Malaria Diagnosis using Deep Learning

Hassan Mahmoud Abdelkhaleq Sarwat

July – 2019

1-Definition

1.1 Project Overview

Malaria is a life threatening disease that is caused by a parasite; it is transmitted to people through the bite of infect female Anopheles mosquitoes. In 2017 there were estimated 200 million cases of malaria across 87 countries. According to World Health Organization the estimated number of malaria deaths stood at 435 million in 2017.

Medical professionals commonly examine thick and thin blood smear and compute parasitemia, however this is not always accurate, it depends on the skill of the medical professional available, and the other problem is that malaria is very common in rural areas therefore there may not always be a medical professional available.

This is where deep learning comes in, having a computer recognize the disease will make it easier, faster, and cheaper to diagnose the disease and we will be able to prevent more deaths.

1.2 Problem Statement

The goal of the project is to teach a model to identify cells that have been parasitized, since this an image classification task the first technique that comes to mind is convolutional neural networks, however training a convolutional neural network from scratch is not always the best solution, which is why I opted to use transfer learning instead.

There were 3 models I wanted to test this on, VGG19 [1], InceptionV3 [2], and InceptionResnetV2 [3] as they are fairly new and have not been tested in the benchmark model.

1.3 Metrics

The metrics used to evaluate the project were accuracy, which is how well the model performs in general.

(#Correctly identified parasitized cells + #Correctly identified uninfected cells)/#Cells

Other 2 metrics were F1 score and recall, however for us the most important one is the recall, which is number of correctly classified parasitized cells/number of parasitized cells.

A high recall means it has a good detection rate, as it's better to misdiagnose an uninfected cell than to misdiagnose a parasitized cell because that way the patient will not be unaware of the problem.

2- Analysis

2.1 Data Exploration

I used the dataset acquired the National Institutes of Health, it contains 27,558 images of cells. 13,779 of those are parasitized cells and 13,779 are uninfected cells.

Since both classes have the same number of images, and they both contain more than 10,000 images I did not think it was necessary to perform data augmentation on this dataset

2.2 Exploratory Visualization

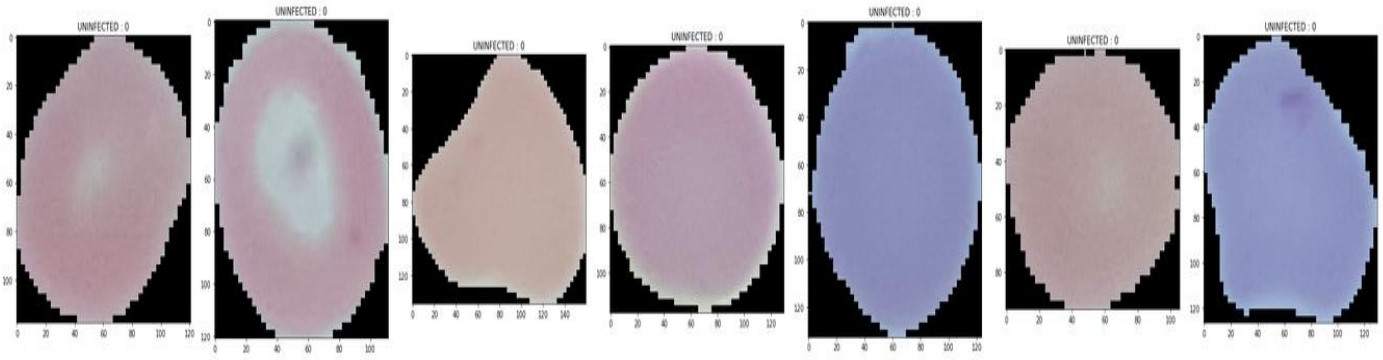


Figure 1 *Images of Uninfected Cells*

If you have a look at the above figure, you'll observe that cells come in many shapes however we do notice that they do not contain an edge and some cells have white spots inside of them. We also observe that they appear in many colors, mostly pink/skin color while some are taking more of a blue hue.

