White Blood Cell Differential Counting in Blood Smears via Tiny YOLO

Sharon Newman
Department of Electrical Engineering
Stanford University
newmans@stanford.edu

Therese Persson
Department of Electrical Engineering
Stanford University
therese@stanford.edu

Abstract

White blood cell (WBC) density and relative abundances in whole blood are quantitative indications of our immune system state, and are typically the first line of disease screening. Automation of WBC differentiation and counting is necessary to increase the throughput of diagnosis in hospitals and decrease diagnostic costs in low-resource areas. This work leverages deep learning by inputing blood smear images into a Tiny YOLOv2 model that differentiates between WBC types and count total subtype ratios. Our work extends state of the art works by incorporating both WBC subtype classification and counting. Our results are above the baseline for our validation set, and has a 0.82 mAP for test. We are enthusiastic for further applications of our model. All code can be found at: https://code.stanford.edu/newmans/CS230

1 Introduction

White blood cell (WBC) density and relative abundances in whole blood are quantitative indications of our immune system state. Testing for changes in baseline levels of different WBC's are necessary for a myriad of diagnostic tests [2], and is conducted in most blood tests. For example, screening for Leukemia is typically done by first counting the lymphocyte (type of WBC) ratio of a blood smear[1]. These differences can be seen in Figure 1.

This project uses deep learning to automatically classify and count white blood cell subtypes from microscopic blood smear images. The input to our model are blood smear images of multiple types of white blood cell types including Eosinophils, Lymphocytes, Monocytes, and Neutrophils. We use a Tiny YOLOv2 model that localizes and classifies the cells to different classes. The output are counts of each of the above stated WBC types. Incorporating both differentiation and counting multiple cell types in a blood smear is a necessary task that we have not seen in current implementations for WBC classification.

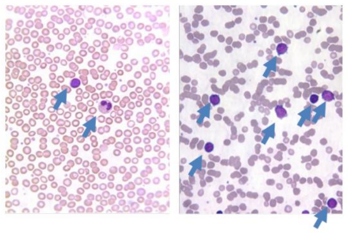


Figure 1: Blood smears of (top) healthy patient with low lymphocyte count and (bottom) Leukemic patient with high lymphocyte count [1]

2 Related work

Currently in hospitals, differentiation and counting is done manually by visual inspection (labor intensive, requires a trained technician, and prone to human error) or via a Coulter counter or flow cytometer that costs upwards of $10k and are only able to classify cells based on size [2, 3]. These systems are outdated and can significantly benefit from faster and more reliable classification methods.

Various works have attempted classifying white blood cells, but none were found by the authors that both classify and count all four blood cell types in an image[6, 8, 9]. The highest precision found with a reasonable methodology classifies single WBC images with 0.912 mAP using a novel "WBCsNet" architecture [5]. Shahin et al. have access to a large privately obtained WBC image dataset and tested a variety of models [5]. Of most interest is their visualization of the last layer weights for their models which showed why poor performing models are unable to capture WBC features. As a baseline however, we decided to work towards Mooney's accuracy of 0.835 mAP with a LeNet architecture as his work uses the same publicly available dataset as used here [7]. This is not a direct comparison to our work as Mooney also only classifies images with single WBC's. Finally, our work extends the state of the art by incorporating both WBC subtype classification and counting.

CS230: Deep Learning, Spring 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 Dataset and Features

The dataset obtained from kaggle contains 368 original histology stained cell images of size 640 x 480 [7]. Of these, 351 are of single cells. Of that set, 20 are Monocytes, 33 are Lymphocytes, 207 are Neutrophils, 88 are Eosinophils, 3 are Basophils (Figure 2). The images were all taken from a single microscope and are labeled in a csv array where each row holds the image name and the corresponding cell type.

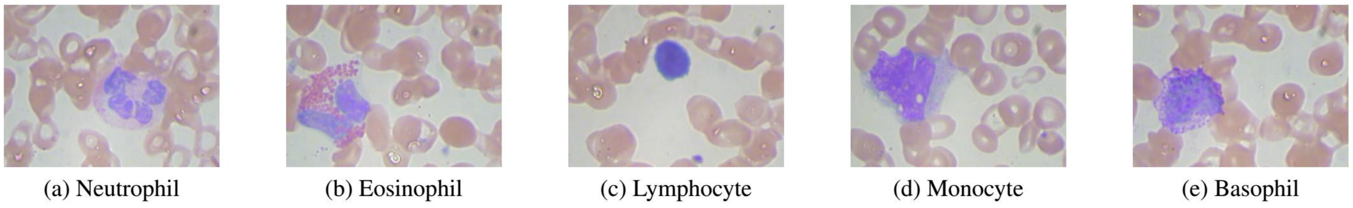


Figure 2: Examples of stained white blood cell types from the dataset.

