Malaria Cell Classification with Residual Neural Network

Xiangke Chen[1*†], Yixuan Chen[2*†], Jie Shen[3†], Yiran Wang[4†]

[1]Computer Science, Virginia Tech, Blacksburg, Virginia, 24061, United States

[2]Computer Science, University of Toronto, Toronto, ON, M4Y 1E8, Canada

[3]Computer and information Science, The Ohio State University, Columbus, OH, 43210, United States

[4]Applied mathematics, Beijing Forestry University, Beijing, Beijing, 100083, China

[*]Corresponding author's email: xiangkec19@vt.ed, yixuanch.chen@mail.utoronto.ca, shen.1389@osu.edu , yrjjlili@gmail.com

[†]The authors are equally-contributed.

Abstract

Malaria is a transmittable disease caused by the parasites which belong to the Plasmodium family which is spread by the bite of the female mosquito. In the past, they were detected by trained microscopists who analyze microscopic blood smear images. However in recent years, a series of deep learning methods had been appeared. In this passage, we use ResNet with different parameters to classify Malaria cell images. We preprocess a dataset with more than 27,000 Malaria cell images from more than 350 patients. On this dataset, we train a convolutional neural network which outputs the classification accuracy and loss. By using transfer learning, we find the best network architecture in which we can classify the cells with the accuracy of 94.93%.

Keywords: Residual Neural Network, cell classification, Malaria cell

1. Introduction

1.1. Background

Malaria was caused by invasion of parasites into humans' red blood cells. Just through a bite of infected mosquitoes, the plasmodium parasites could take the chance to invade your organs and red blood cells. If not treated within 24 hrs after the first symptoms, this disease would lead to severe illness or death. According to WHO, there are around 409000 deaths in 2019, and the experts suggested that "early diagnosis can reduce disease and prevent deaths", so lab tests and imaging of human red blood cells are especially important. Therefore, in our research, we would like to classify two types of cells (uninfected and parasitized) using five pre-trained CNN models.

Malaria parasites can be identified by examining a drop of the patient's blood under the microscope, spread out as a "blood smear" on a microscope slide. Thick and thin smears of Giemsa-stained peripheral blood samples are used for diagnosis. Thin smears are used to assess degree of infection and species of malaria parasite. The specimen is stained with Giemsa stain, which expresses the parasites in a distinct manner. Giemsa stain is used to differentiate nuclear and cytoplasmic morphology of platelets, red blood cells, white blood cells and parasites. Giemsa staining solution stains up nucleic acids, and therefore those components containing the nucleic acids, namely parasites, white blood cells and platelets, are highlighted in a bluish purple color. Red blood cells are usually colored in slight pink. The rings of the trophozoites take up pale blue color[1].

Above all, malaria cell classification is an essential task, which aims to classify the types of cells. Numerous works have been proposed to achieve an accurate classification performance.

1.2. Related work

In the research conducted by Prasad K[2], authors used 200 images, containing 100 images representing Plasmodium vivax infection, 50 images representing Plasmodium falciparum infection and 50 images representing joint manifestation of Plasmodium vivax and Plasmodium falciparum infection. They did the colour analysis to select the infected red blood cells as regions of interest. The work demonstrated in this paper helps develop a decision support system for efficient detection of infected cells. However, this system still needs remote support from an expert to help with the diagnosis.

In the study by Maude R, Jaeger S and Thoma G[3], they evaluated the performance of 5 pre-trained CNN networks and a customized neural network using 5-fold cross validation on the dataset of cell images collected from 150 infected and 50 healthy patients. The result is that ResNet50 outperformed the other models both on the patient level and cell level.

In the article[4], the author used 1182 color microscopic images of thick blood smear malaria slides, which contains 948 malaria-infected and 234 normal images. They improved YOLOV3 and YOLOV4 models by boosting feature scale and adding more detection layers to improve their capability of discovering small objects without particularly decreasing detection speed. Finally, they demonstrated the feasibility and effectiveness of proposed YOLOV-based architectures. However, the improved models are all slower than the original versions.

In the article[5], the author proposed a customized CNN model which exploits the bilateral filtering and augmentation techniques. With 27558 cell images which contain 13779 parasitized and 13779 uninfected, they use a bilateral filter to remove noise firstly. The result shows that the proposed method performs better than other deep learning models and it is also computationally efficient.

