MIT Applied Data Science Program

Malaria Detection

Final Paper

Roula Krayem

12/13/2022

Executive Summary

Background

Malaria infection has been detrimental to human health outcomes, especially in African counties. Therefore, we decided to automate Malaria detection in human microscopic blood cell images. We investigated the data structure and properties to reach our goal and discussed using a convolutional neural network algorithm to configure the model. Later, we built, trained, and tested 14 models using different techniques. Comparing the 14 models, we decided that the first model was the best-performing model. Lastly, we provided suggestions for future improvements and policymakers.

Methods and Procedures

We followed multiple steps to determine the best-performing model. First, we built two models from scratch using different numbers of convolutional layers and activation functions. Second, we manipulated the images to increase the variation by implementing an image data generator. This step included training our second model eight times using one technique at a time, then all of them at once. Lastly, we used the transfer learning process by borrowing the convolutional layers from a pre-trained VGG16 model and building a fully connected layer on top of it. The lastly mentioned model was trained using different learning rates and batch sizes.

Conclusion

With the help of methodology and a multi-step process, we reached a model that correctly predicted 99% of our testing images. We can confidently say that clinicians can use our model to diagnose patients with Malaria. However, we strongly suggest reviewing the uninfected classified images by a specialist to avoid delaying the patient's treatment.

Design Thinking Process

Problem Definition

Out of the many microorganisms that could cause serious illness or even death, Plasmodium remains one of the most prevalent. Plasmodium is a parasite that can be transmitted through the bite of infected female Anopheles mosquitoes causing the human to have Malaria. The parasite grows and multiplies first in the human's liver, then in the blood's red cells—patients with Malaria present with symptoms such as fever, nausea, vomiting, and headaches. In severe cases, the loss of red blood cells can lead to kidney failure, coma, and death (1). In 2020, reports showed 241 million cases of Malaria with 627,000 deaths. 95% of the deaths occurred in African regions (2).

Since Plasmodium is known to attack human blood cells, clinicians use a Blood Smear test to diagnose patients with suspected Malaria. Under the microscope, the physician can locate the parasites and count their number. A Blood Smear test is the first diagnostic step. However, there are other useful tests, such as Polymerase Chain Reaction (PCR), Complete Blood Count (CBC), and Blood Glucose test (3).

This project intends to automate Malaria detection using images from the blood smear test. Therefore, we will build a machine-learning model. Then train and test the model using the microscopic images of the red blood cells. The outcome will be a classification solution of either parasitized (have Malaria) or uninfected (don't have Malaria). To build such a model, first, we must tackle the critical question: What machine learning method would produce an accurate and cost-effective model? To answer the question, we will explore the data (images). Then we will formulate the techniques to design the solution.

Data Exploration

To build an accurate model, we need to understand the dataset which the Great Learning Academy provided. The dataset was collected by The National Library of Medicine (NIH) and is publicly available (4). The zipped file is called cell_images, containing two subfolders, test and train. The test subfolder contains two additional subfolders:

1. Parasitized with 1300 Plasmodium-infected cell images
2. Uninfected with 1300 normal cell images, totaling 2600 images.

Also, the train folder contains two subfolders:

1. Parasitized with 12,582 Plasmodium-infected cell images
2. Uninfected with 12,376 normal cell images, totaling 24958 images.

We can infer from the number of the images above that the dataset is balanced as the training dataset is almost split in half between infected and uninfected cells. The same is true for the testing dataset. Additionally, the size of the images was different across the original folder. However, we resized all images to (64 x 64) when loading the dataset on Google Colab as shown in Figure 1.

```python
# We will run the same code for "parasitized" as well as "uninfected"
# folders within the "train" folder
for folder_name in ['/parasitized/', '/uninfected/']:

    # Path of the folder
    images_path = os.listdir(train_dir + folder_name)

    for i, image_name in enumerate(images_path):

        try:

            # Opening each image using the path of that image
            image = Image.open(train_dir + folder_name + image_name)

            # Resizing each image to (64, 64)
            image = image.resize((SIZE, SIZE))

            # Converting images to arrays and appending
            # that array to the empty list defined above
            train_images.append(np.array(image))

            # Creating labels for parasitized and uninfected images
            if folder_name == '/parasitized/':

                train_labels.append(1)

            else:

                train_labels.append(0)

        except Exception:

            pass

# Converting lists to arrays
train_images = np.array(train_images)

train_labels = np.array(train_labels)
```

```python
# We will run the same code for "parasitized" as well as "uninfected"
# folders within the "test" folder
for folder_name in ['/parasitized/', '/uninfected/']:

    # Path of the folder
    images_path = os.listdir(test_dir + folder_name)

    for i, image_name in enumerate(images_path):

        try:
            # Opening each image using the path of that image
            image = Image.open(test_dir + folder_name + image_name)

            # Resizing each image to (64, 64)
            image = image.resize((SIZE, SIZE))

            # Converting images to arrays and appending that
            # array to the empty list defined above
            test_images.append(np.array(image))

            # Creating labels for parasitized and uninfected images
            if folder_name == '/parasitized/':

                test_labels.append(1)

            else:

                test_labels.append(0)

        except Exception:

            pass

# Converting lists to arrays
test_images = np.array(test_images)

test_labels = np.array(test_labels)
```

