Capstone Project

Machine Learning Engineer Nanodegreee

Paulo Henrique Zen Messerschmidt
May 6th, 2019

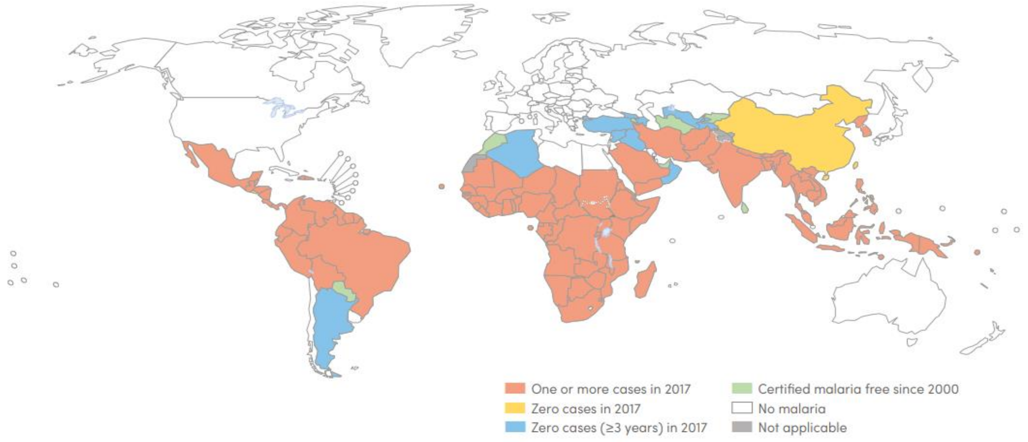
I. Definition

Project overview

Among the many diseases, malaria is a life-threatening disease caused by the parasite *Plasmodium*. Female Anopheles mosquitoes pick up the parasite from infected people when they bite to obtain blood needed to nurture their eggs. Inside the mosquito the parasites reproduce and develop. When the mosquito bites again, the parasites contained in the salivary gland are injected and pass into the blood of the person being bitten. (World Health Organization, 2019). According to World Malaria Report in 2017 there were 219 million cases of malaria compared to 217 million cases in 2016. The estimated number of malaria deaths was 435,000 in 2017, a similar number as in the previous year.

In addition, by 2017 the African region contained 92% of malaria cases and 93% of malaria deaths. In the same year, 5 countries accounted for nearly half of all malaria cases in the world: Nigeria (25%), Democratic Republic of Congo (11%), Mozambique (5%), India (4%) and Uganda %) (World Malaria Report, 2018). The Figure 1 shows the status of the disease in each country of the world in the year 2017. As can be seen, practically every African and Asian continent presented one or more cases in 2017 in addition to part of the Latin countries, such Brazil and Mexico. While the disease is uncommon in temperate climates, malaria is still common in tropical and subtropical countries.

Figure 1 - Malaria status by country in 2017.


One or more cases in 2017
Zero cases in 2017
Zero cases (≥3 years) in 2017
Certified malaria free since 2000
No malaria
Not applicable

World Malaria Report, 2018.

It can be said that, in short, malaria affects mostly underdeveloped countries, which in addition, generally have more precarious public health systems. Although the disease is curable, this precariousness of basic health care can often hamper diagnosis and even access to treatment.

Besides that Malaria is an acute febrile illness. In a non-immune individual, symptoms usually appear 10–15 days after the infective mosquito bite. The first symptoms (fever, headache and chills) may be mild and difficult to recognize as malaria because they are common to many types of diseases. If not treated within 24 hours, P. falciparum malaria can progress to severe illness, often leading to death. (World Healt Organization, 2019).

Problem statement

Based on this information, it is exposed the motivation for the development of this work, which is summarized in ***how to create a machine learning model to detect the presence of malaria based on cell images***. This model aims to aid in the early diagnosis of the presence of malaria, allowing the patient to have medical treatment in anticipation. In general terms, the purpose of this work is to contribute to save lives.

Metrics

Given the context of this problem and the adopted benchmark model, the following metrics will be considered:

- Accuracy:

$$Accuracy = \frac{TruePositives + TrueNegatives}{Total} \qquad (1)$$

- Recall:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \qquad (2)$$

In order to compare the number of false positives to the numbers of false negatives and get a more interpretable results for the probabilistic forecast returned by the final dense layer the ROC (**R**eceiver **O**peration **C**haracteristic) curve will be ploted. It is important notice that ROC curves are appropriate when the observations are balanced between each class (i.e., the dataset contains equal or almost equal number of samples from the positive and negative class), whereas precision-recall curves are appropriate for imbalanced datasets (Brownlee, 2018). The ROC cruves basically returns the **A**rea **U**nder the **C**urve (AOC), will be used as a metric, which is plotted in a Cartesian plane, where:

- The y-axis is represented by the sensitivity (equivalent to recall score):

$$Sensitivity = \frac{TruePositive}{TruePositive + FalseNegative} \qquad (3)$$

- The x-axis is represented by 1-specificity:

$$Specificity = \frac{TrueNegative}{TrueNegative + FalsePositive} \qquad (4)$$

The shape of the curves contains a lot of information such as the expected false positive rate and the false negative rate (Brownlee, 2018). Considering the problem context, the model should be able to reduce the false negative rate. It means that, in general, the model should not indicate that a patient is not infected, while in fact he is. This error type (type II) can be fatal because the patient will not receive the treatment.