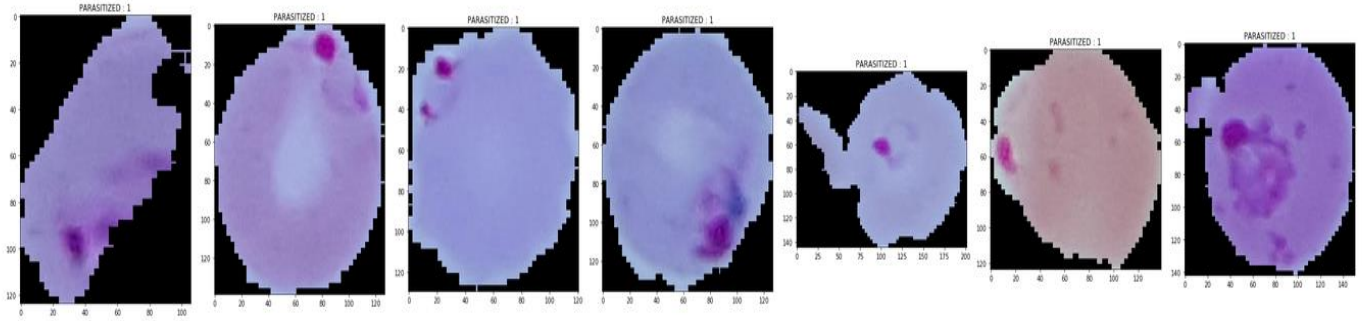


Figure 2 Images of Parasitized Cells

If you have a look at the above figure, you'll see that parasitized cells have a dark spot/area inside of them, we can also notice that the dark spots are not always consistent in location, shape or size. In some cases it around the borders of the cell and in other it is around the center, sometimes it just a spot while others appears more as a smudge like in the 7th cell in **Figure 2**. We also observe that parasitized cells contain a purplish hue, though that is not always the case as you can observe in the 6th cell in **Figure 2**.

2.2 Algorithms

Since this is an image classification task, a convolutional neural network will probably bring about the best result. A convolutional neural network is a form of neural network that cares more about the spatial and temporal dependencies, which can be crucial when identifying images or objects in images.

In this project 3 pre-trained convolutional neural networks were used, pre-trained meaning that the models have already been trained but not on our dataset, in this case they were pre-trained on Imagenet dataset.

At the end of each network I added a global average pooling, it basically reduces a *h*w*d* tensor into a *1*1*d* tensor. It reduces each *h*w* feature map to a single number by taking the average of all *hw* values.

After the global average pooling layer [4] I added a fully connected layer, followed by a dropout layer and then the final layer which consists of 2 nodes (number of classes.)

The dropout layer works by dropping 50% of the values passed around the network, this is used to help reduce over fitting and preventing the network from becoming dependent on a few layers instead of all of them.

I used 2 different optimization functions, stochastic gradient descent and RMSprop. Stochastic gradient descent [5] works by randomly selecting a sample to optimize the gradient, and RMSprop works by dividing the gradient by a running average of its recent magnitude.

2.3 Benchmark

The benchmark model is based on the published article **"***Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images***"** which was released in 2018. They also used pre-trained models as feature extractors and trained their network.

They reported the below accuracies for each model:

Models	Accuracy
AlexNet	0.937 ± 0.012
VGG-16	0.945 ± 0.015
ResNet-50	**0.957 ± 0.007**
Xception	0.890 ± 0.107
DenseNet-121	0.931 ± 0.018
Customized	0.940 ± 0.010

3- Methodology

3.1 Data Preprocessing

There wasn't much preprocessing used in this dataset, as mentioned in section 2.1 the dataset contained 27,558 images, there was no class imbalance so there wasn't really a need for data augmentation. I read the images in rgb format with size 224*224 because that was the default for the VGG19, however I believe implementation should work just fine as long as size is not smaller than 100*100 and is in rgb format.

```
for pic in tqdm(parasite):
    try:
        image=cv2.imread("../input/cell_images/cell_images/Parasitized/"+pic)
        image_from_array = Image.fromarray(image, 'RGB')
        size_image = image_from_array.resize((224, 224))
        data.append(np.array(size_image))
        labels.append((1,0))
    except AttributeError:
        print("")
```

+ Code + Markdown

```
for pic in tqdm(uninfected):
    try:
        image=cv2.imread("../input/cell_images/cell_images/Uninfected/"+pic)
        image_from_array = Image.fromarray(image, 'RGB')
        size_image = image_from_array.resize((224, 224))
        data.append(np.array(size_image))
        labels.append((0,1))
    except AttributeError:
        print("")
```

3.2 Implementation

After reading the images, first thing I did was divide the dataset into train and test data with a 80-20 split.