Figure 1: python code for loading and resizing the images to (64 x 64)

Looking at the plotted red blood cell images (figure 2), we can see variation among them. Some cells are shown as full circular shaped, and others are quite distorted. Even the whole cells do not have a perfect circle shape and have different positions. Lastly, the color is inconsistent in all cells as they range from bluish to pink. However, the parasitized cells have a magenta globular structure. On the other hand, the uninfected cells do not have the same dark pink spots. This analogy is drawn from the small sample of images we see in the plot. Therefore, we cannot generalize the assumptions for all images.

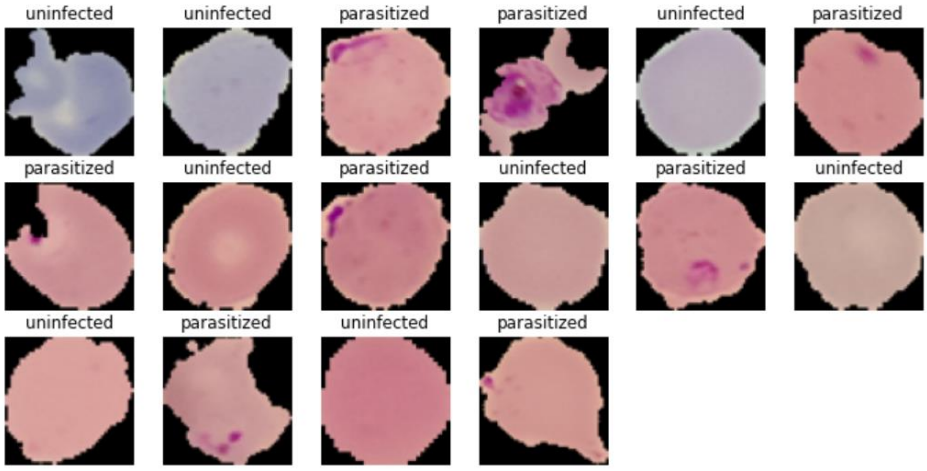


Figure 2: sample of the HSV converted testing dataset

To build an efficient model in detecting useful features, we can perform the following data processing:

1. Normalizing the data. When checking the pixel values of the training and testing dataset, we see that the images range from 0 (minimum) to 255 (maximum). To reduce the complexity of computing high numeric numbers, we can normalize the values by dividing them by 255.
2. One-Hot encoding the images. It is a process that quantifies categorical data.

3. Changing the images' color model from RGB (red-green-blue) to HSV (hue-saturation-value). This step makes detecting the area of interest easier because it separates the image luminance from the color information and creates significant color differences. After performing the conversion, we see in figure 3 and 4 that all the cells became pink, and what was the magenta globular structures became yellow. Now we can focus on locating the yellow elements and ignore the pink color variation.