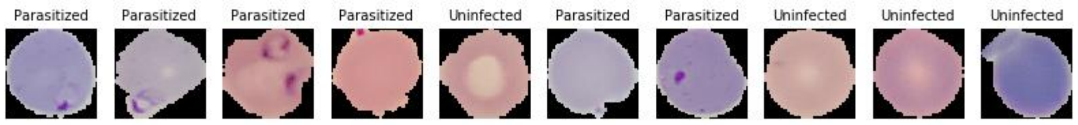
II. Analysis

Data Exploration and Visualization

The data used in this project was obtained from kaggle, specifically in the Malaria cell Images dataset.

This dataset is originally provided by the U.S. National Library of Medicine (NIH), and can be found in this link. This dataset has a total of 27,558 images where 13,778 images labeled as "Infected" and 13,778 images labeled as "Not Infected" (balanced dataset). Images are colored and have varying sizes, generally smaller than 300x300 pixels. The Figure 2 shows 10 sample images from the dataset with its respective labels. As can be seen, apparently the parasitized cells tend to show purple spots.

Figure 2 - - Example of an infected cell and an uninfected cell.



Each image has '.png' format and its respective filename as follow:

"C33P1thinF*IMG*20150619*114756a*cell_179.png"

where P1 denotes the patient ID (NIH).

Algorithms and techniques

In order to solve the proposed problem, a deep neural network model, more specifically Convolutional Neural Networks (CNNs), will be implemented, which is composed of convolutional layers and pooling layers. For the construction of the network, two approaches will be considered:

- **First approach:** create a CNNs from scratch, which will be composed of a certain amount of convolutional layers interspersed by pooling layers. The final layer will consist of a densely connected 2-node layer whose nodes will have a softmax activation function to return the probability of each label (infected and uninfected). In order to optimize the model the following parameters can be tunned:
    - Training parameters:
        - Number of epochs: associated with training length;
        - Batch size: how many images will be considered at once during a single training step;
        - Optmizers: what algorithms is used for learning;
        - Learning rate: how fast the model will learn;
        - Weight decay: it can be adjusted to prevent overfitting (prevents the model beign dominated by a few neurons)
    - Neural Network Architecture:
        - Number of layers
        - Layer types: Convolutional, fully-connected(dense) and Pooling
        - Layer parameters:

- Convolutional layers parameters
- Pooling layers parameters
- Dense layers parameters

Furthermore, some parameters in data-preprocessing can be tuned, for instance, rescale (divide each image pixel by 255) and change the image size. In Keras, ImageDataGenerator class can be used for this purpose.

- **Second approach:** create a model using the **transfer learning** technique. This technique consists of adapting neural networks already consolidated and trained in ImageNet database and adapt to the problem that we are trying to solve, in this case the prediction of the presence of malaria.

In a first moment, the algorithm used will be the VGG-16, but others algorithms already available in Keras can be tested in for performance comparison. The implementation steps based on the chosen approach will be:

- Cut the final dense layers of the network, keeping only the first layers, responsible for identifying common patterns, such as borders, common shapes such as stripes, circles, rectangles.
- Add to the pre-trained initial layers densely connected layers corresponding to the number of classes of the data set under analysis, in this case "Infected" and "Not infected".
- Freeze the weights of the pre-trained network and train only the part of the network added.

For the dense layers in both approaches the Dropout technique will be used. The idea behind the dropout technique is simple, every layer unit which dropout is applied is given the probability p of being temporarily ignored in calculations. In this case p is a hyper parameter called *dropout rate*. This technique prevents the neural network from overfitting.

Considering that the model can overfit according to the number of epochs, a check-point will be used to check the training progress and save the best weights of the model (set save*best*only = True). In order to implement this, keras provide a ModelCheckpointer class that can be instantiated and passed to the callback hyper parameter of .fit() method.

Finally, the performance of the network model created from scratch will be to compare to model using transfer learning. The transfer learning model is expected to perform better. The implementation steps are detailed in the ***Methodology section***