There were 3 models tried in this project, the first model was the VGG19, after extensive tries I found out that the best layer to extract from would be '*block5_conv3*' and a learning rate of 0.0001, this resulted in 95% accuracy. Other layers I tried extracting from would be '*block5_conv2*' and the final output layer, I also played around with the decay and tried adding another fully connected layer and a dropout layer but that gave me an accuracy of 91%.

I added a validation split in all training so that I can identify over fitting when it happens.

```
vgg_19 = VGG19(weights='imagenet',include_top=False,input_shape=(224,224,3))
vgg_19 = Model(input=vgg_19.input,output=vgg_19.get_layer('block5_conv3').output)
#vgg_19.summary()

x = vgg_19.output
x = GlobalAveragePooling2D()(x)
x= Dense(1024,activation='relu')(x)
x = Dropout(0.5)(x)
prediction = Dense(2, activation='softmax', name='predictions')(x)

from keras.optimizers import SGD

model = Model(inputs=vgg_19.input,outputs=prediction)

for layer in vgg_19.layers:
    layer.trainable=False

sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(optimizer=sgd,
              loss='mse',
              metrics=['accuracy'])

model.fit(X_train, y_train,
      batch_size=32,
      epochs=60,
      shuffle=True,
      validation_split=0.1,
      verbose=1)
```

The 2nd model used was inception v3, my first 2 tries in this model led to a 50% accuracy, the model was predicting most of the cells as uninfected. This led me to perform many changes, first was changing loss function from mean squared to categorical cross entropy, which didn't cause much improvement.

After a bit of research though I realized that this might be caused by freezing the layers from the inception model, as my dataset is very different from the imagenet dataset I decided to unfreeze the layers, I also changed the optimization function from SGD to RMSprop.

My biggest problem in inception v3 though was overfitting, the model would overfit under 10 epochs, but this problem was solved after decreasing learning rate to 0.000001. After many trials, I found out that the best layer is '*mixed7*', this one got me the highest reported accuracy of 96.5% but once again, inconsistent and not sure if it is repeatable.

```
x = INCV3.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024,activation='relu')(x)
x = Dropout(0.5)(x)
prediction = Dense(2, activation='softmax', name='prediction')(x)

from keras.optimizers import RMSprop

model = Model(inputs=INCV3.input,outputs=prediction)


rms = RMSprop(lr=0.000001)
model.compile(optimizer=rms,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

hist = model.fit(X_train, y_train,
      batch_size=64,
      epochs=15,
      shuffle=True,
      validation_split=0.1,
      verbose=1)
```

Final model used was InceptionResnetV2[incres], this one got the best result, consistently getting 96%, at the beginning I included the entire model, but just like inceptionv3 I noticed it too was overfitting, however after picking layer '*block17_9_ac*' it didn't overfit even after training for 30 epochs. Just like the inceptionv3, I unfreezed the inceptionresnet layer, allowing the model to overwrite the weights that were initialized to imagenet, I also used RMSprop and categorical crossentropy.

```
IncRes= InceptionResNetV2(weights='imagenet',include_top=False,input_shape=(224,224,3))
IncRes = Model(input=IncRes.input,output=IncRes.get_layer('block17_9_ac').output)

x = IncRes.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024,activation='relu')(x)
x = Dropout(0.5)(x)
prediction = Dense(2, activation='softmax', name='prediction')(x)

from keras.optimizers import SGD
from keras.optimizers import RMSprop

model = Model(inputs=IncRes.input,outputs=prediction)

# for layer in INCV3.layers:
#     layer.trainable=False
#sgd = SGD(lr=0.00001, decay=1e-6, momentum=0.9, nesterov=True)
rms = RMSprop(lr=0.000001)
model.compile(optimizer=rms,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

hist = model.fit(X_train, y_train,
      batch_size=64,
      epochs=30,
      shuffle=True,
      validation_split=0.1,
      verbose=1)
```

3.3 Refinement

After trying out the 3 models, I have come to learn that while VGG19 can be improved, it will not perform better than the inceptionresnet or inceptionv3.

First off, for the inceptionv3 and inceptionresnet I did not try out as many layers as I'd like, picking the layer to extract from was more of a guess than me fine tuning the process, I did try the final layer of inceptionresnet and inceptionv3, but the model was over fitting.

