Malaria Detection

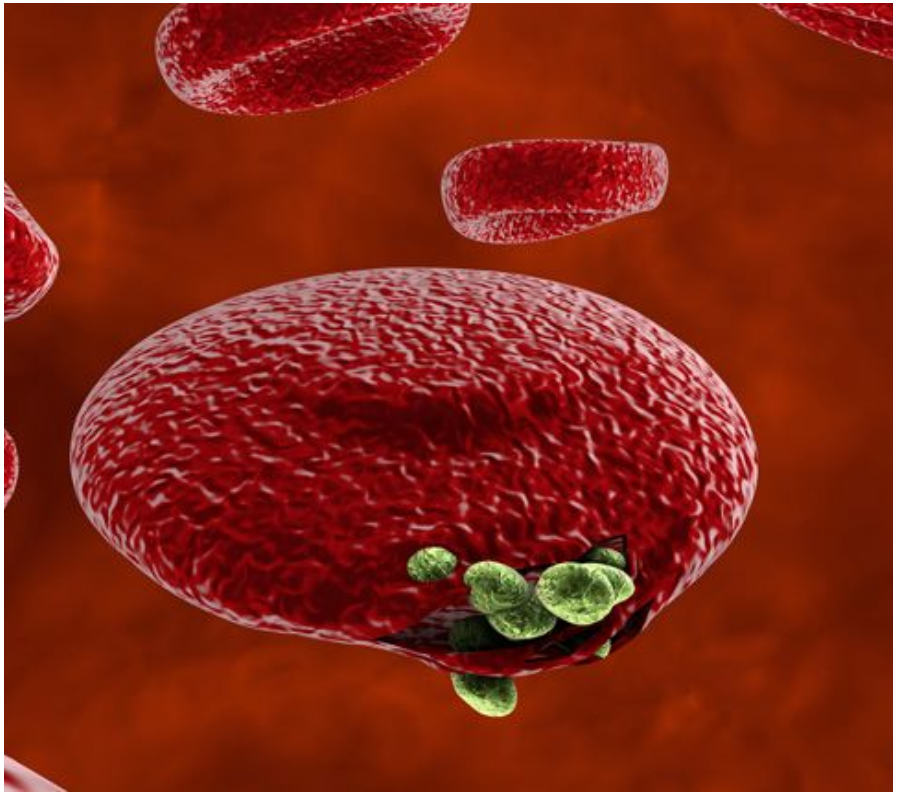
MIT ADSP Capstone ‘22

Ciara Adkins

https://www.linkedin.com/in/ciara-adkins/
https://github.com/ciaraadkins

Executive Summary

- The current process for detecting malaria is inconsistent and inefficient
- Machine learning can be used to detect parasitized malaria cells up to a 99% accuracy
- Before adoption, these models should be used alongside existing techniques in order to prove efficacy



Context

- ~627k people died from malaria in 2020
- Many malaria causing parasites have developed resistance to common antimalarial drugs
- Potentially fatal but curable with early and accurate intervention

Problem

- Diagnosis of malarial infection in a lab is a tedious and manual process
- Results can differ based on the observer

Goals

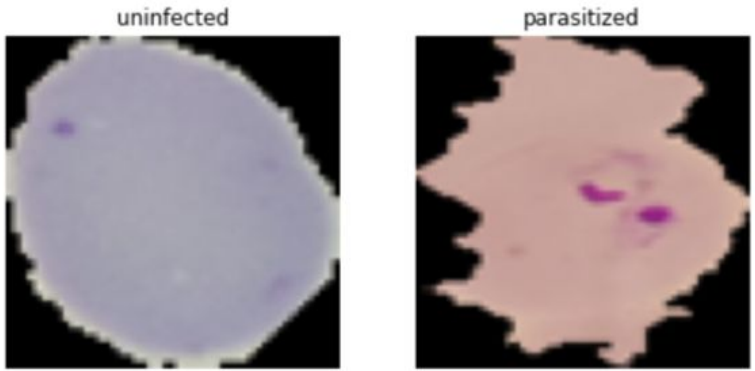
- Increase rate of early intervention
- Increase diagnostic accuracy, consistency, and reliability
- Decrease amount of time scientists have to dedicate towards diagnosis