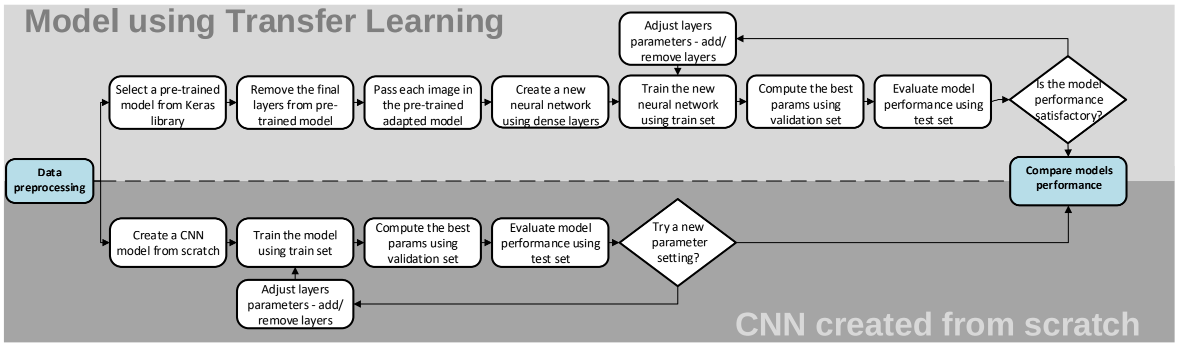
Benchmark model

The benchmark model used for comparison is presented in Ross et al (2006). In this model, the automated image processing algorithm is designed to diagnose malaria in the same way as a human operator performing microscopy. To do this, the algorithm finds and identifies erythrocytes and malaria parasites present in a microscopic field of blood (thin layer of blood). Based on the parasites and erythrocytes found, the program makes a diagnosis about whether or not malaria is present. Using this algorithm, the authors obtained an accuracy of 73% and recall score (sensitivity) of 85%.

III. Methodology

The project workflow is planned according to the flowchart shown in Figure 3. As can be seen, there are basically two sets of activities from the data preprocessing: the construction of the model using knowledge transfer; and building the CNN model from scratch. Considering that the models demand intense computational requirements, this process will be performed in Google Colaboratory environment.

Figure 3 - Workflow


Model using Transfer Learning
Data preprocessing
Select a pre-trained model from Keras library
Remove the final layers from pre-trained model
Pass each image in the pre-trained adapted model
Create a new neural network using dense layers
Adjust layers parameters - add/ remove layers
Train the new neural network using train set
Compute the best params using validation set
Evaluate model performance using test set
Is the model performance satisfactory?
Compare models performance
Create a CNN model from scratch
Train the model using train set
Compute the best params using validation set
Evaluate model performance using test set
Try a new parameter setting?
Adjust layers parameters - add/ remove layers
CNN created from scratch

Data preprocessing

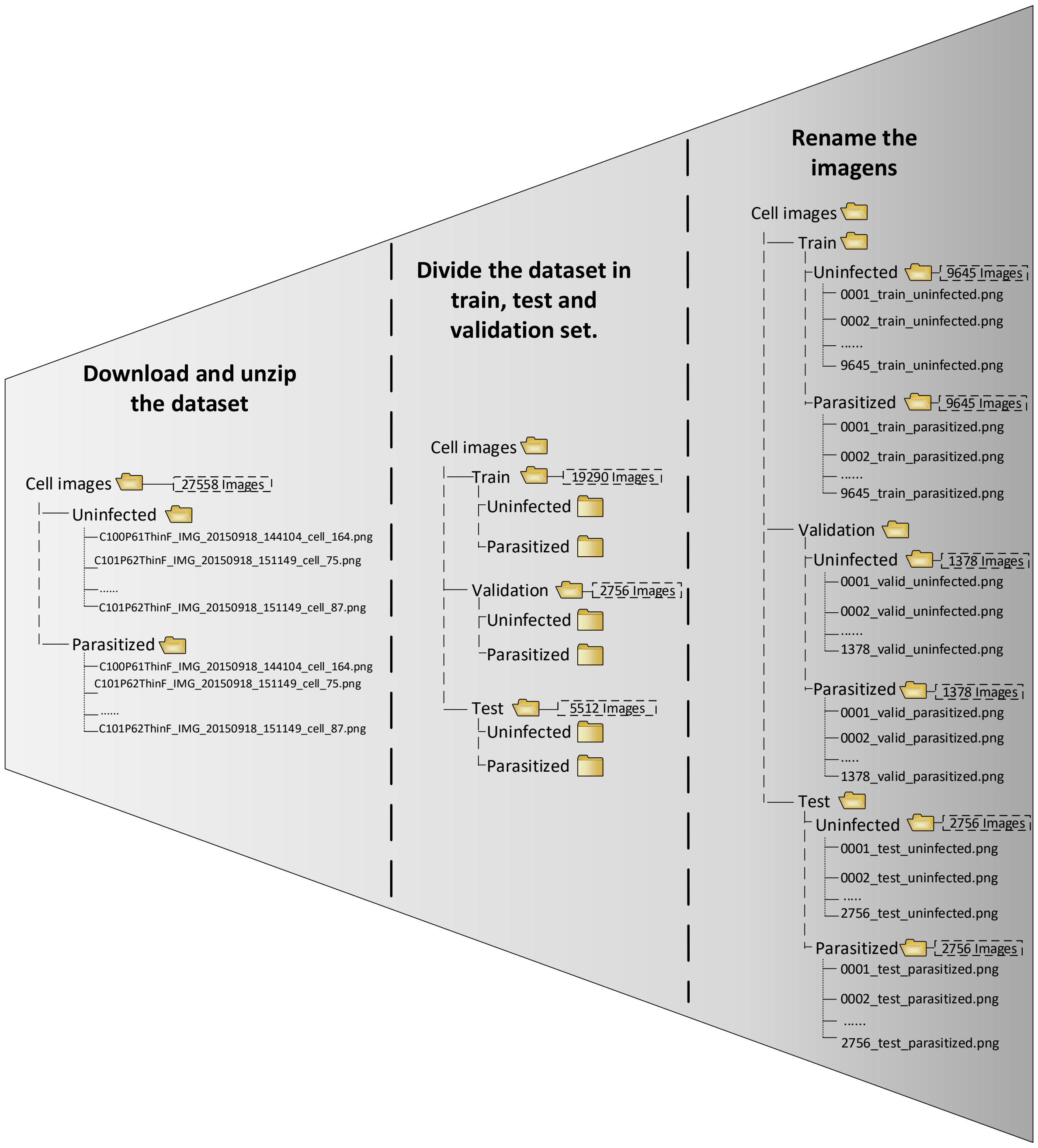
The data preprocess can be divided in two main steps: create the training, validation and test set; and transform the images in 4D vector to use them as input in the CNN.