Nevertheless, I believe there are many methods I can try to improve the models. First off, I can train them for more epochs, another method would be to try out different network structure after the pre-trained model, I mostly used the 1024 nodes + drop out layer but maybe I can try adding another similar fully connected and dropout layers. A different solution would also be to try to better tune the hyperparameters, like batch size, loss function, optimization function, or learning rate using a method like GridSearch.

4- Results

4.1 Model Evaluation and Validation

Final model I'd pick would be either the inceptionresnet or the inceptionv3, the results for their top models were as follows

Model	Accuracy	F1_Score	Recall
InceptionV3	0.965529753265602	0.965290464011691	0.973112338858195
InceptionResNet	0.962445573294629	0.962246945103045	0.975322283609576

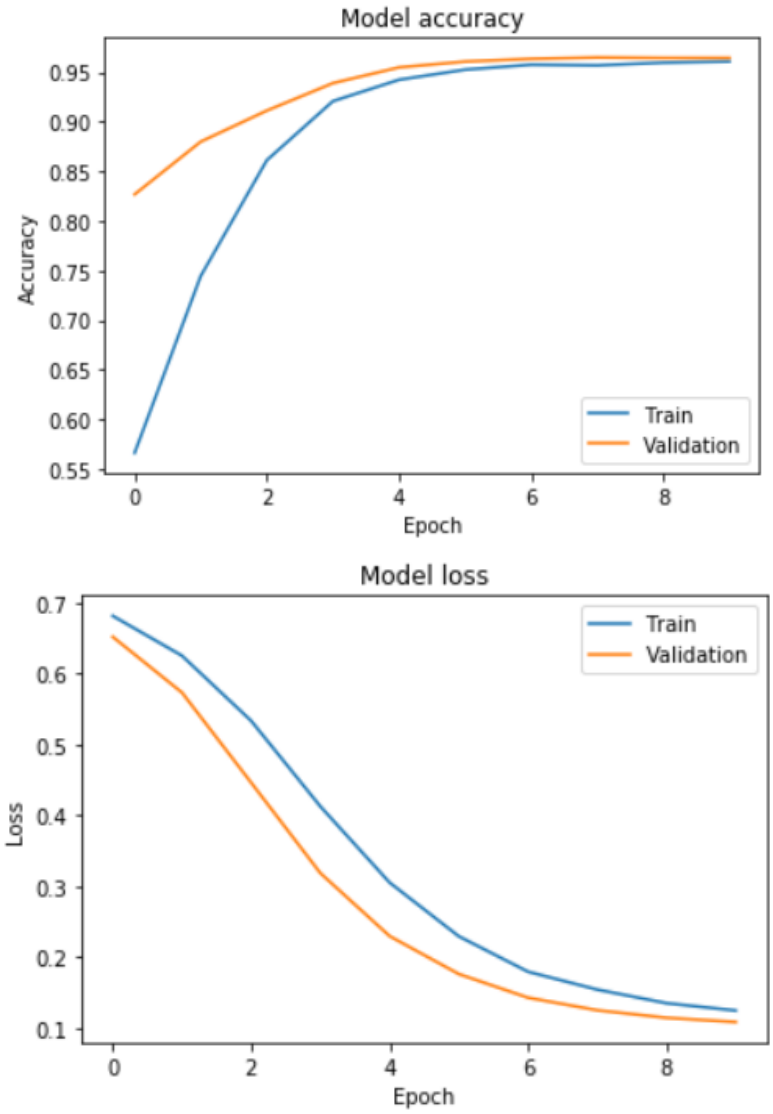
While the InceptionV3 had better overall accuracy, the inceptionresnet had better recall, and like previously mentioned it is better to misdiagnose an uninfected cell than to misdiagnose a parasitized cell. The final learning rate was 0.000001, for both models, I find this one was the best as it prevents over fitting.

4.2 Justification

The 2 models perform better than the benchmark resnet model, and they both can consistently reach 96% accuracy, however the improvement is not as much as I'd hoped and it can get better, however due to lack of computing power I was not able to try out different hyperparameters as I'd like.

5- Conclusion

5.1 Visualization



If you look at the figure to the right you can see how the training performed in 10 epochs for the InceptionV3 model. You can see that in the beginning it was improving very rapidly but at around 5 epochs improvement slowed down to a crawl.

