Energy-efficient Machine Learning System Final report

Prof. Yuan
Yao Shi

Topic: (Type B project)
Apply a Neural Network on the White Blood Cell Recognition

Abstract:

Establishing an accurate count and classification for the white blood cells is crucial because it will greatly increase the diagnostic efficiency as well as provide an additional way of cross-validation. There are four different types of cells to be classified in this project: Eosinophil, Lymphocyte, Monocyte, and Neutrophil. The classification will be conducted through a convolution neural network via Tensorflow.

Introduction

Although it is challenging to build a classifier for the white blood cells, the structure of a neural network provides a good insights for this domain problem. The model will be trained via Tensorflow white being monitored using Tensorboard.

Related work

- Deep Learning from Scratch + Insights (https://www.kaggle.com/placidpanda/deep-learning-from-scratch-insights)

- Blood Cell Subtype Identification cudnn+CNN+RNN (https://www.kaggle.com/coder98/blood-cell-subtype-identification-cudnn-cnn-rnn)

Data description

This dataset is downloaded from Kaggle and I give acknowledgement to https://github.com/Shenggan/BCCD_Dataset. This dataset contains 12,500 augmented images of blood cells (JPEG) with accompanying cell type labels (CSV). There are approximately 3,000 images for each of 4 different cell types grouped into 4 different folders (according to cell type). The cell types are Eosinophil, Lymphocyte, Monocyte, and Neutrophil. This dataset is accompanied by an additional dataset containing the original 410 images (pre-augmentation) as well as two additional subtype labels (WBC vs WBC) and also bounding boxes for each cell in each of these 410 images (JPEG + XML metadata). More specifically, the folder 'dataset-master' contains 410 images of blood cells with subtype labels and bounding boxes (JPEG + XML),

while the folder 'dataset2-master' contains 2,500 augmented images as well as 4 additional subtype labels (JPEG + CSV). There are approximately 3,000 augmented images for each class of the 4 classes as compared to 88, 33, 21, and 207 images of each in folder 'dataset-master'.

Method description

- There are two different datasets. The ‘dataset1-master’ contains the original pictures of different types of cells, however, the problem is that these images are not evenly distributed. The ‘dataset2-master’ contains pictures after various image transformations such as minoring, rotation, etc. Therefore, images from ‘dataset2-master’ are used for the model.
- Initially, “Relu” is used for the activation layer, however, the model was often stuck at its local minimum during the training phase. After several experiments, “LeakyRelu” replaces “Relu” for a better performance.
- After experimenting with “Batch normalization”, it produces fast learning rate comparing to “dropout”, however, it tends to overfit the network pretty quickly as well. When using “batch normalization”, the validation accuracy always oscillates significantly as the the training accuracy increases along the number of epochs. Therefore, this model only uses “dropout” instead of “Batch normalization”.

