Malaria Infected Cells Classification via Multi-Layer Perceptron

Weining Hu
MS in Data science
George Washington University
whu369@gwu.edu

1 Introduction

This project built a Multi-Layer Perception ("MLP") model to classify the cells into 4 different categories, which are: "red blood cell", "ring", "schizont", and "trophozoite." The best model over the past seven-day data challenge competition has achieved a reasonable performance with a 0.6147 macro-averaged F1-score and 0.6212 Cohen's Kappa Score. The average score of the two is 0.6179.

Due to my personal mistake, I submitted a predict_whu369.py file with the wrong input dimension. Therefore, the last updated model has not been tested on the private held-out test set. It is unclear if the last updated model can achieve a higher performance score. The final script (train_whu369.py) that I submitted in the final deliverables is the script for writing the last updated model rather than the model that achieved the best performance score during the competition. The mlp_whu369.hdf5 file is the last updated model, which I perceive as the best model.

2 Dataset

The original dataset[1] ("the Train dataset") contains 8,607 cell images of size 100 X 103 and 8,607 txt files recording the corresponding string label for each cell image. Of the 8,607 cell images, 7,000 are red blood cell, 365 are ring, 133 are schizont, and 1,109 are trophozoite. 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 dataset.

2.1 Preprocessing

Image Resizing: The size of the images in the Train dataset varied from one another. Since MLP network requires a unified size for each input vector, a preprocessing step was implemented to resize all images into a shape of (50, 50, 3).

Coding technique: This step was achieved by using Pillow ("PIL"), the image processing package in Python.[2] Each image was resized into a square shape with padding while keeping its aspect ratio unchanged. I first set the maximum size of the input image equal to the desired size (50). Secondly, I pad the resized image to make it square.

Data Augmentation: As discussed, the Train dataset has a skewed distribution of cell types particularly over-representing red blood cells. Data augmentation was implemented to balance the training set and artificially added more training examples for the three under-represented classes. The augmentation was

[1] The dataset unzipped from the train.zip file will be referenced as the Train dataset hereafter.
[2] Image resizing was achieved by the function "get_input(images_paths)" in the final script *train_whu369.py*.

carried out by rotating and shifting. After augmentation, the final training dataset consists of 27,505 images (approximately 7000 images for each cell type).

Coding Technique: The augmented images were created using the ImageDataGenerator API in Keras.[3] The data generator created artificial images only for the three under-represented cell types, so that the final training dataset can have a balanced distribution over the four cell types. Since cell images can be rotated for any angle, the rotation range was set to 180. Additionally, considering the original cell image were mostly cropped in a way that preserved the cell body in the center of the canvas, the width shift range and the height shift range were deliberately limited to a very small step to 0.2.

Labeling: The original labels in the Train dataset were string values. A label encoding step were implemented to convert red blood cell to integer 0, ring to 1, schizont to 2, and trophozoite to 3.[4]

3 Modeling

3.1 Network Architecture

Model: The model used in this project was a three-layer MLP network.[5] The input dimension is 7,500 (50 X 50 X 3). The number of neurons for each layer is 128, 32, and 32, respectively. The activation functions used in the hidden layers are *Regular Linear Unit* (*"*Relu") function, while the output layer adopts the *Softmax* activation function, which is particularly useful for calculating the probability distribution of multi-class outputs. The optimizer is *Adam,* and the loss function is *Categorical Crossentropy*, a function that specifically targeted at categorical outputs.

Cross Validation: A 10-fold cross-validation was implemented to test the effectiveness of the model.[6]

3.2 Tuning of Network

Hyperparameters Grid Search: A grid search of learning rate, batch size, activation functions, weight initializer, and optimizer was built to firstly test on a smaller training dataset of 800 images (200 images for each cell type).[7] Once a more optional hyperparameter range was found, the optimal hyperparameters will be applied on the final train dataset.

