BUG OFF MALARIA

DEEP LEARNING TECHNIQUES TO DETECT MALARIA USING IBM CLOUD

Pratham Shah(19BCE2028)
Annanya Popat (19BBS0048)
Jhanak Gupta(19BCE2037)
Neeladri Chatterjee(17MIS7137)

INDEX

INTRODUCTION

- Malaria is a life-threatening disease that is spread by the Plasmodium parasites. It is detected by trained microscopists who analyze microscopic blood smear images.
- Modern deep learning techniques may be used to do this analysis automatically.
- The need for the trained personnel can be greatly reduced with the development of an automatic accurate and efficient model. In our project, we have developed an automated convolutional neural network (CNN) based model which shall predict if a person has malaria using the microscopic blood smear images.
- Our deep learning-based model can detect malarial parasites from microscopic images with an accuracy of 96-97% .
- For a practical and real world application, we have made a website with which our model has been integrated with using flask.

PROBLEM STATEMENT

Implement Deep Learning Techniques to detect malaria using IBM Cloud

SOLUTION

Malaria can be detected by analyzing microscopic blood smear images.
This process can be automated by using convolutional neural networks (CNN) model which shall use a training dataset of almost 28000 images to get a high accuracy in prediction as it becomes really important in the field of medicine to get diagnosed properly.
We have made a website which allows the user to upload the blood smear image and our model will predict if malaria is detected or not.

TRAINING THE MODEL

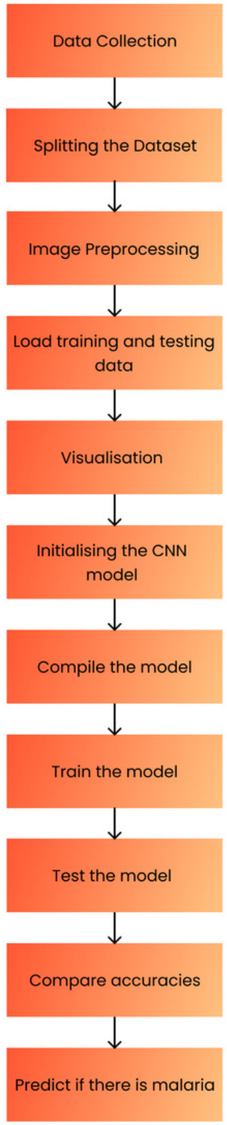
- We have used Python which is a statistical mathematical programming language.
- Import all the necessary libraries in python.
- Apply image DataGenerator functionality to train and test set
- Import required model building libraries
- Initialize the model , add convolution layers , pooling layer , flatten layer and dense layer
- Compile the model fit and save the model

TESTING THE MODEL

The model is to be tested with different images to know if it is predicting correctly. Import the packages that are used to load the model and get the predictions.

Pre-processing the image includes converting the image to array and resizing according to the model. Give the pre-processed image to the model to know to which class your model belongs to.

PROCESS FLOWCHART



RESULT

So, after running the model through various iterations and a lot of fine-tuning, we need up with a model having an accuracy of around 96-97%, and was able to predict the uninfected and parasitized quite accurately.