3.1 Preprocessing

Several steps were performed to preprocess the dataset before training the network. This includes dividing the dataset into training, validation and test sets, performing data augmentation as well as labeling the data (Figure 4). Details are as follows.

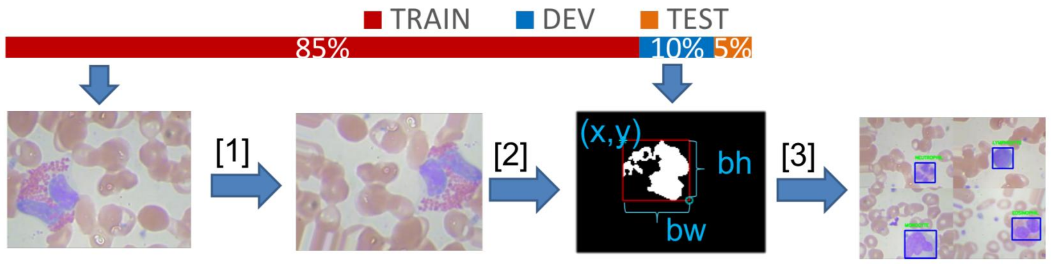


Figure 3: Overview of the preprocessing steps. [1] Training data is augmented. [2] All images are automatically relabeled. [3] 15,000 Artificial multicell images are created.

Data cleaning and splitting: Prior to splitting, all Basophil images were removed, since the dataset only contained 3 samples of Basophils, which is too few to reasonably train our network with. The remaining original single cell images were then split into training (85 %), validation (10 %) and test (5 %) sets. The images in split into each set were randomly selected from the original dataset while ensuring validation and test sets have an equal distribution of cell types (Table 1).

	Training	Validation	Test	Total
Neutrophil	195	8	4	207
Eosinophil	76	8	4	88
Lymphocyte	21	8	4	33
Monocyte	8	8	4	20
Total	300	32	16	

Table 1: Split of the original dataset to training, validation, and test.

Data augmentation: As seen in Table 1, the training set has a skewed distribution of cell types particularly over-representing neutrophils. Data augmentation was implemented to balance the training set and artificially add more training examples. The augmentation was carried out by rotating, flipping, skewing, applying Gaussian noise, changing contrast, and implementing pixel dropout to images. Additionally, classes that were underrepresented in the dataset were oversampled during augmentation for a balance class distribution. After augmentation, the training set consists of 2,000 images (500 images of each cell type).

Stitching of images: The main goal for this project is to detect and count the number of cells in each image, however the dataset used mainly contains images of single cells. To get data that better reflects the aims of the project, the images were therefore randomly stitched together in a 2x2 grid where each quadrant holds a unique cell. By doing this the size of the training, validation and test sets were increased to 15,000 images, 200 images and 100 images respectively.

Labeling: In parallel with stitching the augmented multicell images, relabeling had to be done to suit the classification and detection task at hand. The labels for the original files are in csv-format and only contain the name of the depicted cell type (no localization). The implementation of object detection and classification in our repository requires labels in xml-format encoding bounding boxes and class types. Since this xml-format is easy to scale to multiple objects, we decided to keep this labeling schema. To localize the cell bounding boxes, we decided against manual labeling and implemented a series of image segmentation steps (gray-scale conversion, thresholding, centroid calculation) for our images. This script detected the center and bounding boxes of the single cell images, and created xml-format labels with these values. Figure4 shows an example thresholded cell with its corresponding bounding box.

4 Methods

To automate counting of blood cells we need an algorithm that is able to both detect and classify cell types with high accuracy. There are several possible deep learning algorithms that could be used for this purpose such as Fast R-CNN[11], Deformable parts model (DPM)[12] and You Only Look Once (YOLO)[13]. Redmon et al. show that YOLOv2 is capable of training with higher resolution images and achieve higher mAP than Fast R-CNN. The ability to analyze high resolution images is important for future applications in cell histology classifications as small changes is cell types can be indications of different diseases, and thus we chose to start with YOLOv2–also very fast object detection algorithm.