Train, validation and test set

As can be seen in the Figure 4, after download and unzip the dataset, two main folders (one for each label) are created. Each folder contains the same number of samples for Uninfected and Parasitized label. The next step is create the training set, validation set and test set. For this, one folder for each set is created and the files are cut and pasted according to their respective labels. The number of samples for each set was chosen as follow:

- Training: 19,290 samples
- Validation: 2,756 samples
- Test: 5,512 samples

Figure 4 - Dataset splitting.


Download and unzip the dataset
Cell images
27558 Images
Uninfected
C100P61ThinF_IMG_20150918_144104_cell_164.png
C101P62ThinF_IMG_20150918_151149_cell_75.png
......
C101P62ThinF_IMG_20150918_151149_cell_87.png
Parasitized
C100P61ThinF_IMG_20150918_144104_cell_164.png
C101P62ThinF_IMG_20150918_151149_cell_75.png
......
C101P62ThinF_IMG_20150918_151149_cell_87.png
Divide the dataset in train, test and validation set.
Cell images
Train
19290 Images
Uninfected
Parasitized
Validation
2756 Images
Uninfected
Parasitized
Test
5512 Images
Uninfected
Parasitized
Rename the imagens
Cell images
Train
Uninfected
9645 Images
0001_train_uninfected.png
0002_train_uninfected.png
......
9645_train_uninfected.png
Parasitized
9645 Images
0001_train_parasitized.png
0002_train_parasitized.png
......
9645_train_parasitized.png
Validation
Uninfected
1378 Images
0001_valid_uninfected.png
0002_valid_uninfected.png
......
1378_valid_uninfected.png
Parasitized
1378 Images
0001_valid_parasitized.png
0002_valid_parasitized.png
.....
1378_valid_parasitized.png
Test
Uninfected
2756 Images
0001_test_uninfected.png
0002_test_uninfected.png
.....
2756_test_uninfected.png
Parasitized
2756 Images
0001_test_parasitized.png
0002_test_parasitized.png
......
2756_test_parasitized.png

Finally, each sample is renamed using the following pattern: `SampleNumber_set_label.png` (e.g., 0001*test*uninfected.png). Considering that the dataset contain 27558 samples, this process is time consuming if performed manually. In order to automatize this process a function called `rename_function` was created (this function is available in this link.)

This last step (renaming each file) is important to ensure the evaluation of the model consistently and is directly associated with the second main step (transform the images in 4D vector). The association between this tasks will be discussed next .

The second main step is to transform the images into 4D vectors. However, since this dataset is too large to load all the images into RAM at once, they should be loaded in batches. In addition, loading images in batches helps prevent overfitting and improves th ability of the model to generalize (Rosebrock, 2018). In keras, this process can be performed using ImageDataGenerator class. This class is generally used for data augmentation, but in this case the only transformation applied to the images is rescale the images dividing each pixel by 255 (1./255), as can be seen in the code below. An ImageDataGenerator object was created for each set.

```
from keras.preprocessing.image import ImageDataGenerator
```

```
datagen_train = ImageDataGenerator(rescale=1./255)
datagen_test = ImageDataGenerator(rescale=1./255)
datagen_valid = ImageDataGenerator(rescale=1./255)
```

To load images in batches directly from the directory `flow_from_directory()`method was used. As can be notice, all images was resized for 150x150 pixels. In addition, specifically for the `test_generator` (test_set):

- the `shuffle` parameter was set False. Since the test set is used for evaluate the model using the predict_generator and the images are loaded in batches, is necessary to maintain the order of the images to compare with the original labels.
- the `bach_size` was set 52. This is because its needed to considered that all images will be compared with their respective labels. This value can be changed according to the parameter `steps` in predict_generator. It is important to note that:

$$Steps = \frac{Number of samples}{Batchsize} \qquad (5)$$

It is important to choose a integer number for steps to ensure that all images will be predicted. In this case, `number_of_samples=5512`, `batch_size=52`, `steps=106`. The discussion about this topic is available in this link.