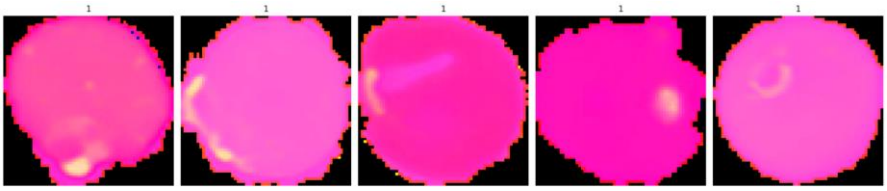


Figure 3: sample of the HSV converted training dataset

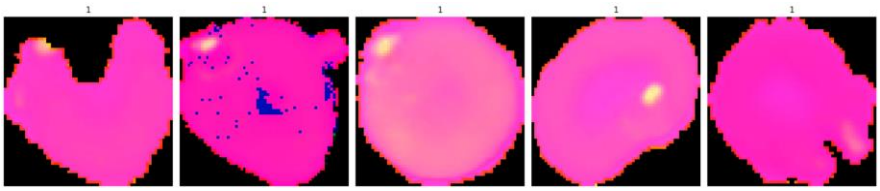


Figure 4: sample of the HSV converted testing dataset

4. Blurring/smoothing the dataset. Images have many features such as edges, brightness, contrast, color. Therefore, to reduce the noise and speckles within the images, we can perform a Gaussian Blur. We can see the blurred training and testing blurred images below (figure 5 and 6).

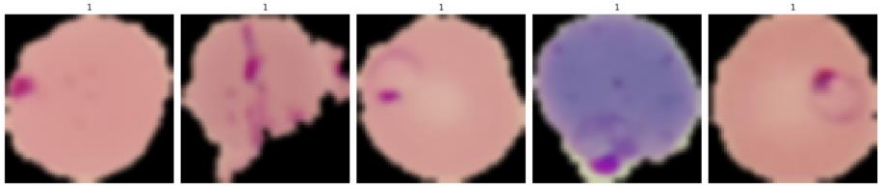


Figure 5: sample of the blurred training dataset

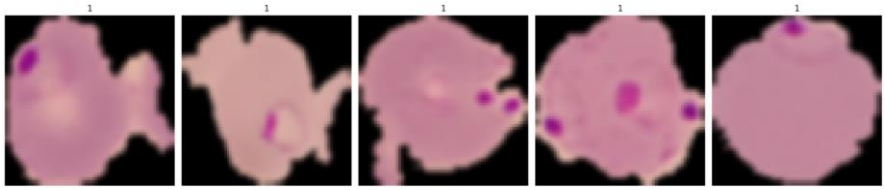


Figure 6: sample of the blurred testing dataset

Proposed Approach

A potential approach we can take to leverage the information available in the 2600+ red blood cell images is to use Convolutional Neural Networks (CNN). The deep learning techniques, CNN, 1) take in an input image, 2) assign importance by using learnable weights and biases to a different image aspect, and 3)

differentiate one aspect from the other. Therefore, CNN can be successfully applied in the feature extraction process and convert them into lower dimensions without losing characteristics (5). These capabilities can be obtained through TensorFlow and Keras. TensorFlow is free and open-source platform developed by Google that helps training large-scale machine learning models. Keras is an open-source software library that provides a Python interface for artificial neural networks.

CNN models include six essential layers (5). They are:

1. Input layer- contains image data represented by a three-dimensional matrix.
2. Convo layer- feature extractor layer.
3. Pooling layer- used between two convolution layers to reduce the spatial volume of the input image.
4. Fully connected layer- contains multiple layers of neurons that perform the weights and biases assessment and classification.
5. SoftMax/logistic layer- the last layer which produces the binary or multi classification.
6. Output layer- contains the labels

By implementing the six layers in the model structure, the CNN-based deep learning model should be capable of classifying uninfected and parasitized blood cells. Therefore, facilitate disease identification. However, different models might have multiple convo and pooling layers before the fully connected layer with a different number of parameters. Moreover, methods can be added to the model to avoid overfitting the data, such as simplifying the model, dropouts, and early stopping. For this reason, measures should be in place to test the performance of a model after fitting the data.

We are going to build multiple models, train, test, and evaluate the performance of each model using the following measures:

1. Precision- the proportion of positive identifications that was truly correct.
2. Recall- the proportion of actual positives that were identified correctly
3. F1 score- the weighted average of the precision and recall
4. Accuracy- the fraction of predictions our model got right
5. Train and Validation Loss- the difference between the expected outcome and the outcome
6. Execution time

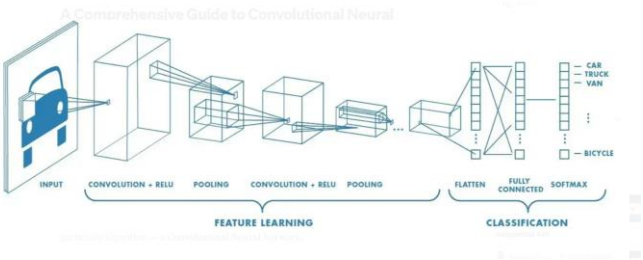


Figure 7: an illustration of the Convolutional Neural Networks components

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Comparison of Different Models and their performance