Model description

- After preprocessing, the labels are translated using the “one hot encoding” and all images are resized into a dimension of “80 x 80” using OpenCV. The model summary is listed below. In addition, Adam optimizer is applied to this model.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 78, 78, 32)        896
_________________________________________________________________
activation_12 (Activation)   (None, 78, 78, 32)        0
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 39, 39, 32)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 37, 37, 32)        9248
_________________________________________________________________
activation_13 (Activation)   (None, 37, 37, 32)        0
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 18, 18, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
dropout_6 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                262208
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 64)                0
_________________________________________________________________
dropout_7 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_9 (Dense)              (None, 5)                 325
_________________________________________________________________
activation_14 (Activation)   (None, 5)                 0
=================================================================
Total params: 291,173
Trainable params: 291,173
Non-trainable params: 0
_________________________________________________________________
```

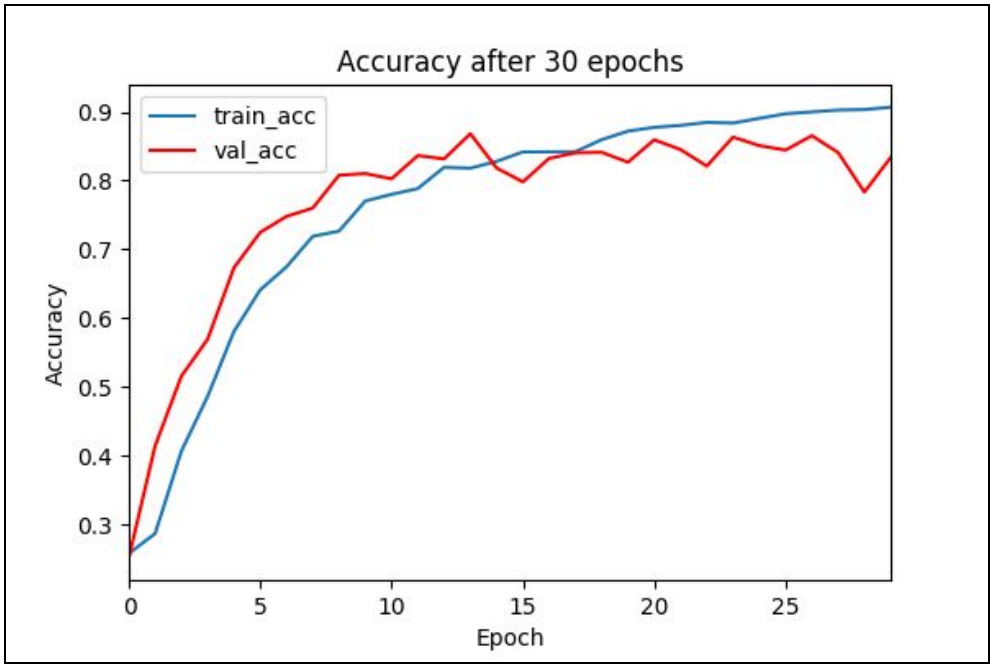
Experimental procedure

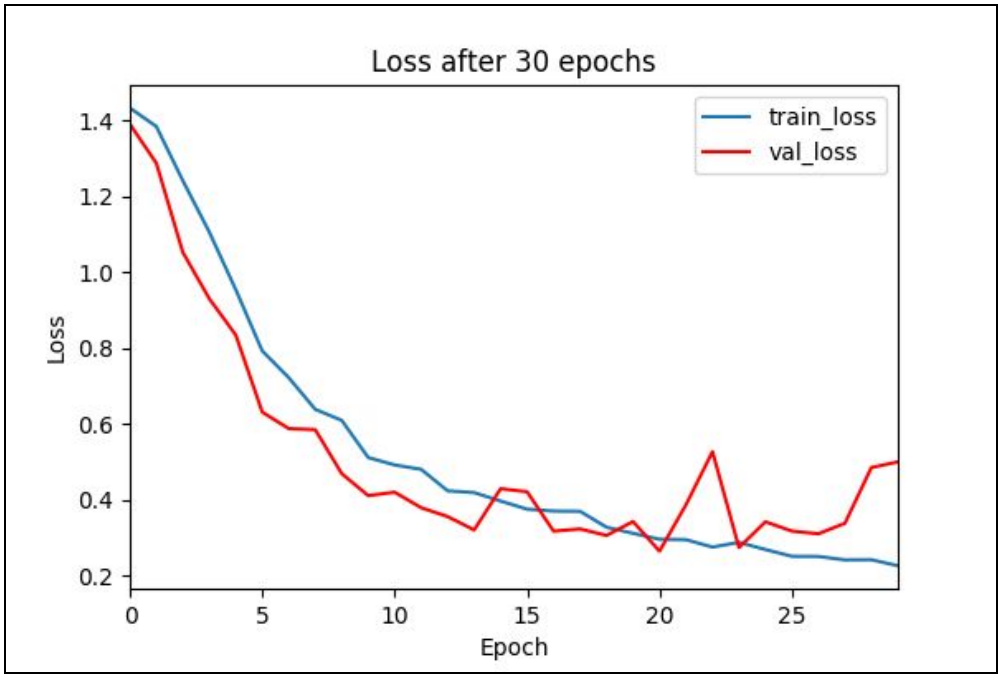
- Monitoring the model training with Tensorboard.
- Experiment with different parameters for the layers.
- Experiment with different functions for the activation layers and the optimizers.
- Experiment with both “dropout” and “batch normalization”.

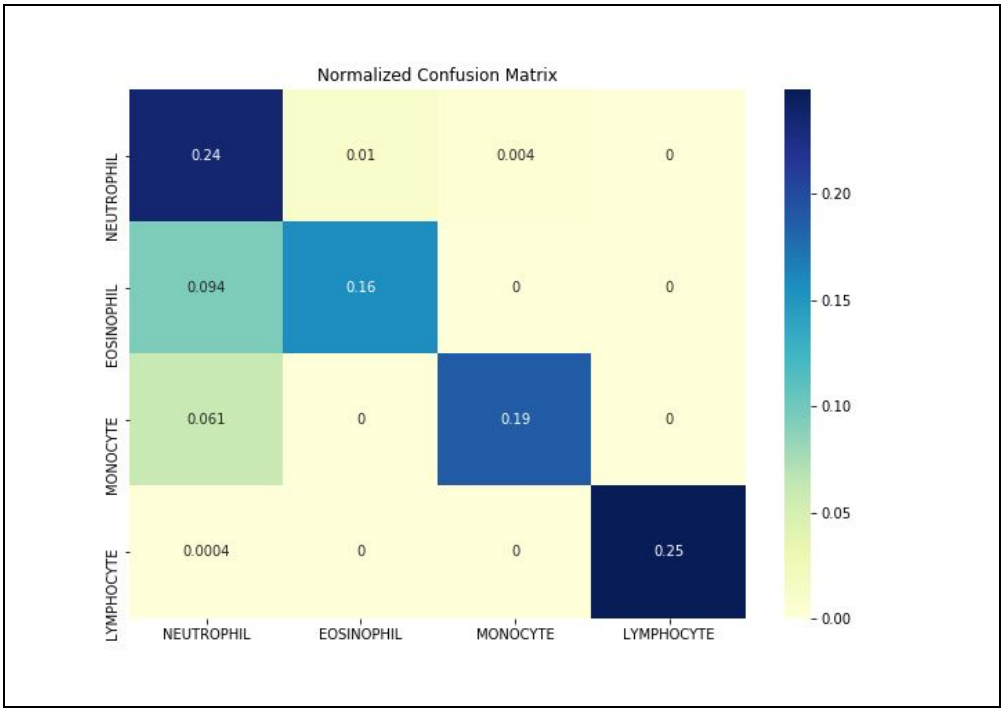
Results

