Multi-label Classification of Blood Cells Using Convolutional Neural Network

Weining Hu
MS in Data Science | The George Washington University | whu369@gwu.edu

1 Introduction

This project used *PyTorch* to build a Convolutional Neural Network (CNN) model to recognize all types of cells that are present in each of the given images. These cell types are: red blood cell, difficult, gametocyte, trophozoite, ring, schizont, and leukocyte. The best model over the past seven-day data challenge competition yielded a loss score of 4.54085.

2 Dataset

The training set contains 929 rectangular cell images of varied size, 929 txt files with the corresponding string labels, and 929 json files with the cell detection bounding boxes. During the course of the competition, only the raw images and the txt files were used to train the model. Since the dataset is unbalanced and the raw data (.png and .txt) cannot be directly used to train the network, the following steps were performed to preprocess the data.

2.1 Data Pre-processing

Image Resizing: In order to feed the images into a CNN architecture, I used OpenCV to resize each image into a uniformed shape of (400, 400, 3). Since *PyTorch* requires the channel dimension to be placed before height and width, I transposed the position of the channel dimension after resizing the image.[1] The final shape of the image is (3, 400, 400).

Data augmentation: Exploratory data analysis shows that the training set is extremely unbalanced. The largest class has 206 samples, while the smallest class has only one sample. 25 classes out of the 40 classes contain less than 10 samples each. During the course of the competition, I concentrated the time on loading the data and making the model works. Therefore, image augmentation was not performed on the training set.

Another concern that make me restrained from increasing the number of the minority class images is because certain classes never appear in the training set. Increasing the images of the existing minority classes without doing the same for the non-existing classes may affect the distribution of the labels. Given the time and technical constraints, the primary efforts during the competition were committed to loading data and building training classifier.

Labeling: After reading all string values from each txt file into string lists, I used a pre-defined dictionary to convert each of the newly extracted label list into an integer list, where the appeared cell types are denoted by 1 and the absent cell types are denoted by 0.

Loading Data: Unlike *Keras* in Exam 2, *PyTorch* has its own requirements for formatting the dataset to be loaded. First, I created a dataset parent class[2] to wrap both the inputs and labels. I called the *random_split* API to initialize the train set and validation set. Lastly, I passed the data to *DataLoader*, which manages all the shuffling and does the actual job to read the data into memory. The data loading step consumed me the most time during

[1] https://github.com/pytorch/pytorch/issues/14330
[2] This step is achieved by *Class CellData()* in the train script.

the competition due to the fact that I was not familiar with the subclass structure. It was finally completed after trying out several different methods suggested by multiple online references.[3,4]

3 Modeling

3.1 Network Architecture

In this project, I built a network that contains 12 layers.[5] As illustrated in Figure 1, the model follows the block design pattern, where one convolutional layer (filters of size 3 X 3), one max pooling layer (factor 2), and one *ReLU* activation layer are stacked as a single learning block. The whole model is formed by four learning blocks, followed by three fully connected layer, and lastly following by a sigmoid activation function to render the probabilities of each class independently for classification. I chose *Adam* as the model optimizer and *BCELoss* as the function loss.

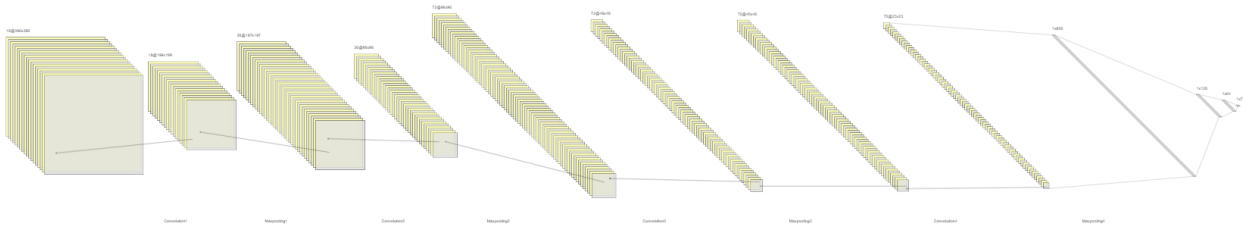

Figure 1. LeNet Style Diagram of the Trained Networks[6]

3.2 Model Optimization and Fine-tuning

After making the model work, I spent most of the time tuning the hyperparameters, including batch size, learning rate, number of epochs, dropout rate, and filter size. The results show that decreasing the dropout rate from 0.5 to 0.05, batch size from 512 to 33, and learning rate from 0.05 to 0.001 can reduce the loss rate from 5.1 to 4.5, but the changes were not significant enough to fundamentally improve the model learning ability.

4 Conclusion/Future Work

In this project, I used only one type of CNN architecture to train the data. There are at least two different methods that can be implemented to improve the current results. Firstly, I can try out different pretrained, state-of-the-art relevant models and even assemble multi-models to collectively train the data. Secondly, image augmentation should be applied to the training set. Instead of simply over-sampling or under-sampling the entire training set to balance all labels before loading the data to the model, latest research work in addressing unbalance dataset for multi-label classification problems show that I can adaptively balance the data in each batch during the training stage. This novel Selective Learning method is proposed by Hand et.al. in AAAI 2018.[7] As time allowed, all these methods can be implemented to improve the model.

---

[3] https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/pytorch/dataloader-and-datasets.html

[4] https://towardsdatascience.com/building-efficient-custom-datasets-in-pytorch-2563b946fd9f

[5] The inspiration for building this building comes from the following reference. https://www.analyticsvidhya.com/blog/2019/04/build-first-multi-label-image-classification-model-python/

[6] This diagram was drawn using this online tool: http://alexlenail.me/NN-SVG/LeNet.html

[7] Hand, Emily M., Carlos Castillo, and Rama Chellappa. "Doing the best we can with what we have: Multi-label balancing with selective learning for attribute prediction." *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16962/16273