Data Description

- 24k train images and 2.5k test images
- Each entry is a 64x64 pixel microscopic image of individual cells (RGB)
- Additionally, each entry is “labeled” parasitized (1) or healthy (0)

Approach

- Build a convolutional neural network (CNN) that can identify the parasitized cells



Process

- Ran nearly 30 different models with a variety of parameters and inputs
- Reviewed the Confusion Matrices
- Evaluated the Accuracy vs Epoch plots

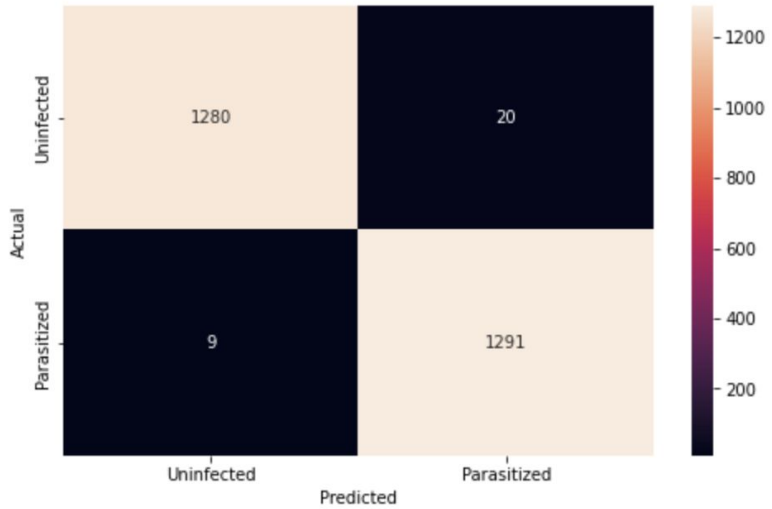
Solution

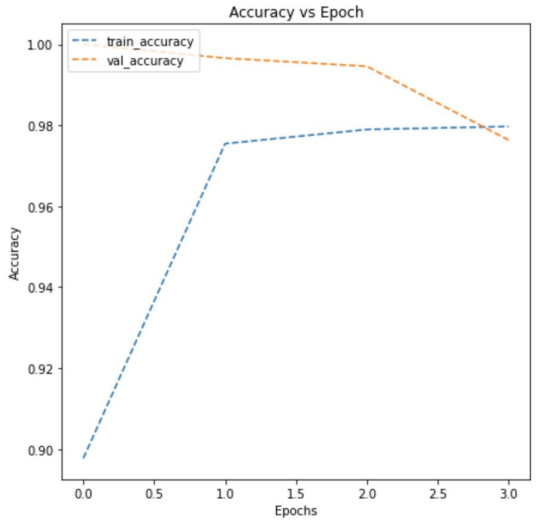
- Gaussian blurred input images
- 20+ layer CNN
- Batch size = 64
- Loss Fxn = Binary Crossentropy

```
model1_gbx2.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
model1_gbx2.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
model1_gbx2.add(MaxPooling2D(pool_size=2))
model1_gbx2.add(BatchNormalization())
model1_gbx2.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu"))
model1_gbx2.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu"))
model1_gbx2.add(MaxPooling2D(pool_size=2))
model1_gbx2.add(BatchNormalization())
model1_gbx2.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
model1_gbx2.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
model1_gbx2.add(MaxPooling2D(pool_size=2))
model1_gbx2.add(BatchNormalization())
model1_gbx2.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu"))
model1_gbx2.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu"))
model1_gbx2.add(MaxPooling2D(pool_size=2))
model1_gbx2.add(BatchNormalization())
model1_gbx2.add(Dropout(0.2))
```