In the article[6], the author puts forward a faster region-convolutional neural network (R-CNN) to identify tumor cells with the aid of auto encoders to solve the problem that the tumor growth areas are normally obscure in ultrasound images and it is not feasible to prepare ultrasound images directly with CNN. 13,500 examples are implemented in this experiment as data sets. Experimental results show that the proposed R-CNN method using auto encoders performs better when compared to traditional data mining and Artificial Intelligence (AI) methods.

In the article[7], the author presents deconvolution as an alternate approach to local maxima detection. In this experiment, the author modifies a convolutional neural network (CNN) to convolve its output with the same mapping filter and trains it for the mapped labels. Then Output of the trained CNN is deconvolved to generate points as cell detection. The author uses 13484 (70%) examples from 99 sub-images for training and 5672 (30%) examples from 26 sub-images for testing. The results show that the proposed approach detects cells with comparatively high precision, compared with state-of-the-art deep learning approaches.

In the article[8], the authors applied cyclical learning rates(CLR) techniques on their CNN model. CLR will help prevent the model land to a low loss area while decreasing the learning rate. Meanwhile, CLR will boost up model convergence using less experiments and hyperparameter updates. The author used 22046(80%) images for the training set and 5512 images for the validation set. Meanwhile, the numbers of images for parasitized class and normal class are equal. By using the CLR-triangular schedule, the accuracy improved from base model 0.9646 to 0.9712.

In the article[9], the author used DeepSweep, a deep learning model, to identify loci under positive malaria selection. Specifically, DeepSweep is an already developed software, which uses AlexNet classifier architecture for its built in CNN. The author used 640 simulated datasets, 512 for training and 128 for validation. By article, changing the sweep type will affect the accuracy of the result. By using recent sweeps, the accuracy can reach up to 0.971.

1.3. Contributions

In this paper, we aim to find the best parameters and network architecture of the ResNet model by using transfer learning. In our first step, we change the optimizer to find the best one. Then in our second step, we compare different network architecture of the ResNet model in the same data set. To our surprise, the result told us there is no positive correlation between accuracy and depth of ResNet. By comparing the result of the before and after freezing convolution layer, we can naturally come to the conclusion that we had better not freeze it. In the end, we add hidden layers， dropout and ReLU. Finally we got the best accuracy 94.93% which is ResNet50 with an optimizer as Adam, learning rate as 0.001, hidden layer dropout and ReLU.

2. Method

2.1 Data collection

The source of our data is Quan2020_Artucle_AnEffectiveConvolutionalNeural. The data set is collected by Jaeger S from the National Library of Medicine. The dataset contains a total of 27,558 cell images with equal instances of parasitized cells from 151 patients and uninfected cells from 201 patients. In our experiment, we are going to split these data into three parts --- training (80%), validation (10%), and testing (10%) based on two different levels. The first one is the cell level, which means that all cell images from two categories will be split randomly. The second one is the patient level, which means that we will split all patients into three parts, and collect their cell images as our training, validation, and testing data sets. The main difference between these two methods is that the patient level data splitting can extract all cell images of one patient into one data set, while the cell level data cannot.

2.2 Data preprocessing

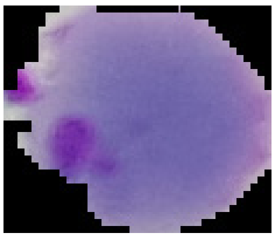
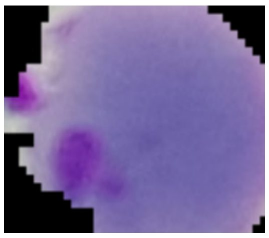
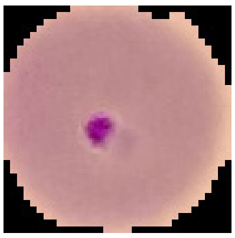
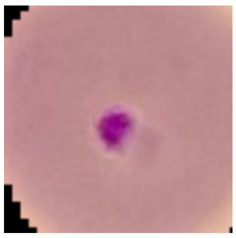
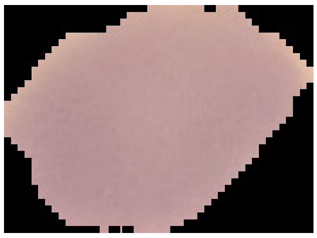
By using the normalize, crop and flip, we increase the diversity of our data set. In this way, our model will be able to recognize a diverse sample. In the following paragraph, we will go through the code and briefly explain the modifications that this part of code did. In this section of code, we use the basic graph operation class transforms.

