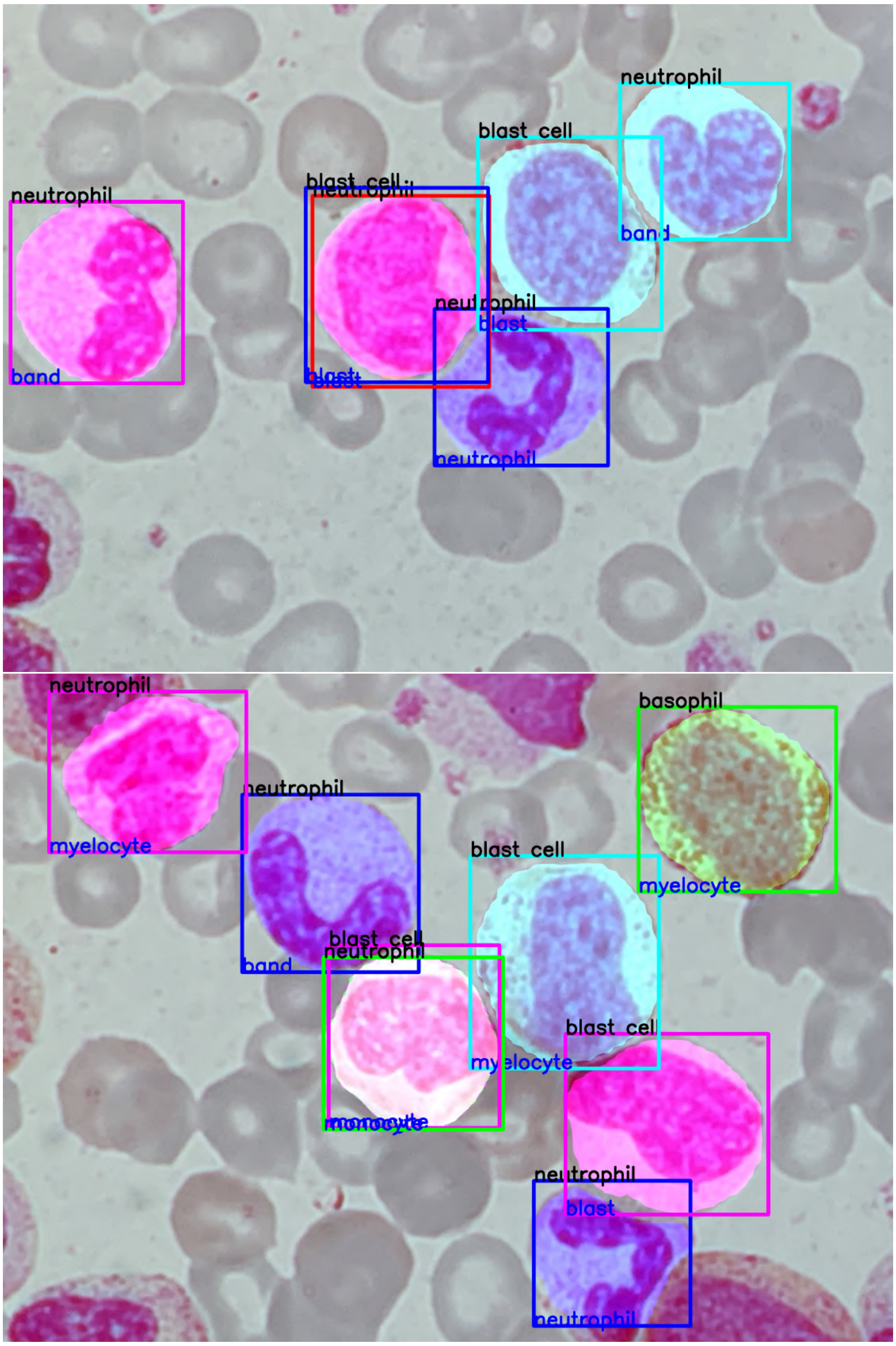


Figure 6.18: The results of unseen mixed classes with domain adaptation pipeline.

Chapter 7

Conclusions

In this study we build a pipeline for automated masking of Peripheral Blood Smear images which can detect and segment regions of interest comprising of white blood cells using Mask RCNN. This method will help to create and hasten the creation of a novel blood smear dataset according to Kolkata's demography. Since collection and annotation of data is an expensive process, for the time being we have used very limited data for our particular task. Using domain adaptation, the improvement in various metrics have been reported. The convolution backbone of the Domain Adaptation pipeline has been selected from the state-of-the-art models which performed best in both PBC cropped and Smear Slides Cropped datasets.

Having surpassed the state of the art as reported in PBC paper, we present a novel contribution in classification of the data, which concludes that a cell is identified more accurately when we have neighbors. When the dataset was in the verge of making, we have tried various experimental methods for masking and detection, such as our own novel GRAD-CAM based detection, which can be improved furthermore using attention-based mechanisms. Having experimented our own classical methods, we can perform satisfactory jobs when the data is less and we need to create our own masking methods. We have also created a novel PBC cropped dataset which goes hand to hand with domain adaptation.