The last observation is that `flow_from_directory()` read the files in different order. Considering three files `f_1.jpg`, `f_2.jpg`, `f_10.jpg`, these files will be loaded in the following order: `f_1.jpg`, `f_10.jpg` `f_2.jpg`. Here is the importance of the rename step present above. In order to compare the predicted labels with the original labels is important to ensure that the `flow_from_directory()` method will loaded the image batches in the same order of the root folder. More information about this topic can be founded in this link.

```
#Define a batch_size parameter
batch_size=32

# Here .flow_from_directory is used to transform
train_generator = datagen_train.flow_from_directory(
    'content/cell_images/r_train', #Train folder path
    target_size=(150,150), #all images will be resized to 150x150
    batch_size=batch_size,
    class_mode='categorical') # We use categorical_crossentropy loss,
                              # we need categorical labels

test_generator = datagen_test.flow_from_directory(
    'content/cell_images/r_test', #Test folder path
    target_size=(150,150), #all images will be resized to 150x150
    batch_size=52,
    class_mode='categorical',
    shuffle=False)

valid_generator = datagen_valid.flow_from_directory(
    'content/cell_images/r_valid', #all images will be resized to 150x150
    target_size=(150,150),
    batch_size=26,
    class_mode='categorical')
```

Model implementation

After preprocess the data, for the first approach (CNN from scratch) the steps shown in Figure 5 was followed to create the model from scratch. Also, a ModelCheckpoint object was created to save the best parameters obtained during the training process. For the model training process the `fit_generator()`method was applied logging the validation/training loss and the validation accuracy. Finally, if the model performance was not satisfactory a changing parameter could be performed or new model architecture could be created.

For the transfer learning implementation (second approach) the steps shown in Figure 6 was performed. In this case, the VGG16 model was chosen. For this model, only the final dense layers was removed and the pre-trained weights of the initial layers was freeze. In this case, if the model performance was not satisfactory, the following approaches was considered:

- Initialize all the weights randomly and training all layers.
- Keep the weights of the initial layers freezing and adjust the only final dense layers added (change parameters or add/remove layers)

Refinement

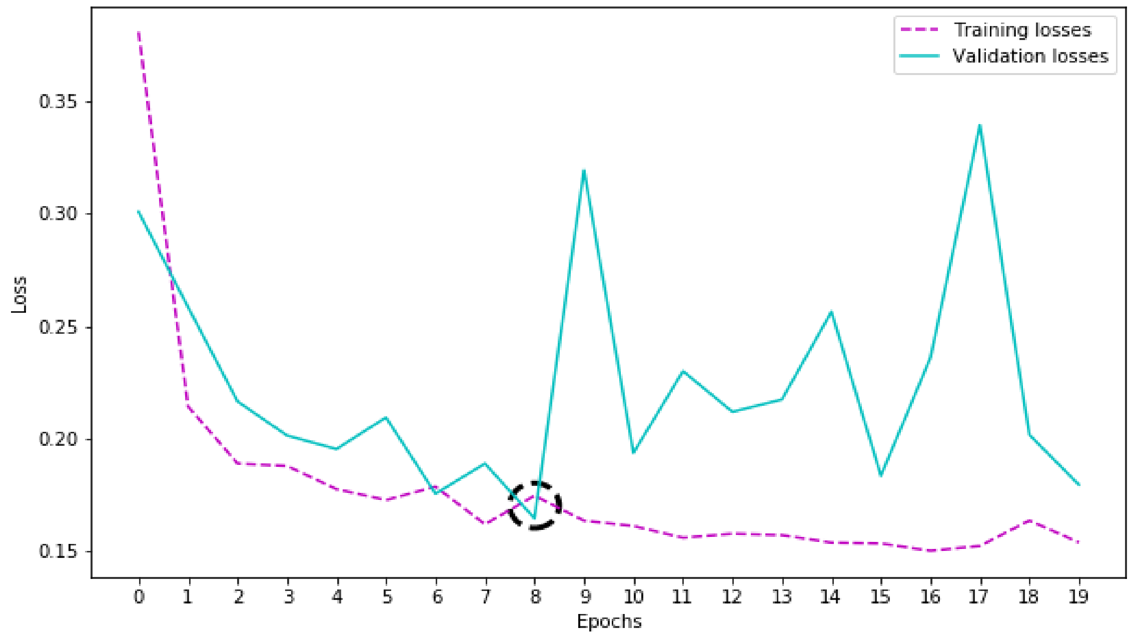
As mentioned, the benchmark model achieved an accuracy of 73%, recall score (sensitivity) of 85% and precision score of 81%. In spite of the accuracy score achieved by the model. Although the accuracy score has not reached a very high value, the recall score has reached a satisfactory value, since it has a lower false negative rate, which is desired in problems of this context.