We made a few modifications to the original code to make it work better for our experiment. For training set data, we went through the following processing on each image consecutively: transforms.RandomResizedCrop(), transforms.RandomHorizontalFlip(), transforms.Normalize().

2.2.1 Data preprocessing for training data: Crop and Flip

RandomResizedCrop() is a function that takes an integer as parameters, in our experiment. It will crop the image to our desired size with a random aspect ratio (Table 1-2). We set the final size to be 224 by 224 square. The next step for preprocessing the training data is to randomly flip the image horizontally by calling the function RandomHorizontalFlip() which takes a float as its only parameter. We want to have a 50% chance of flipping the image. Hence, we make the function take its default value for parameters which is 0.5.

Table 1. The examples of image crop.

Types	Raw Image	Processed Image
Parasitized		
Parasitized		
Uninfected		

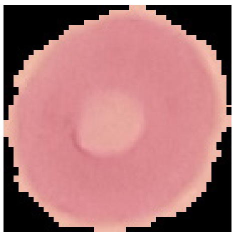
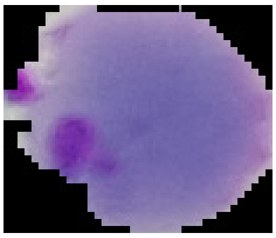
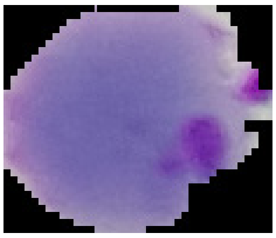
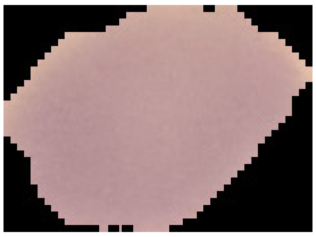
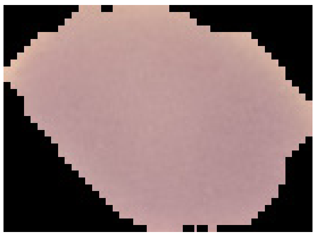
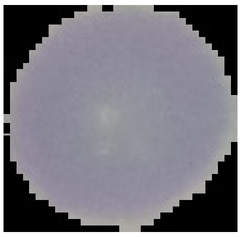
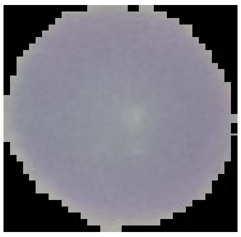
Uninfected		

Table 2. The examples of image flip

Types	Raw image	Processed image
Parasitized		
Parasitized		
Uninfected		
Uninfected		

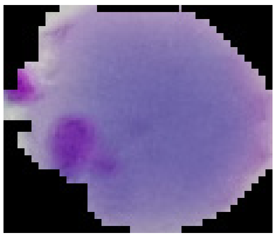
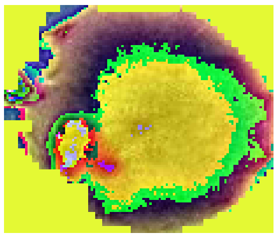
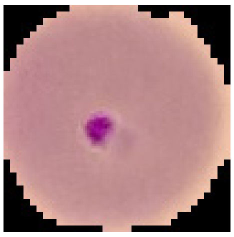
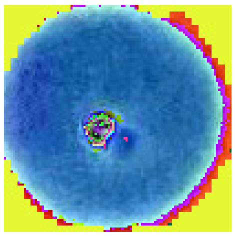
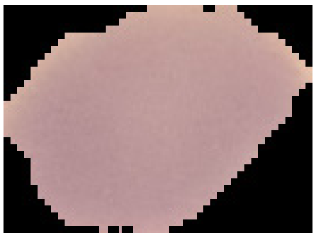
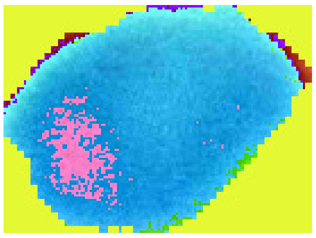
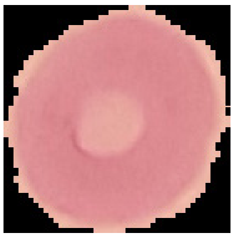
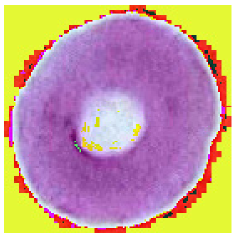
2.2.2 Data Preprocessing for training: Normalize

Through the above two steps, we initially diversified the original training set. The next process we need to do is data normalization by calling the function Normalize(). This function takes three parameters which are means, std and inplace. Means is the sequence of means for each channel. Std is the sequence of standard deviations for each channel. And the last boolean type inplace is optional, we go by default value false.