Future scope lies in generating real like images using Generative Adversarial Neural Networks. This can be done only when we have sufficient data for the task. Another area of research which can be explored is inventing a novel algorithm for color transfer, which can convert PBC 8 dataset at ease to Smear Slides Cropped 8 dataset. This will help to create similar dataset or transform a dataset to a target domain by learning features from both the domains. This will be significantly different from Neural Style Transfer and other known algorithms.

In conclusion we can say that this work will be deployable in real world scenario only when we get more data which helps in more precise masking and helping to understand the cells better. This will help doctors and practitioners to automate their daily job of counting and reporting blood smear statistics for blood test. Having done relatively good job in classifying and detecting cells, this knowledge can be used to actively create a better dataset.

Appendix A

Evaluation Metrics

In this work the following evaluation metrics were used:

- True Positive (TP)
- True Negative (TN)
- False Positive (FP)
- False Negative (FN)
- Accuracy (AC) $= \dfrac{TP + TN}{TP + TN + FP + FN}$
- Recall or Sensitivity (SE) $= \dfrac{TP}{TP + FN}$
- Precision (PC) $= \dfrac{TP}{TP + FP}$
- F1-Score $= \dfrac{2 \times (PC \times SE)}{PC + SE}$

The following Loss Function were used:
Binary Cross-Entropy, which can be written as :

$$L_{y'}(y) := -\frac{1}{N} \sum_{i=1}^{N} (y'_i \log(y_i) + (1 - y'_i) \log(1 - y_i))$$

Where, y_i is the predicted class , y'_i is the original class value. (log = ln, natural log)

Categorical Cross-Entropy, which can be written as :

$$L_y(y') := -\frac{1}{N}\left(\sum_{i=1}^{N} y'_i \cdot \log(y_i)\right)$$

Where y_i is the predicted class, and y'_i is the true value of a label predicted, generally in one-hot-encoded form.

Bibliography

[1] Acevedo, Andrea; Merino, Anna; Alferez, Santiago; Molina, Ángel; Boldú, Laura; Rodellar, José (2020), "A dataset for microscopic peripheral blood cell images for development of automatic recognition systems", Mendeley Data, V1, doi: 10.17632/snkd93bnjr.1.

[2] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. `https://github.com/matterport/Mask_RCNN`, 2017.

[3] Andrea Acevedo, Santiago Alférez, Anna Merino, Laura Puigví, and José Rodellar. Recognition of peripheral blood cell images using convolutional neural networks. *Computer Methods and Programs in Biomedicine*, 180:105020, 2019.

[4] Serge Beucher. Watershed, hierarchical segmentation and waterfall algorithm. In Jean Serra and Pierre Soille, editors, *Proceedings of the 2nd International Symposium on Mathematical Morphology and Its Applications to Image Processing, ISMM 1994, Fontainebleau, France, September 1994*, volume 2 of *Computational Imaging and Vision*, pages 69–76. Kluwer, 1994.

[5] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.

[6] Nazli Farajidavar. Transductive transfer learning for computer vision., 2015.

[7] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 1180–1189. JMLR.org, 2015.

[8] Hao Guan and Mingxia Liu. Domain adaptation for medical image analysis: A survey, 2021.

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[11] Han-Kai Hsu, Wei-Chih Hung, Hung-Yu Tseng, Chun-Han Yao, Yi-Hsuan Tsai, Maneesh Singh, and Ming-Hsuan Yang. Progressive domain adaptation for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[12] Genta Ishikawa, Rong Xu, Jun Ohya, and Hiroyasu Iwata. Detecting a fetus in ultrasound images using grad CAM and locating the fetus in the uterus. In Maria De Marsico, Gabriella Sanniti di Baja, and Ana L. N. Fred, editors, *Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods, ICPRAM 2019, Prague, Czech Republic, February 19-21, 2019*, pages 181–189. SciTePress, 2019.

[13] I. T. Jolliffe. *Principal Component Analysis and Factor Analysis*, pages 115–128. Springer New York, New York, NY, 1986.

[14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[15] Michael Linden, Jerrold M. Ward, and Sindhu Cherian. *Hematopoietic and Lymphoid Tissues*, pages 309–338. Elsevier Inc., 2012. Copyright: Copyright 2013 Elsevier B.V., All rights reserved.

[16] Shuying Liu and Weihong Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[17] Zak Murez, Soheil Kolouri, David Kriegman, Ravi Ramamoorthi, and Kyungnam Kim. Image to image translation for domain adaptation. In *2018*

IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4500–4509, 2018.

[18] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[19] H. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 2007.

[20] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, 2017.

[21] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016.

[22] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[23] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2962–2971, 2017.

[24] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[25] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[26] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018.

RAMAKRISHNA MISSION VIVEKANANDA UNIVERSITY
SCHOOL OF MATHEMATICAL SCIENCES
SCHOOL OF INDIAN HERITAGE

Department of Computer Science
Ramakrishna Mission Vivekananda Educational and Research Institute
Kolkata - 711202, India