The results mentioned above are presented in the next few slides:

```
In [22]: hist = malaria_model.fit_generator(x_train, steps_per_epoch = 344 , epochs = 30 , validation_data = x_test, validation_steps = 86, callbacks=[early_stop] )
```

```
Epoch 1/30
344/344 [==============================] - 388s 1s/step - loss: 0.6884 - acc: 0.5360 - val_loss: 0.6773 - val_acc: 0.5525
Epoch 2/30
344/344 [==============================] - 384s 1s/step - loss: 0.3112 - acc: 0.8857 - val_loss: 0.1673 - val_acc: 0.9502
Epoch 3/30
344/344 [==============================] - 384s 1s/step - loss: 0.1716 - acc: 0.9479 - val_loss: 0.1386 - val_acc: 0.9539
Epoch 4/30
344/344 [==============================] - 385s 1s/step - loss: 0.1536 - acc: 0.9532 - val_loss: 0.1375 - val_acc: 0.9531
Epoch 5/30
344/344 [==============================] - 384s 1s/step - loss: 0.1502 - acc: 0.9546 - val_loss: 0.1556 - val_acc: 0.9602
Epoch 6/30
344/344 [==============================] - 386s 1s/step - loss: 0.1426 - acc: 0.9561 - val_loss: 0.1357 - val_acc: 0.9600
Epoch 7/30
344/344 [==============================] - 385s 1s/step - loss: 0.1411 - acc: 0.9564 - val_loss: 0.1226 - val_acc: 0.9608
Epoch 8/30
344/344 [==============================] - 385s 1s/step - loss: 0.1385 - acc: 0.9581 - val_loss: 0.1429 - val_acc: 0.9593
Epoch 9/30
344/344 [==============================] - 386s 1s/step - loss: 0.1334 - acc: 0.9583 - val_loss: 0.1379 - val_acc: 0.9622
Epoch 10/30
344/344 [==============================] - 387s 1s/step - loss: 0.1286 - acc: 0.9591 - val_loss: 0.1377 - val_acc: 0.9622
Epoch 11/30
344/344 [==============================] - 384s 1s/step - loss: 0.1294 - acc: 0.9589 - val_loss: 0.1263 - val_acc: 0.9609
Epoch 12/30
344/344 [==============================] - 385s 1s/step - loss: 0.1306 - acc: 0.9589 - val_loss: 0.1343 - val_acc: 0.9618
Epoch 13/30
344/344 [==============================] - 383s 1s/step - loss: 0.1256 - acc: 0.9606 - val_loss: 0.1170 - val_acc: 0.9582
```

```
Epoch 14/30
344/344 [==============================] - 387s 1s/step - loss: 0.1271 - acc: 0.9595 - val_loss: 0.1269 - val_acc: 0.9602
Epoch 15/30
344/344 [==============================] - 391s 1s/step - loss: 0.1237 - acc: 0.9615 - val_loss: 0.1469 - val_acc: 0.9602
Epoch 16/30
344/344 [==============================] - 391s 1s/step - loss: 0.1218 - acc: 0.9617 - val_loss: 0.1365 - val_acc: 0.9615
Epoch 17/30
344/344 [==============================] - 389s 1s/step - loss: 0.1231 - acc: 0.9609 - val_loss: 0.1167 - val_acc: 0.9618
Epoch 18/30
344/344 [==============================] - 391s 1s/step - loss: 0.1202 - acc: 0.9606 - val_loss: 0.1173 - val_acc: 0.9635
Epoch 19/30
344/344 [==============================] - 390s 1s/step - loss: 0.1180 - acc: 0.9622 - val_loss: 0.1280 - val_acc: 0.9622
Epoch 20/30
344/344 [==============================] - 385s 1s/step - loss: 0.1203 - acc: 0.9611 - val_loss: 0.1144 - val_acc: 0.9613
Epoch 21/30
344/344 [==============================] - 386s 1s/step - loss: 0.1202 - acc: 0.9617 - val_loss: 0.1128 - val_acc: 0.9617
Epoch 22/30
344/344 [==============================] - 384s 1s/step - loss: 0.1176 - acc: 0.9626 - val_loss: 0.1155 - val_acc: 0.9606
Epoch 23/30
344/344 [==============================] - 386s 1s/step - loss: 0.1186 - acc: 0.9607 - val_loss: 0.1102 - val_acc: 0.9624
Epoch 24/30
344/344 [==============================] - 385s 1s/step - loss: 0.1137 - acc: 0.9631 - val_loss: 0.1072 - val_acc: 0.9618
Epoch 25/30
344/344 [==============================] - 385s 1s/step - loss: 0.1159 - acc: 0.9619 - val_loss: 0.1148 - val_acc: 0.9622
Epoch 26/30
344/344 [==============================] - 386s 1s/step - loss: 0.1190 - acc: 0.9622 - val_loss: 0.1117 - val_acc: 0.9628
Epoch 27/30
344/344 [==============================] - 384s 1s/step - loss: 0.1130 - acc: 0.9627 - val_loss: 0.1111 - val_acc: 0.9620
Epoch 28/30
344/344 [==============================] - 386s 1s/step - loss: 0.1171 - acc: 0.9613 - val_loss: 0.1025 - val_acc: 0.9620
Epoch 29/30
344/344 [==============================] - 385s 1s/step - loss: 0.1142 - acc: 0.9629 - val_loss: 0.1263 - val_acc: 0.9626
Epoch 30/30
344/344 [==============================] - 386s 1s/step - loss: 0.1121 - acc: 0.9638 - val_loss: 0.1262 - val_acc: 0.9618
```