In the following part, we will briefly explain why and how we got such numbers. First of all, we need to know why we need to normalize the image. Different from the human eye, the way that computer looks at an image is based on the values of each pixel. We may recognize an infected cell by its shape and color, but computers may not. Although two images are showing the same thing, computers are not able to recognize them with different values from each pixel, which may be caused by different exposure, contrast, saturation, and so on. Normalize() function can help with this issue. By using the toTensor() function, we first transform a PIL image to tensor. Then call the Normalize() to map the image to the range we specified. Then all the training data will have a relatively similar shape in the aspect of values from each image (Table 3). In this case, our training will converge, and finally lead to a precise recognition.

The data we used in our experiment for the mean is [0.485, 0.456, 0.406] and for the std is [0.229, 0.224, 0.225]. In an image, we have all three channels, red, green and blue, ranging from 0 to 255, however, the mean here is in the range from 0 to 1. The reason behind is that we call the function toTensor() which will convert a PIL image in range [0, 255] to a torch.FloatTensor of shape in range [0, 1]. As for number 0.485, 0.456 and 0.406, they are calculated from random examples from imagenet training set data, which is provided on the official pytorch website[10].

Table 3. The examples of image normalization

Types	Raw image	Processed image
Parasitized		
Parasitized		
Uninfected		
Uninfected		

2.2.3 Data Preprocessing for testing: CenterCrop and Normalize

The functions we used for modifying our test set data are transforms.Resize(), transforms.CenterCrop(), transforms.Normalize(). To make the testing data more diverse, we applied the CenterCrop() function here. Compared to RandomResizedCrop() function, CenterCrop() crop image with the desired size passed by an integer parameter. The Normalize() used here takes the same parameters as we did for the training set. The purpose that we applied Normalize here is to make the image recognizable for the model we trained.

2.3 Algorithms

As for the degradation problem, resnet introduces a deep residual learning framework (Figure 1). Then let each few stacked layers fit a residual mapping. By using feedforward neural networks with “shortcut connections”, we change the original mapping from F(x) into F(x)+x . In our experiments, the shortcut connections simply perform identity mapping which means informulation, and their outputs are added to which is the outputs of the stacked layers. We adopt residual learning to every few stacked layers. All of their structure can be seen below. In our paper, we use the structure shown in the right subplot.

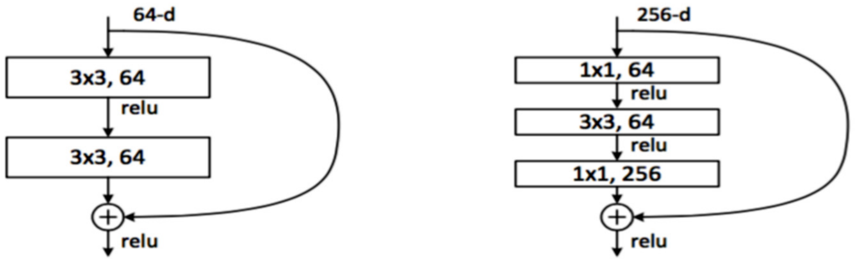
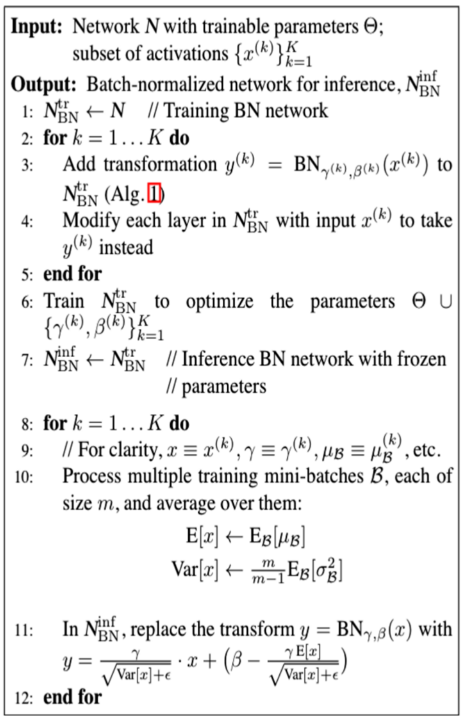


Figure 1. A deeper residual function FF for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.