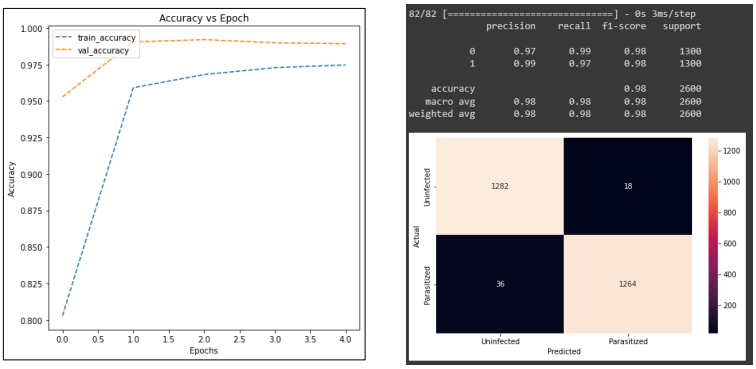
We built Many models and we compared them per the following:

Base model:

The base model has 1,058,786 total trainable parameters. It contains three convolutional layers, one dense layer, and an output layer. Each convolutional layer was built with 32 filters and used ReLU activation function. They also perform the max-pooling technique and drop 20 % of neurons before moving to the next set of the hidden layer. Lastly, the fully connected layer has one dense layer with a ReLU activation function, a dropout of 40% of the neurons. It also has an output layer with two neurons that uses the Softmax function.

When fitting the data, a callback function was used to store the best-performing epoch and end training. The function was set for a patience of 2, so the function would wait for two epochs before stopping the training. The fitting process used five epochs to train the model, 32 batch size, and we split 20% of the data for validation. The result yielded a higher validation accuracy than training accuracy (Figure 1-a). On the 5th epoch, the training accuracy was 0.974, the training loss was 0.076, the validation accuracy was 0.99, and the validation loss was 0.05.

We tested the trained model using the testing images. As we can see in Figure (1-b), the F1-score for both parasitized and uninfected categories is high (0.98) with an accuracy of 98%. Looking at the confusion matrix, we notice only 36 images of the parasitized cell were classified incorrectly (uninfected), and 18 images of the uninfected were classified as parasitized or infected.



(a) (b)

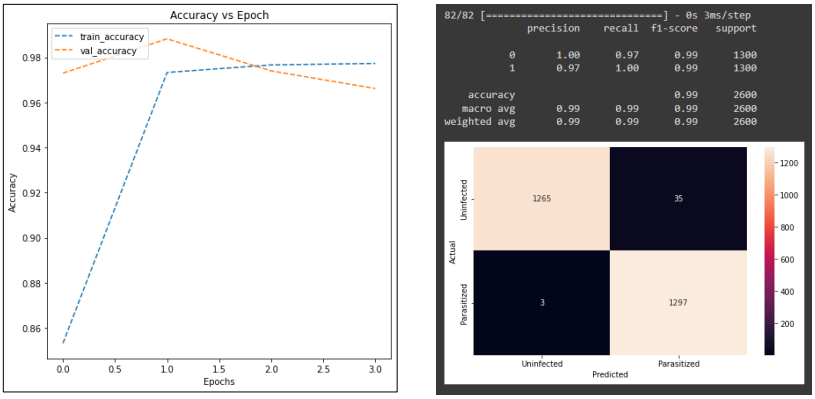
Figure 1: The Base Model's (a) Accuracy Vs Epoch chart and (b) Confustion Matrix

Model (1):

The first model had 84,002 total trainable parameters. Model (1) has five convolutional layers and the same fully connected layer compared to the base model, except that we changed the dropout rate from 0.4 to 0.2 before the output layer. Moreover, the convolutional layers have an alternating ReLU activation function, dropping out 20% of the neurons three times before the fully connected layer.

Compared to the base model, in this model we decreased the dropout rate from 40% to 20% in the fully connected layer before the output classification layer. The reason is that we wanted to improve training accuracy and loss. Therefore, we see in the accuracy/epoch chart (Figure 2-a) an increase in the testing accuracy compared to the validation accuracy. The model used 4 epochs to be trained. Looking at Figure (2-b), we notice an increase in the F1 scores. The increase in the parasitized F-1 score is linked to improving recall, which is clinically meaningful. We want our model to do better in diagnosing patients with Malaria

so they can start treatment sooner. And from the confusion matrix, we see a decrease in misclassified parasitized cells (3). On the other hand, the number of incorrectly misclassified uninfected cells increased to (35) Lastly, the accuracy also improved to 99%.