The first CNN model created from scratch achieved an accuracy of 50%. In fact, this simple model was not predict the parasitized cells, since it predict all training samples as uninfected cells. In order to improve the model performance a new model was created; the result was an accuracy of 95,14%; recall score of 93%. The improvement was improved considering the following techniques:

- Improve the model complexity: based on the initial model, more two convolutional and Pooling layers was added. Considering that only one layer (convolutional and pooling layer) was not able to detect all the image patterns, the main idea behind the increase of model complexity was to turn the model able to identify more image patterns.
- Dropout layer: the dropout was added to each final dense layers (except the last one that predict the labels). As mentioned before, every layer unit which dropout is applied is given the probability p of being temporarily ignored in calculations. This technique was applied to prevent the neural network from overfitting.
- Model Checkpoint: a model checkpoint was implemented in the training process. As mentioned before, model checkpoint is used to output the model weights each time an improvement is observed during training. These outputs are saved in a file and loaded to the model to make predictions using the test set.

This second model was training in an iterative fashion in order to find the best parameters (e.g, *dropout rate*, activation functions in dense layers, number of neurons). This model has an accuracy of 95.13% and sensitivity of 93.00% on test set. As can be seen in Figure 5, the model reaches the lowest validation loss in the epoch 9. The checkpoint object saves the weights that results in this loss values and continues training the model. In this case there was not model improvement. The weights saved ate epoch 9 was loaded to the model to evaluate the performance on test set and the results mentioned above was obtained.

Figure 5 - Training and validation losses.



Finally, the model using transfer learning (tflearning_model.ipynb file)achieved an accuracy of 92.55% and sensitivity of 87.00%. This result was obtained removing the final dense layers and freezing all pre-trained weights of the initial layers. As can be noticed, there was not an improvement comparing to the second model performance.

IV. Results

Model Evaluation and Validation

Considering the metrics and models performance, the final model considered was the second model. The final architecture and hyperparameters was chosen based on the best performance among the tried combinations. The model architecture is described as follow:

- Three convolutional layers: the first layer learns 16 filters, the second 32 filters and the third learns 128. For the first layer, the input shape is set as 150x150x3.
- The shape of each filter is 2x2.
- The stride parameter in each convolutional layer is 1 (no difference between the resolution of input and output matrices)
- After each convolutional layer, a MaxPooling layer was added. The shape of each filter is 2x2.
- Three final dense layers (fully connected): the first final dense layer with 128 nodes (outputs), the second dense layer with 64 nodes. For both first two dense

layers the activation function is relu. Additionally, a dropout was added with dropout rate = 0.2.

- The final dense layers has 2 outputs ('Uninfected' or 'Parasitized'). This activation function of this layer is "softmax".