Outcomes

- The model predicted the parasitized cells with a 99% accuracy
  - Precision: 99%
  - Recall: 99%
- Additionally the model minimizes false negatives





Stakeholder Recommendations

- Start using the deep learning algorithm alongside manual techniques to evaluate its effectiveness when compared to the current manual technique
- Use this additional data to continue to retrain the model
- Continue to compare the effectiveness over time
- Once the deep learning algorithm consistently meets or exceeds manual identification, the model can be used exclusively
- However, continue to retrain the model over time with additional data
    - When the model sees more data it will be able to be more accurate
    - This will also account for any changes in the appearance of malaria over time
- Caveat: No matter how good the deep learning algorithm, there will inevitably be use cases where someone has malaria and the algorithm does not detect it

Next Steps

- As we get more data, continue to refine the model
- Review the false negative use cases (examples below) and try to find commonalities and potentially create a second algorithm that would be able to detect these
- Malaria is caused by 5 different parasites, a dataset with more information on the strains might make it easier for us to create algorithms with more information

Appendix

Model No.	Notes	Inputs	Precision	Recall	F-1 Score	Accuracy	FN	FP
0.0	original	normalized	0.98	0.98	0.98	0.98	28	34
1.0	original	normalized	0.99	0.99	0.99	0.99	16	22
2.0	original	normalized	0.98	0.98	0.98	0.98	23	33
3.0	original	normalized	0.98	0.98	0.98	0.98	28	35
4.0	original	normalized	0.92	0.91	0.91	0.91	29	199
0.1.1	update loss_weights [1,20]	normalized	0.98	0.98	0.98	0.98	23	21
0.1.2	update loss_weights [20,1]	normalized	0.98	0.98	0.98	0.98	21	35
1.1.1	loss_weights=[1,20] batch_size=64	normalized	0.98	0.98	0.98	0.98	13	36
1.1.2	loss_weights=[20,1] batch_size=64	normalized	0.98	0.98	0.98	0.98	6	42
1.1.3	loss_weights=[50,1] batch_size=32	normalized	0.98	0.98	0.98	0.98	12	37
1.1.4	loss_weights=[50,1] batch_size=64	normalized	0.98	0.98	0.98	0.98	13	42
2.1.1	move BatchNorm and re-run	normalized	0.97	0.97	0.97	0.97	86	4
2.1.2	loss_weights=[50,1] batch_size=32	normalized	0.98	0.97	0.97	0.97	56	9
2.1.3	loss_weights=[1,50] batch_size=32	normalized	0.97	0.97	0.97	0.97	86	4
2.1.4	loss_weights=[1,50] batch_size=64	normalized	0.96	0.96	0.96	0.96	106	5
3.1	n/a							
4.1	add dropout layers							
0.2	batch_size=32	HSV	0.59	0.57	0.55	0.57	270	848
1.2	loss_weights=[20,1] batch_size=64	HSV	0.25	0.5	0.33	0.5	0	1300
2.2		HSV	n/a	n/a	n/a	n/a	n/a	n/a
3.2		HSV	n/a	n/a	n/a	n/a	n/a	n/a
4.2		HSV	n/a	n/a	n/a	n/a	n/a	n/a
0.3		Gaussian blur	0.97	0.97	0.97	0.97	57	28
1.3.1	batch_size=32	Gaussian blur	0.98	0.98	0.98	0.98	11	30
1.3.2	batch_size=64	Gaussian blur	0.98	0.98	0.98	0.98	9	36
1.3.3	batch_size=64 loss_weights=[1,50]	Gaussian blur	0.98	0.98	0.98	0.98	33	30
1.3.4	batch_size=64 loss_weights=[50,1]	Gaussian blur	0.98	0.98	0.98	0.98	16	28
2.3		Gaussian blur						
3.3		Gaussian blur						
4.3		Gaussian blur						

Information about all of the models that were run to get to the final model