(a) (b)

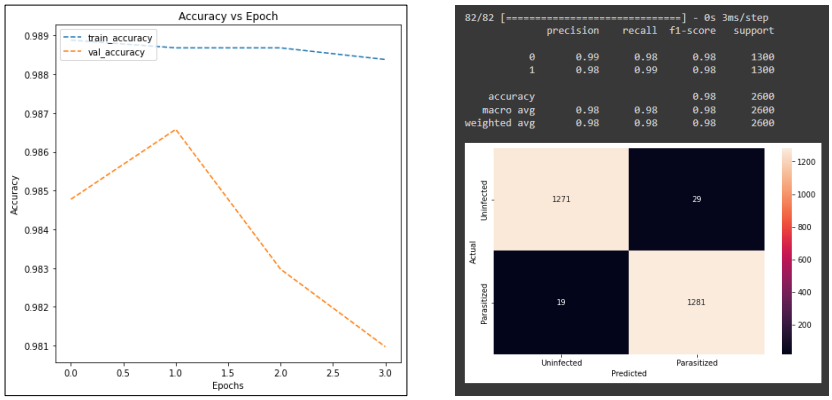
Figure 2: Model (1)'s (a) Accuracy Vs Epoch chart and (b) Confustion Matrix

Model (2)

The second model has 84,610 total trainable parameters. It has the same structure as the previous model, except that we removed the ReLU activation function and added a LeakyReLU after each convolutional layer. Moreover, we replaced the Dropout function before the fully connected layer with Bach-Normalization and kept the dropout rate before the output layer at 20%.

We used The callback function to avoid overfitting the model, and we split 20% of the data for validation. As a result, figure (3-a) shows a more stable trend for both the training and validation accuracy. On the 4th and final epoch, our training dataset's accuracy and loss were 0.988 and 0.03, respectively. And the accuracy and loss of our validation dataset were 0.981 and 0.08, respectively.

Testing the model resulted in a 0.98 F1 score for the parasitized and uninfected cells and a 0.98 model accuracy. We see a slight increase in the incorrectly identified parasitized cells (19) and a decrease in the incorrectly identified uninfected cells (29). All in all, the progress looks promising.



(a) (b)

Figure 3: Model (2)'s (a) Accuracy Vs Epoch chart and (b) Confustion Matrix

Data Augmentation Models

Trying to improve the model even further, we decided to use the image data generator provided by the *TensorFlow.Keras.preprocessing.image* library to perform the following on the training images only (no changes shall be performed on the testing dataset):

1. Horizontal Flip 2. Vertical Flip 3. Height Shift 4. Width Shift 5. Rotation
6. Shear 7. Zoom

This process is called data augmentation and is used to increase the differences in the training data; therefore, improved the model prediction accuracy. Moreover, it prevents data scarcity and reduced chances of overfitting.

We trained the second model using the images generated one technique at a time, then will all of them at once. Table 1 and 2 show that the results of the 8 models are very similar.

Data Image Process	Number of Epochs	Train Accuracy	Validation Accuracy	Train Loss	Validation Loss	Run Time (sec)
Zoom	6	98%	97%	0.065	0.081	142
Shear	7	98%	98%	0.073	0.073	162
Rotation	9	98%	98%	0.062	0.07	211
Width shift	8	98%	98%	0.068	0.077	191
Height Shift	11	98%	98%	0.06	0.065	266
Vertical Flip	6	98%	98%	0.067	0.071	33
Horizontal Flip	8	98%	98%	0.064	0.075	33
All	11	98%	98%	0.074	0.063	266

Table 1: results of training the second model using image data generator

Data Image Process	Precision Unin*	Precision Para*	Recall Unin*	Recall Para*	F1 Score Unin*	F1 Score Para*	Accuracy	Misclassified Para*	Misclassified Unin*
Zoom	0.97	0.99	0.99	0.97	0.98	0.98	98%	37	17
Shear	0.98	0.99	0.99	0.98	0.98	0.98	98%	30	15
Rotation	0.98	0.98	0.98	0.98	0.98	0.98	98%	22	21
Width Shift	0.97	0.99	0.99	0.97	0.98	0.98	98%	39	11
Height Shift	0.99	0.98	0.98	0.99	0.98	0.98	98%	14	31
Vertical Flip	0.97	0.99	0.99	0.97	0.98	0.98	98%	38	15