The implementation code of this model architecture is present below.

```
model = Sequential()
model.add(Conv2D(16, kernel_size=2, strides=1, padding='same',
                 activation='relu', input_shape=(150,150,3)))
model.add(MaxPooling2D(pool_size=2, strides=1))

model.add(Conv2D(32, kernel_size=2, strides=1, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2, strides=1))

model.add(Conv2D(128, kernel_size=2, strides=1, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2, strides=1))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

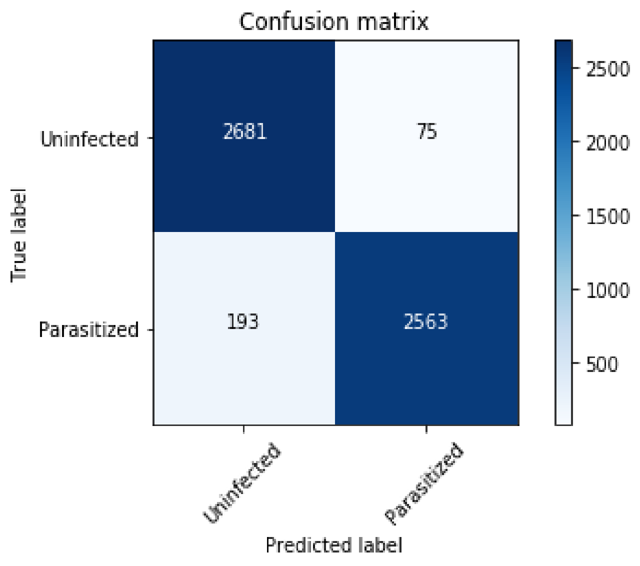
Justification

As mentioned before, the benchmark model has an accuracy of 73% and recall score (sensitivity) of 85%. In this project, the following results were obtained:

- Accuracy score around 95%
- Sensitivity score around 93%

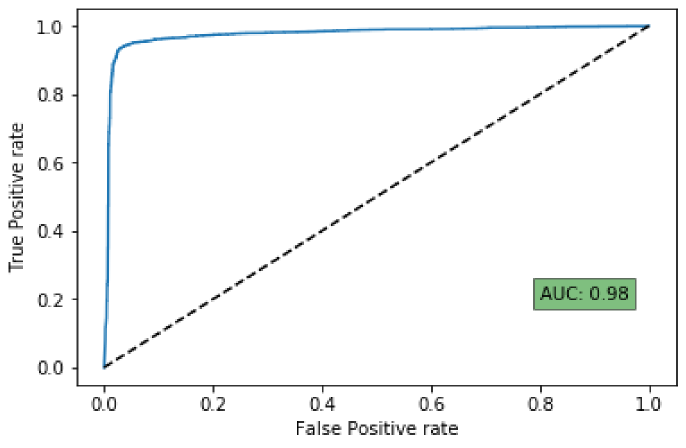
To better understand this scores, a confusion matrix was plotted (Figure 6). As can be seen, the model produces only 193 false negatives predictions and 75 false positives.

Figure 6 - Confusion matrix.



In this case, model precision is better than the model sensitivity. The ROC curve presented in Figure 7 shows the trade-off between True positive rate (sensitivity) and the False positive rate (1-specificity). As can be seen, the area under curve (AUC) is around 0.98. Considering that in an ideal classification model has area under curve equals to 1 and the sensitivity is 100%. The AUC represents degree of separability, i.e., it tells how much model is capable of distinguishing between classes. It can be concluded that the presented model is satisfactory.

Figure 7 - ROC curve.



Despite this, the model can generate better results compared to the benchmark model. In summary, this model is useful to provide early malaria diagnosis in order to anticipate an effective malaria treatment.

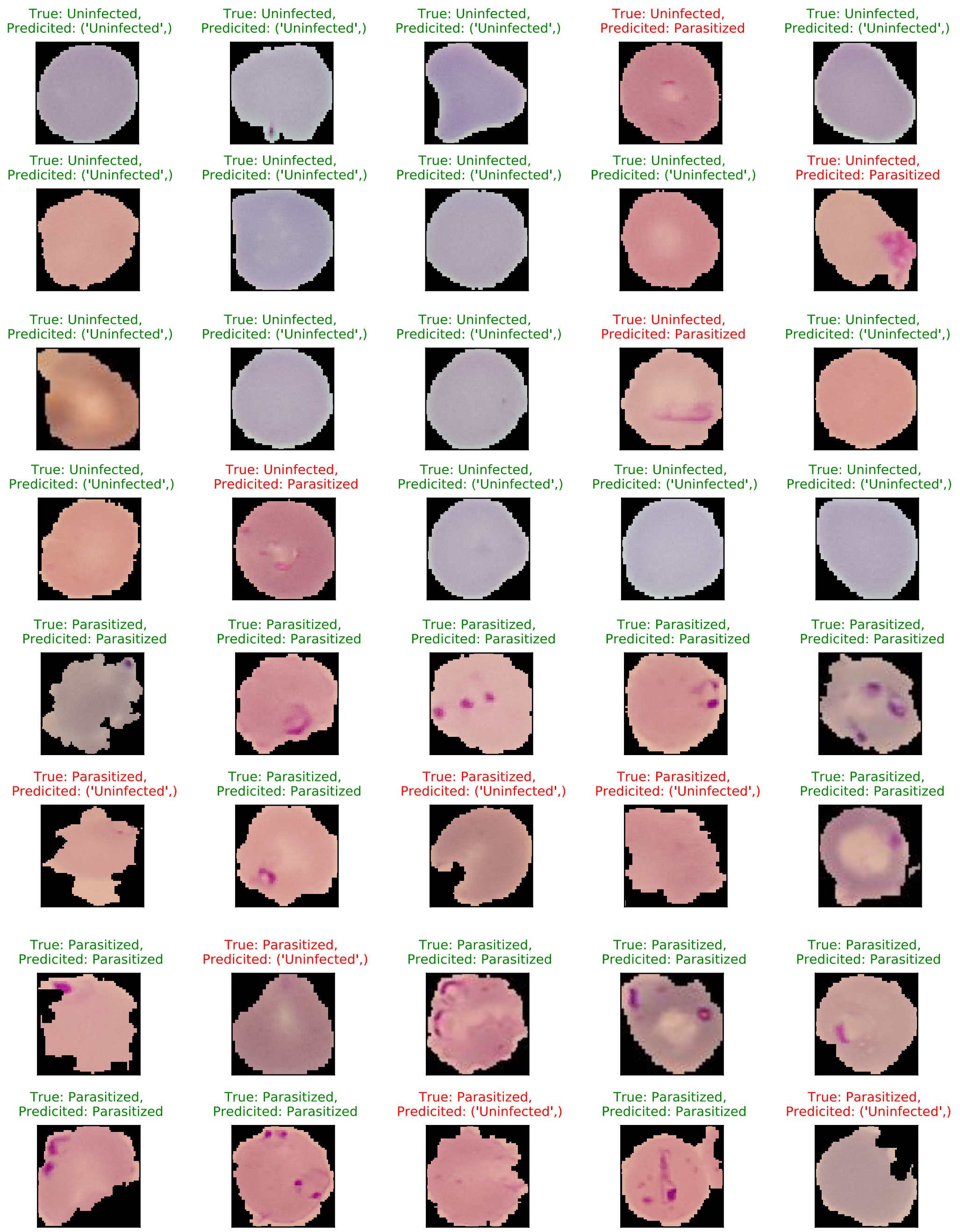
V. Conclusions

Free-form Visualization

The Figure 8 shows part of the test images with its true label and the predicted labels by the model. The green text means that the model predicted the correct label for the sample and the red text indicates that the model could not predict the correct label for the respective sample (Type I or Type II errors).

Figure 8 - Visualization of test set images with respective true and predicted labels.

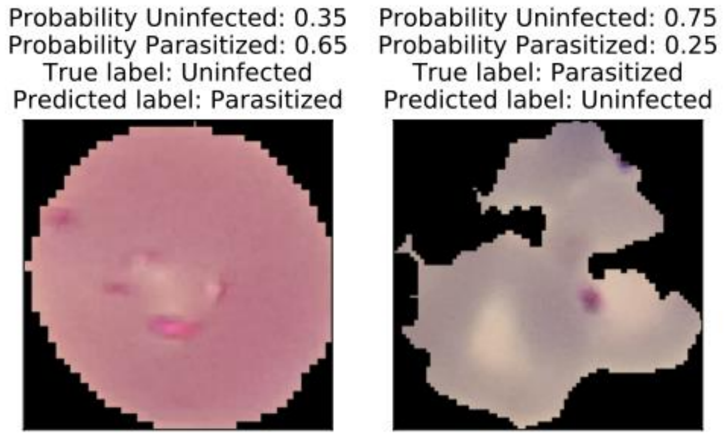
Figure 8 - Visualization of test set images with respective true and predicted labels.



True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: Parasitized
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: Parasitized
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: Parasitized
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: Parasitized
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: ('Uninfected',)
True: Uninfected, Predicited: ('Uninfected',)
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: ('Uninfected',)
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: ('Uninfected',)
True: Parasitized, Predicited: ('Uninfected',)
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: ('Uninfected',)
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: ('Uninfected',)
True: Parasitized, Predicited: Parasitized
True: Parasitized, Predicited: ('Uninfected',)

The Figure 9 shows two samples that was incorrectly predict by the model. In the right sample the model produces a false negative error (Type II) and in the left sample the model produces a false positive error (Type I). For this model, the threshold value is 0.5, i.e., the model will label the sample with the class that presents a probability equal to or greater than 0.5. In summary, the threshold value can be adjusted to reduce the false negative rates.