4.1 YOLOv2

The YOLO algorithm uses a single convolutional neural network to perform both object detection and classification during one evaluation of an image by handling it as a single regression task. The algorithm looks at features from the whole image when predicting bounding boxes for where objects are located[13]. Below follows a summary of the steps in YOLO and how they are applied to our problem.

- The image is divided into an $S \times S$-grid.
- The square that contains the center of a white blood cell is responsible for detecting that cell.
- Each grid square predicts bounding boxes and a confidence for each box. The confidence is calculated as: $confidence = Pr(object) * IOU^{true}_{pred}$. The bounding box prediction consists of the box's position *(x,y)*, the box's size *(w,h)* and the confidence *(p_c)*.
- Each grid square also predicts the cell type (class) for the detected bounding boxes. The predicted values for the cell classes represent conditional probabilities for the respective cells, given that the square contains an object.
- YOLOv2 uses anchor boxes[14] to account for if multiple objects are located in the same grid square. The number of anchor boxes and their sizes can be varied depending on the size of the objects.

The predictions for this specific task with one anchor box can be summarized as

$$\hat{y} = [p_c, x, y, w, h, c_{neutrophil}, c_{eosinophil}, c_{lymphocyte}, c_{monocyte}]^T \tag{1}$$

4.2 Non-max-suppression

When the network has predicted bounding boxes it is likely that too many boxes have been predicted. The reason is that only the grid square containing the center of a cell should predict a bounding box, but often adjacent grid squares containing parts of the cell will also predict a bounding box. To avoid this problem, non-max suppression (nms) is performed. This is done by comparing the confidence values of boxes and only keep the boxes with sufficiently high confidence. For boxes that have a large overlap (high IoU), only the box with the highest confidence is kept (Bottom of Figure 4).

4.3 Network architecture

The YOLO implementations used in this work builds upon Ngoc Anh's implementation[10] of YOLO in Keras. We have explored two different types of YOLO architectures to perform blood cell detection, Full YOLOv2 and Tiny YOLO. Both models have been trained with pretrained YOLO-weights. Full YOLOv2 contains 22 convolutional layers with batch normalization and uses leaky ReLU as activation functions. The network also contains a skip connection between layer 13 and layer 21. Tiny YOLO is a simpler implementation of YOLO with fewer layers, it contains 8 convolutional layers with similar structure as for Full YOLO but no skip connections. The full network is depicted in Figure 4 along with non max suppression.

4.4 Loss function

The following loss function proposed by Redmon was used [13]:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} L_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} L_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

$$+\lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} L_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} L_{ij}^{noobj} (C_i - \hat{C}_i)^2 + \sum_{i=0}^{S^2} L_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

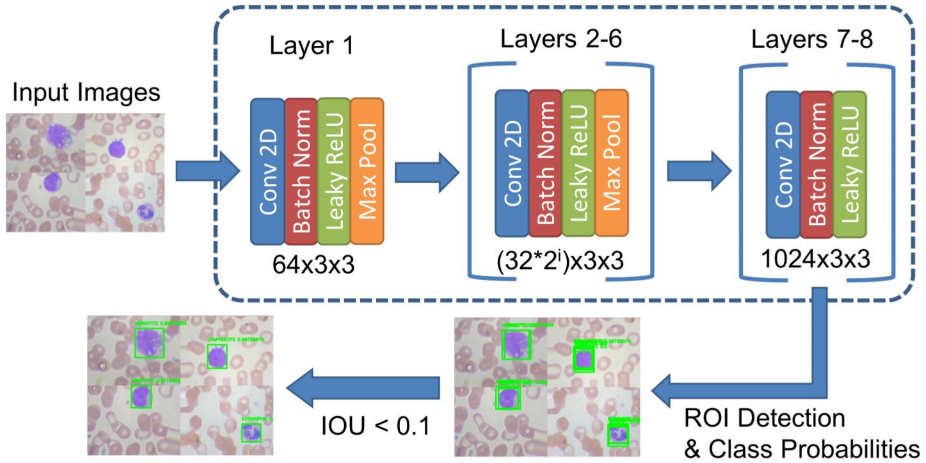


Figure 4: Overview of TinyYOLO with non max suppression steps

This loss function penalizes both when the algorithm predicts the wrong cell type and if the algorithm predicts a bounding box at an incorrect location. The scale factors λ_{obj}, λ_{coord} and λ_{noobj} determine how much to penalize coordinate and classification errors. λ_{obj} is a scale factor our original repository incorporated into the loss function. L_{ij}^{obj} is 1 for the i:th grid square and j:th bounding box predictor and 0 otherwise. L_{i}^{obj} is 1 if an object appears in the i:th grid square.