Technical Challenge: The grid search did not work well on my cloud terminal. The function only works when I tested on a total of two hyperparameters with just two values for each hyperparameter. If the hyperparameter has a list of more than two values, i.e., batch_size = [20, 40, 60, 70, 80, 100], the program was able to run, but did not show the final mean test scores. Additionally, n_job = -1 did not work on the cloud terminal, which resulted in a longer waiting time when setting n_job = 1. Unfortunately, the current optimal hyperparameters used in the best model that I submitted were chosen manually.

[3] Data augmentation was achieved by the function "img_aug(rbc_num, img_arr, current_img_num)" in the final script *train_whu369.py*.
[4] This step was achieved by the function "label_encoding(t)" in the final script *train_whu369.py.*
[5] This step was achieved by the function "train_model3(x_train, x_test, y_train, y_test, original_x, original_int_y) in the final script *train_whu369.py.*
[6] This step was achieved by the function "cross_validation(inputs, targets, original_x, original_int_y)" in the final script *train_whu369.py.*
[7] The script is saved separately in the folder named *Day 5* as *train_whu369_d5_model_1.py*.

3.3 Results

During the course of modeling, an interesting phenomenon occurred due to an unexpected mistake made in my code scripts. My initial intension was to use image augmentation generator to create a batch of artificial images for each image from the three under-represented cell types in the Train dataset. However, I ended up creating 7,000 artificial images based on just one unique image originated from each one of the under-represented cell types due to an indexing mistake in the codes.

I subsequently built my first model (Model 1) based on a collection of 7,000 unique red blood cell images, and 20,505 artificial images that were transformed from just three distinct images from the other three cell types. The averaged macro F1-score was stagnant at the level around 0.6 between the 5th and 6th day of the competition, and the constraints of the imperfect augmented dataset made it extremely difficult to achieve any further significant breakthrough in the performance score. A screenshot of the indexing mistake in my code is displayed as below, followed by a modified version.

```
def sub_img_paths(targets, inputs):
    targets = list(targets)
    ring_imgs = []
    schizont_imgs = []
    trophozoite_imgs = []
    for t in targets:
        if t == 1:
            idx = targets.index(t)
            ring_imgs.append(inputs[idx])
        if t == 2:
            idx = targets.index(t)
            schizont_imgs.append(inputs[idx])
        if t == 3:
            idx = targets.index(t)
            trophozoite_imgs.append(inputs[idx])
    return np.array(ring_imgs), np.array(schizont_imgs), np.array(trophozoite_imgs)
```

Figure 1: Imperfect Data Extraction Method

```
def sub_img_paths(targets, inputs):
    targets = list(targets)
    ring_imgs = []
    schizont_imgs = []
    trophozoite_imgs = []

    for index in range(0, len(targets)):
        t = targets[index]
        if t == 1:
            ring_imgs.append(inputs[index])
        if t == 2:
            schizont_imgs.append(inputs[index])
        if t == 3:
            trophozoite_imgs.append(inputs[index])
    return np.array(ring_imgs), np.array(schizont_imgs), np.array(trophozoite_imgs)
```

Figure 2: Modified Data Extraction Method

Model 1: Although model 1 only used one distinct image originated from each one of the under-represented cell types, the model was still able to achieve a mean score of 0.61.

Model 2: After adding more varieties to the final training dataset by using the improved data extraction function, the model has achieved a significant progress compared with that of Model 1. This indicates that more data can improve the model's learning ability. Whereas less data (as shown in Model 1) can sometimes do a good job in generalizing the data feature.

4 Conclusion/Future Work

The coding mistakes from Model 1 inspired me to rethink about the number of images that we should use to train the model. For future work, it would be interesting to build a new model that just learn from a small number of images from each cell types. The results can reveal findings such as how much data is enough to feed a model and let it learn the pattern without generating overfitting.

Additionally, there are many other image augmentation techniques available in the open source. Due to the time constraint, I did not try the edge segmentation technique, which I think will be helpful to reduce the noises in the image and enhance the features of each cell types, so that the model can learn better by generalizing the characteristics of different cell types.