Figure 9 - Two samples with their respective class probabilities.


Probability Uninfected: 0.35
Probability Parasitized: 0.65
True label: Uninfected
Predicted label: Parasitized
Probability Uninfected: 0.75
Probability Parasitized: 0.25
True label: Parasitized
Predicted label: Uninfected

Reflection

The process followed for this project can be summarized in the following steps:

1. Problem definition: in this step problem was found based on public datasets. Next, the problem statement was establish.
2. Data download and split: in this step the dataset was downloaded. The dataset was segmented in training, validation and test set. After split the dataset, all files were renamed following a specific pattern.
3. Benchmark model: a benchmark model was defined for the proposed model.
4. Data preprocessing: the dataset was loaded and preprocessed in batches.
5. Model architecture: an initial model was defined and trained using the data. In order to increase the model performance, the model was updated with more layers. The updated model was trained multiple times, until the best parameters were found.
6. Pre-trained model: a model using transfer learning was implemented in order to improve the performance. In this case, VGG16 was used as pre-trained model.
7. Results comparison: the performance of the two models (from scratch model and pre-trained model) was compared in order to select the best model for the classification task.

I found steps 2 and 4 the most difficult, as i had to understand how to use the ImageDataGenerator class to preprocessing the images in batches, considering that the dataset could no be loaded directly in RAM. In addition, the main difficult founded was understand how the flow*from*directory loads the data batches and how it affects the data order, since to maintain the order of the data, the parameters must be adjusted correctly and data must be named in a specific way.

In general, i'm very glad that i found this dataset shared by NU.S. National Library of Medicine (NIH). I'm sure this dataset is very useful for those who are interested to learn about image recognition and also for those interested in machine learning for social good. I'm also happy about getting use of CNNs for image classification, as i learned about how neural networks works and how implement them using keras.

Improvement

The initial hypothesis that the pre-trained model would perform better when compared to the model created from scratch was not confirmed. Although both models performed satisfactorily (over 90% accuracy), the created model from scratch had a superior performance with a sensitivity of 93%. However, to further improvements of the model performance - in this case mainly reduce the false negative rate - it is suggested:

- Optimization of model architecture and deeper search for optimal parameters of the “from scratch” model;
- It is recommended to experiment with different combinations of final dense layers and different combinations of hyperparameters using the pre-trained VGG16 model. In addition, testing the behavior of the model by removing different amounts of convolutional layers (reduced model size);
- Implement other pre-trained models available in Keras Applications (e.g., VGG19 and ResNet);
- Compare the model performance adjusting the threshold value for classification.