RESULT

Check whether the person is infected with malaria or not

Interactive, simple and quick way to predict malaria

PREDICTION

Upload image and check whether it is infected or not with malaria

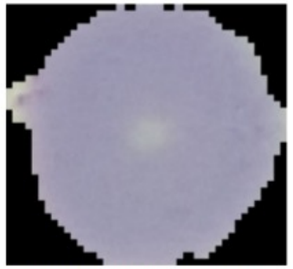
The prediction is : infected

Check whether the person is infected with malaria o

Interactive, simple and quick way to predict malaria

PREDICTION

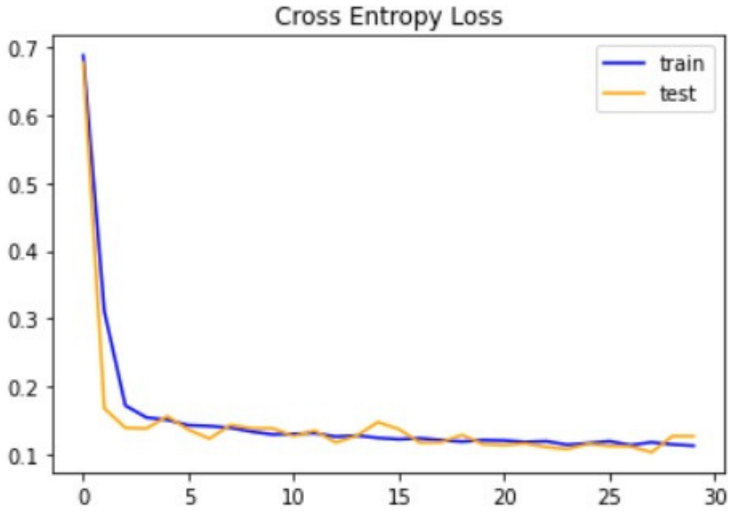
Upload image and check whether it is infected or not with malaria



The prediction is : uninfected

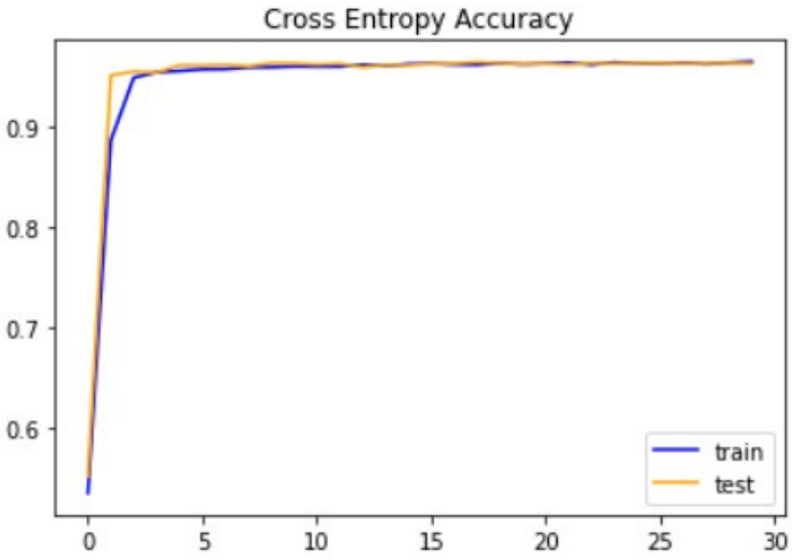
In [23]:
```
# Plotting the accuracies
import matplotlib.pyplot as plt

# plot training and testing loss
plt.title('Cross Entropy Loss')
plt.plot(hist.history['loss'], color='blue', label='train')
plt.plot(hist.history['val_loss'], color='orange', label='test')
plt.legend()
```

Out[23]: <matplotlib.legend.Legend at 0x7f627508fe90>



In [25]:
```
# plot training and testing accuracy
plt.title('Cross Entropy Accuracy')
plt.plot(hist.history['acc'], color='blue', label='train')
plt.plot(hist.history['val_acc'], color='orange', label='test')
plt.legend()
```

Out[25]: <matplotlib.legend.Legend at 0x7f6257e8d8d0>



CONCLUSION

Well, malaria being one of the fatal diseases, has been a problem for a long time. But using deep learning models, like the ones that we used in our project, the detection of the disease can be done much faster and can be deal with the proper treatment accordingly.

Technologies like these neural networks have been a blessing to us and using them in the right way is gonna benefit all of us.

THANK YOU