5 Experiments/Results/Discussion

5.1 Metrics

We use the mean average precision (mAP) to evaluate the performance of our network. The average precision is defined as $\sum_{i=0}^{x} P_i * \Delta r_i$ where P_i is the precision for the i:th first subsets and Δr_i is the change in recall between subset i-1 and subset i. Precision and recall are defined as

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives + False positives}} \qquad \text{Recall} = \frac{\text{True positives}}{\text{True positives + False negatives}}$$

5.2 Tuning of network

The original parameters of Anh's implementation of YOLOv2[10], did not produce any bounding boxes for WBC's after training with our dataset. The following sections describe our network tuning to improve performance.

Anchor boxes fine-tuning was crucial for the network to find white blood cells. By generating new anchor boxes customized for our dataset instead of using the default anchor boxes, the performance of the network significantly improved.

Network Architecture: Starting off, we used the full YOLOv2 architecture described in section 4.3. After hyperparameter tuning, we had mAP for 1 for train, 0.7965 for dev, and .7761 for test. These results suggested that we were overfitting for the training set. Since YOLOv2 is a rather deep network with 22 layers, we figured that this model was too deep for a small dataset as ours. We switched over to the Tiny YOLO model as stated in section 4.3, which at first yielded very poor results (.4146 training, .3355 dev). This made us work with regularization. Knowing that YOLOv2 is able to reduce overfitting and remove the need for dropout by adding batch normalization, we swept momentum values and in the end got our final values presented in Figure 5a after the rest of our hyperparameter sweeps. Since we are still overfitting for the train set, we tried changing all activation functions from leaky-ReLU to ReLU since ReLU has a smaller possible range of output values and lead to a less expressive model. However this gave slightly worse results for training and validation than with leaky-ReLU activations.

5.3 Hyperparameter sweep

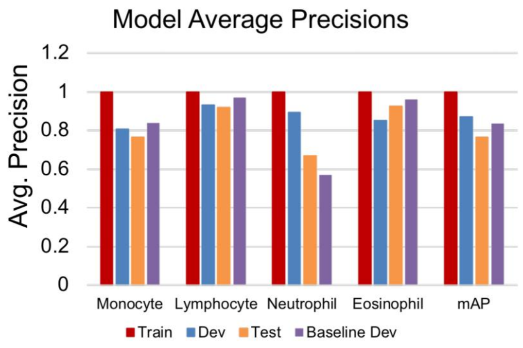
A course grid search of learning rate, batch size, batch normalization momentum (all layers), non max suppression threshold, and various loss function scales was first swept for smaller training and dev sets of 2000 and 200 images respectively. Once a more optimal hyperparameter range was found, a fine random search with all training images was conducted. Table 2 depicts the swept parameters and results for the Tiny YOLO model. NMS threshold was swept since we often got several predicted bounding boxes for each cell. Class and Object scales from the loss function were swept since we were sometimes getting wrong class predictions and no objects detected.

Hyperparameter	Range Swept	Value w/ Optimal mAP
Learning Rate	$10^r, r\exists[-5,0]$	$2*10^-4$
Momentum	$1-10^r, r\exists[-3,-1]$	0.950
Batch Size	1-512	10
Object Scale	1-25	15
No Object Scale	1-25	1
Class Scale	1-25	1
NMS Suppression Threshold	0.05-0.4	0.1

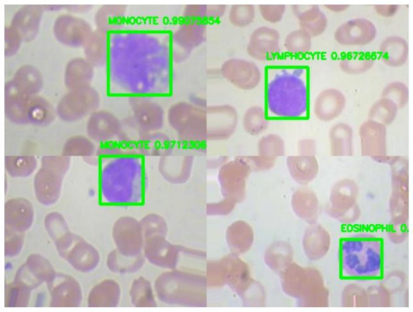
Table 2: Hyperparameter sweep and corresponding optimal values for Tiny YOLO model

5.4 Results

Finally, after architecture changes and hyperparameter sweeps, the Tiny-YOLO model was trained with the optimal hyperparameters as seen in Table 2. The average precisions and mean average precisions (mAP) of training, validation (dev) and test sets are shown in Figure 5a alongside the baseline model precisions of Mooney's work [7]. We achieved a mAP of 0.871 for the validation set and 0.82 for the test set. The can be compared with the baseline value of 0.835 for Mooney's validation set. Mooney only provided results for the validation set, so we are unable to compare our test results.