After several experiments, the training accuracy reaches 91% and the validation accuracy reaches 83%. While increasing the number of epochs can potentially increase the accuracy of the training set, the problem of overfitting becomes a big concern as well. From the figures below, we can conclude that the model reaches an "ideal" states locally after around 25 epochs, although this number may vary depend on how well the weights are randomly initialized. After training the model many times, this model generally produce a good standard of accuracy after around 20-30 epochs.

In addition, I experienced the overfitting problem with "batch normalization" instead of simply "dropout" layer constantly in our case. While having "dropout" results into a graduate trend for the validation accuracy, having "batch normalization" results into greatly varied validation accuracy. Therefore, I only use "dropout" layer in the network.




Loss after 30 epochs
train_loss
val_loss
Loss
Epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
5
10
15
20
25


Normalized Confusion Matrix
NEUTROPHIL
EOSINOPHIL
MONOCYTE
LYMPHOCYTE
0.24
0.01
0.004
0
0.094
0.16
0
0
0.061
0
0.19
0
0.0004
0
0
0.25
NEUTROPHIL
EOSINOPHIL
MONOCYTE
LYMPHOCYTE
0.20
0.15
0.10
0.05
0.00

Conclusion

In our proposed work, I classify the white blood cells through the convolutional neural network via Tensorflow, and such a network shows promising result. While it takes around 10 minutes to train the data through Google Colab via the K80 GPU on the cloud, the training time can be further reduced using a local GPU such as GTX1080. With a small dataset and a short training period, the result is acceptable, however, it is expected to produce better performance with a even-distributed dataset with more distinct sample images.

Reference

- Deep Learning from Scratch + Insights (https://www.kaggle.com/placidpanda/deep-learning-from-scratch-insights)
- Blood Cell Subtype Identification cudnn+CNN+RNN (https://www.kaggle.com/coder98/blood-cell-subtype-identification-cudnn-cnn-rnn)
- Tensorboard usage (https://www.tensorflow.org/guide/summaries_and_tensorboard)
- Publicly shared dataset for the white blood cells (https://www.kaggle.com/paultimothymooney/blood-cells)