With the residual structure, our nets can easily promote accuracy from greatly increased depth.

Algorithm 1. The overall workflow of batch normalization.

Algorithm 1. The algorithm of batch normalization



```
Input: Network $N$ with trainable parameters $\Theta$;
       subset of activations $\{x^{(k)}\}_{k=1}^{K}$
Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$
 1: $N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$   // Training BN network
 2: for $k = 1 \dots K$ do
 3:     Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to
        $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
 4:     Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take
        $y^{(k)}$ instead
 5: end for
 6: Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup$
    $\{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
 7: $N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$   // Inference BN network with frozen
                        // parameters
 8: for $k = 1 \dots K$ do
 9:     // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10:     Process multiple training mini-batches $\mathcal{B}$, each of
        size $m$, and average over them:
            $\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$
            $\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^{2}]$
11:     In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with
        $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
12: end for
```

We also use Batch Normalization as a way to increase accuracy. Batch Normalization is the method of normalizing feature maps for each layer, that makes them follow the distribution with mean of 0 and variance of 1. This makes the input values become more sensitive in terms of the nonlinear function. The process of batch normalization is shown in Algorithm 1.

3. Experiment

As planned originally, we first used the pre-trained model imported from pytorch. The weights of the pre-trained model were trained on ImageNet[11]. Since ImageNet is a very large dataset, the weights saved from the pre-trained model are usually trained on the ILSVRC2012, which contains about 1000 classes and over 12 millions of training images. It is expected that the pre-trained model has a strong robustness because it has learnt many pictures. Initially, we compared the performance of two neural networks trained with two optimizers, Adam and Adagrad. Both neural networks were trained using ResNet50 network structure with batch size of 32, learning rate of 0.001 and number of epochs of 50.

In order to further improve the validation accuracy, we compared the performance of three ResNet networks (ResNet50, ResNet101 and Resnet152) using Adam and Adagrad optimizers. In the third experiment, we chose the neural network ResNet50 to test the performance of freezed convolutional layers and unfreezed convolutional layers. Under this framework, we trained the dataset with two optimizers. In the fourth experiment, we changed the types of optimizers, the structure of fully connected layers under the condition of network structure, ResNet50.

4. Results

We analyze optimizer, network architecture, learnable layers, learning rate and depth of network respectively. We demonstrated five different tables (Table 4-7) to show their min loss and max accuracy.

Table 4. The result of optimizer comparison

compare optimizer with learning rate as 0.001		
optimizer	**adam**	**adagrad**
min loss(val)	**0.1710**	**0.1797**
max accuracy(val)	**94.48%**	**93.67%**

In this table (Table 4), we can find that using Adam as an optimizer is better than Adagrad in the model of resNet50. The two models in the table all show overfitting during the first ten epochs.

Table 5. The result of network architecture comparison

compare network architecture with learning rate as 0.001						
network architecture	**resNet50**		**resNet101**		**resNet152**	
optimizer	**adam**	**adagrad**	**adam**	**adagrad**	**adam**	**adagrad**
min loss(val)	**0.1710**	**0.1797**	**0.1675**	**0.1580**	**0.1582**	**0.1828**
max accuracy(val)	**94.48%**	**93.67%**	**94.16%**	**94.77%**	**94.00%**	**93.07%**

According to the table above (Table 5), Adam is significantly better than Adagard for resNet50 and resNet152. However, for resNet101, Adagrad has better effects than Adam. Hence, the hyperparameters combinations of learning rate=0.001, Network Architecture as resNet101,optimizer as Adagrad has the best effect at present. By the accuracy curve of this model, there may be over-fitting, and the accuracy of validation would drop around 28epoch, which may be related to the learning rate, as we suspected.