(a) Average precisions for each dataset and baseline



(b) Result of prediction

6 Conclusion/Future Work

Our results are comparable to the baseline models in terms of classification, with a higher mAP for the validation set than the baseline. However, it is difficult to make a direct comparison as the models in previous works are only classifying single cell images. In this work, we are able to both classify cell types in images with multiple cells and detect where the cells are located. The algorithm is still overfitting to the training set which is expected considering the small amount of original data used in the project. Future work to reduce overfitting includes: [1] finding a larger dataset to train on, [2] obtaining different distribution of images (ie images taken from different microscope), and [3] using regularization methods to reduce overfitting. We are also interested in visualizing the layer weights to see what histological features the network picks up on to classify cell types. This could provide insight to false classifications and more smartly change our architecture. It would also be interesting to try other network architectures such as Fast R-CNN to see if that would improve the accuracy. Additionally, we would like to train the network with images of blood smears containing multiple cells instead of creating our own stitched images to see how the network performs on more realistic data. Finally, we are excited that our model reached very comparable classification mAP to literature while also adding the critical aspect of cell counting. We believe this work has a strong potential to improve the manual methods of cell differentiation and counting in disease diagnosis. All code used in this report can be found at: https://code.stanford.edu/newmans/CS230.

7 Contributions

The work has been divided as equal as possible between the two team members and most of the time we worked together but on slightly different parts of the project. When relabeling the data Sharon was responsible for writing a script to find the position of the bounding boxes and Therese was responsible for writing the script to convert the labels from csv-format to xml-format as well as dividing the dataset and stitching the images. Sharon wrote a bash script for doing hyperparameter sweeping and has been performing the hyperparameter sweeps. Sharon was also responsible for setting up and maintained our Gitlab repository while Therese has been responsible of setting up a Google Cloud instance used for training the neural network. The rest of the work were divided between the group members.

References

[1] Carleton, "Hematology of Leukemia", April 26, 2017, Available from: https://serc.carleton.edu/woburn/overarching/hema_leuk.html

[2] Blumenreich MS. "The White Blood Cell and Differential Count". In: Walker HK, Hall WD, Hurst JW, editors. Clinical Methods: The History, Physical, and Laboratory Examinations. 3rd edition. Boston: Butterworths; 1990. Chapter 153. Available from: https://www.ncbi.nlm.nih.gov/books/NBK261/

[3] PENG L. "Performance evaluation of the Z2 coulter counter for WBC and RBC counting", August 3 2007. DOI: 10.1111/j.1751-553X.2007.00868.x

[4] de Bruijne, M. "Machine learning approaches in medical image analysis: From detection to diagnosis", June 23, 2016. Available from: https://doi.org/10.1016/j.media.2016.06.032

[5] Shahin, A.I et al. "White blood cells identification system based on convolutional deep neural learning networks". November 14, 2017. Available from: https://doi.org/10.1016/j.cmpb.2017.11.015

[6] Xu, M et al. "A deep convolutional neural network for classification of red blood cells in sickle cell anemia". October 19, 2017. Available from: https://doi.org/10.1371/journal.pcbi.1005746

[7] Mooney, T. Kaggle Data set for single WBC blood cell images https://www.kaggle.com/paultimothymooney/blood-cells

[8] Binary classification of WBC https://github.com/dhruvp/wbc-classification/

[9] Xie, W. and Noble, J. A. and Zisserman, A. "Microscopy Cell Counting with Fully Convolutional Regression Networks". MICCAI 1st Workshop on Deep Learning in Medical Image Analysis. 2015.

[10] Huynh Ngoc Anh YOLOv2 in Keras and Applications https://github.com/experiencor/keras-yolo2

[11] Girshick, Ross. "Fast r-cnn." arXiv preprint arXiv:1504.08083 (2015).

[12] Yan, Junjie, et al. "The fastest deformable part model for object detection." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2014.

[13] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[14] Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." arXiv preprint. 2017.