Horizontal Flip	0.98	0.99	0.99	0.98	0.99	0.98	98%	29	10
All	0.98	0.99	0.99	0.98	0.98	0.98	98%	28	15

Table 2: results of evaluating the second model trained by using image data generator

* Para= Parasitized and Unin = Uninfected

VGG16 Models

In our final attempt to improve the model, we used the VGG16 pre-trained model developed by the Department of Science and Engineering of Oxford University. We used the VGG16 model to replace the convolutional layer in the new model and changed its settings to non-learnable as it is already trained. Then on top of the model, we added a classification layer. Then we created the fully connected layer by adding a flattening function, three dense layers with ReLU functions (256, 128, and 64 neurons, respectively), a dropout function with a 30% rate, a Batch-Normalization function, and an output layer with two neurons and SoftMax function. We used a call-back function, an iteration of 0.001 and 0.00001 learning rate, and a 32, 65, and 100 batch size to fit the model. Table 3 and 4 show that the results of the 8 models are very similar, and the best performing model was the one trained using a learning rate of 0.00001 and a batch-size of 65. Nevertheless, the performance of all the VGG16 models do not compare to the previously built models.

Learning rate / batch size	Number of Epochs	Train Accuracy	Validation Accuracy	Train Loss	Validation Loss	Run Time
0.001/32	6	93%	93%	0.176	0.183	173 sec
0.00001/32	20	92%	93%	0.21	0.17	5 min
0.00001/65	20	92%	94%	0.211	0.17	10 min
0.00001/100	8	93%	94%	0.2	0.17	3 min

Table 1: results of training the second model using image data generator

Learning rate / batch size	Precision Unin*	Precision Para*	Recall Unin*	Recall Para*	F1 Score Unin*	F1 Score Para*	Accuracy	Misclassified Para*	Misclassified Unin*
0.001/32	0.98	0.84	0.82	0.98	0.89	0.91	90%	24	238
0.00001/23	0.96	0.89	0.89	0.96	0.92	0.93	92%	50	147
0.00001/65	0.95	0.91	0.91	0.95	0.93	0.93	93%	66	123
0.0000/100	0.96	0.89	0.88	0.96	0.92	0.93	92%	48	154

Table 2: results of evaluating the second model trained by using image data generator

*Para= Parasitized and Unin = Uninfected

Conclusion for the Final Design

With the results of our models that we tried to improve using the base model, we can conclude that the best-performing neural networks model is the first one we structured. Moreover, we will be proceeding with training and testing using the raw data rather than the augmented data because:

1. We have a large training dataset
2. We did not see overfitting signs before transforming the training dataset. (The F1 scores and accuracy reached 0.99 in the second model)
3. The structure of the images is relatively simple and does not have complex details.

Recommendations

For model future improvements, we suggest trying the following:

1. Try different learning rates and batch-size when fitting the second model using the data image generator.
2. Use the Ensemble technique with multiple weak models to improve the results.

For policymakers, we suggest adopting the first model to automate and facilitate Malaria detection. However, we strongly recommend reviewing the images classified as uninfected by a physician before making the final clinical decision. This is because we do not wish for a patient to go untreated or receive delayed treatment because of a misclassification error.

References

1. Centers for Disease Control and Prevention. (2022, March 22). *CDC - malaria - about malaria - disease*. Centers for Disease Control and Prevention. Retrieved November 29, 2022, from https://www.cdc.gov/malaria/about/disease.html
2. World Health Organization. (n.d.). *Fact sheet about malaria*. World Health Organization. Retrieved November 29, 2022, from https://www.who.int/news-room/fact-sheets/detail/malaria
3. *Diagnosis*. Stanford Health Care (SHC) - Stanford Medical Center. (2019, January 16). Retrieved November 29, 2022, from https://stanfordhealthcare.org/medical-conditions/primary-care/malaria/diagnosis.html
4. National Institutes of Health. (n.d.). *Malaria screener*. U.S. National Library of Medicine. Retrieved November 29, 2022, from https://lhncbc.nlm.nih.gov/LHC-research/LHC-projects/image-processing/malaria-screener.html
5. dshahid380. (2019, February 26). *Convolutional Neural Network*. Medium. Retrieved November 29, 2022, from https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529