Table 6.The result of learnable layers comparison

compare learnable layers with learning rate as 0.001				
optimizer	**adam**		**adagrad**	
freeze all convolutional layers?	**no(1)**	**yes(3)**	**no(2)**	**yes(4)**
min loss(val)	**0.1710**	**0.2258**	**0.1797**	**0.2295**
max accuracy(val)	**94.48%**	**91.73%**	**93.67%**	**88.69%**

In this table (Table 6), we found that choosing Adam as the optimizer for resNet50 had a better overall effect than that of adagrad, and that the model with all convolutional layers freezed won't make the result as good as the unchanged model. However, by the image, the model with all convolutional layers freezed will be less likely to have over-fitting.

Table 7. The result of depth of network comparison

compare depth of network with learning rate as 0.001				
optimizer	**adam**		**adagrad**	
depth of network	**unchanged**	**add hidden layers, dropout and ReLU**	**unchanged**	**add hidden layers, dropout and ReLU**
min loss(val)	**0.1710**	**0.1409**	**0.1797**	**0.1747**
max accuracy(val)	**94.48%**	**94.93%**	**93.67%**	**93.55%**

According to the table (Table 7), for the resNet50 model, we can raise the accuracy and lower the loss by adding hidden layers and making dropout 40%.

5. Conclusion

In this paper, we apply ResNet for malaria cell classification. We first utilize image transformation to the images and sent them to the popular network, namely ResNet for image classification. Moreover, we also give an in-depth analysis on several aspects, including optimizer, network architecture, learnable layers, learning rate and depth of network. Our experiment shows that ResNet obtains satisifying performance on malaria cell classification. Our paper can provide a reasonable method and throughout analysis on how to apply deep neural network to medical image processing.

References

[1]Marada Amrutha Reddy , Ganti Sai Siva Rama Krishna , Teki Tanoj Kumar, 2021, Malaria Cell-Image Classification using InceptionV3 and SVM, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 10, Issue 08 (August 2021),

[2] Prasad K (2012) Image analysis approach for development of a decision support system for detection of malaria parasites in thin blood smear images. J Digit Imaging 25(4):542–549. https ://doi.org/10.1007/s1027 8-011-9442-6

[3] Rajaraman S, Antani S, Poostchi M, Silamut K, Hossain M, Maude R, Jaeger S, Thoma G (2018) Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images. PeerJ. https ://doi.org/10.7717/peerj .4568

[4] Abdurahman, F., Fante, K.A. & Aliy, M.(2021) Malaria parasite detection in thick blood smear microscopic images using modified YOLOV3 and YOLOV4 models. *BMC Bioinformatics* 22, 112 . https://doi.org/10.1186/s12859-021-04036-4

[5] Maqsood, Asma, Muhammad S. Farid, Muhammad H. Khan, and Marcin Grzegorzek.(2021). Deep Malaria Parasite Detection in Thin Blood Smear Microscopic Images. *Applied Sciences* 11, no. 5: 2284. https://doi.org/10.3390/app11052284

[6] Wajeed M.A, Sreenivasulu V (2019) Image based tumor cells identification using convolutional neural network and auto encoders. Traitement du Signal, Vol. 36, No. 5, pp. 445-453. https://doi.org/10.18280/ts.360510

[7] Shan E Ahmed R, Khalid A, Mariam J.H et al. (2019) Deconvolving Convolutional Neural Network for Cell Detection. 2019 IEEE 16th International Symposium on Biomedical Imaging, pp. 891-894, doi: 10.1109/ISBI.2019.8759333.

[8] Masud, M., Alhumyani, H., Alshamrani, S. S., Cheikhrouhou, O., Ibrahim, S., Muhammad, G., Hossain, M. S., & Shorfuzzaman, M. (2020). Leveraging Deep Learning Techniques for Malaria Parasite Detection Using Mobile Application. Wireless Communications and Mobile Computing, 2020, 1–15. https://doi.org/10.1155/2020/8895429

[9] Deelder, W., Benavente, E. D., Phelan, J., Manko, E., Campino, S., Palla, L., & Clark, T. G. (2021). Using deep learning to identify recent positive selection in malaria parasite sequence data. *Malaria Journal*, *20*(1). https://doi.org/10.1186/s12936-021-03788-x

[10] Torchvision.models. torchvision.models - Torchvision 0.10.0 documentation. (n.d.). Retrieved October 9, 2021, from https://pytorch.org/vision/stable/models.html.

[11]Abdulghany, Eman & Osama, Nada. (2021). Classification of Malaria Cell Images with Deep Learning Architectures. 10.13140/RG.2.2.26387.40484.