Junhe Zhang

Date: 11/23

Classify All Cells in Image

Data Preprocessing:

1. Loading training data using cv2 library and reshape each input image into 256 by 256 columns, the original image has more than 2000 pixel which is unnecessary. Shape of the data contains four dimensions, which are RGB value plus alpha rate. Did not flat the input data this time since I am using Convolutional Neural Networks which can take multi-dimension data.
2. Since each input image could have multiple labels, I chose to use MultiLabelBinarizer function from Scikit-learn library, which will encode image with label “red blood cell” and “difficult” as [1,1,0,0,0,0,0]. There are total of 7 different labels in the order of "red blood cell", "difficult", "gametocyte", "trophozoite", "ring", "schizont" and "leukocyte".
3. Using train-spilt function to divide input data into 75% training and 25% testing.
4. Normalize input data so that the training process could be more efficient.
5. Convert both dependent and independent input variable into tensor object to utilize GPU.
6. Transpose dimension one with dimension three (dimension start with zero). Because the original data has shape batch size, width, height and RGB, but Pytorch need input shape to be batch size, RGB, width and height.

Model Construction:

1. Decide to have four convolutional layers since the input dimension is relatively high. Setting the filter size to be 3x3 to capture more details.
2. Each convolutional layer also combined with batch-norm2d layer.
3. Adding traditional Relu activation function and Maxpooling function into each convolutional layer.
4. Add additional dropout layer into each convolutional layer so that the final model could be less bias.
5. Add three fully connected linear layers to connect each component from convolutional layer.
6. The model was highly inspired by Vatsal Saglani’s blog post which has cited below.

Training model:

1. Using BCELoss as criterion and SGD as optimizer because they seem to be the most suitable function in our case which is multi-label classification.
2. Set batch size to an appropriate number to maximize utility of GPU while not exceed the cache size of the GPU.
3. Do save and load model inference while training the model, so that model will keep optimizing and not overwhelming the GPU or CPU. Keep this process for 5 hours.

Conclusion:

The convolutional neuron network is much more powerful and efficient than multi-layer perceptron. It can classify more complex data and accomplish more challenging task. The input images do not need to be flatten and remain its original flavor. It also seems to have high tolerant to imbalanced data since most of the training data include red blood cell, however, the final result is still amazing which has a much lower loss value like 0.00021.

Work Cited

1. https://medium.com/@thevatsalsaglani/training-and-deploying-a-multi-label-image-classifier-using-pytorch-flask-reactjs-and-firebase-c39c96f9c427