You can also observe the same thing in the loss graph, with this we can tell that the training was very stable.

Figure 3 Training history of InceptionV3 Model

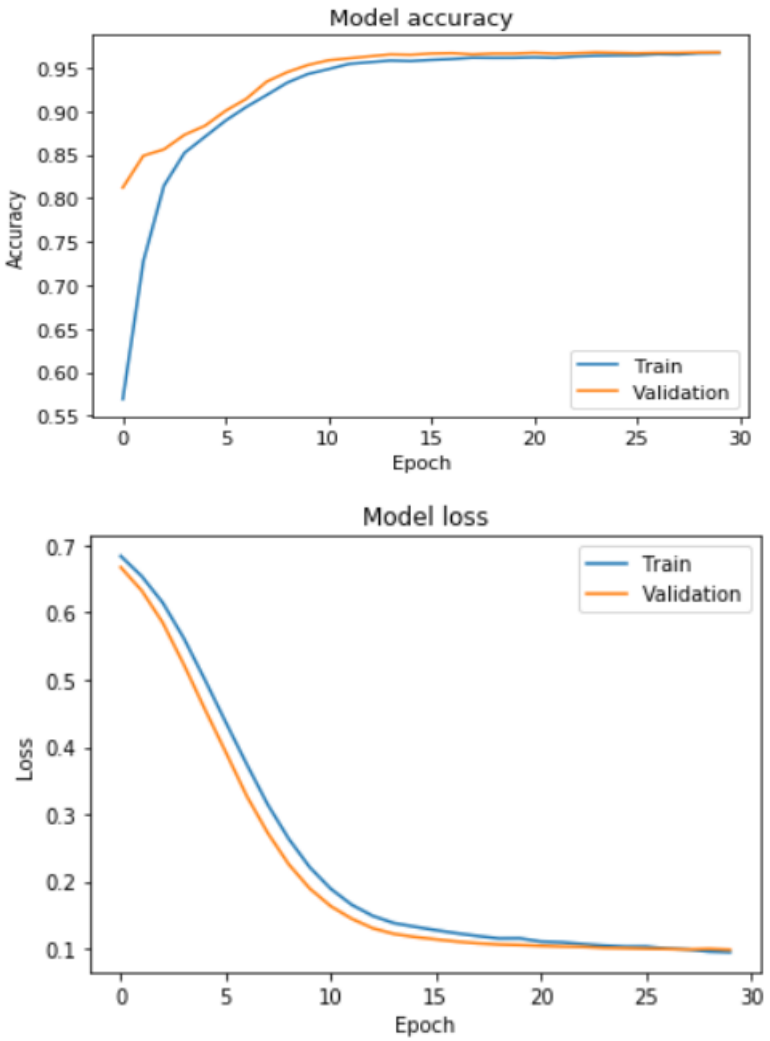


Figure 4 Training history for InceptionResNet

In these graphs, we observe the training for the inceptionresnet, just like the inceptionV3 the training was also stable, but it plateaued later on at the 10th epoch and has been undergoing minor improvements since.

If we look at the loss graph we see that at late 20s epoch the training loss started decreasing more than the validation, this could be a sign that our model will start to overfit after 30 epochs.

5.2 Reflection

Overall I liked this project, I found that overall this was a very educational project. At first I was unsure what to pick but I'm glad I found this dataset, first of all the data was simple to read and didn't really require any preprocessing.

After that there was fine tuning the model, which keras made incredibly simple, I started out in c++ where you have to initialize everything, so coding in python is a huge upgrade from where I started and I can't be thankful enough to whoever made keras. Fine tuning the hyperparameters was interesting, having a validation

split and watching the model overfit was heart breaking though, and taking a long time to train a model can be sort of bothersome when you're pressed for time.

After reading images and training model came the visualization, again, python made incredibly easy using matplotlib, overall the project was fun and educational to work on and I believe I learned a lot.

5.3 Improvement

If I had to say if there's anything that can be improved it would be adding more images to the dataset, even though 27,558 is a lot for images it is not always enough.

References

1- Very Deep Convolutional Networks for Large-Scale Image Recognition
2- Rethinking the Inception Architecture for Computer Vision
3- Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning
4- Network In Network
5- An overview of gradient descent optimization algorithms