Algorithm Engineering and Machine Learning for Biomedical Engineers Pablo Loo Quintero

Dr. John Hanks BMEN 489

Image classification of Malaria parasite reproductive cycle with Machine Learning algorithms

Abstract

In this project, we intend to create a Deep Learning Algorithm capable of predicting the stage of the Malaria parasite in a given image as accurately as possible. An initial dataset of around 1000 images belonging to 7 different classifications was provided by Dr. Paul Gordon to derive the train/test data. Two architectures (custom and AlexNet) were compared, and the highest achieving network was AlexNet, with a 70% training accuracy. In the past, AlexNet was tested to classify images of standing and laying cows, where it reached a 93% training accuracy which makes it clear that the model is very good at classifying certain types of datasets such as the laying and standing cows dataset, however, further work is required for the model to be able to satisfactorily classify the malaria dataset.

Introduction

Malaria is caused by a parasite called plasmodium, which is injected by mosquitoes when they take a blood meal and goes through an asexual reproduction cycle consisting of five stages: merozoites (extracellular parasite), ring (early) trophozoites, mature (late) trophozoites, schizont, and ruptured schizont. Currently, there are several types of diagnostics for malaria, each with their own pros and cons. For example, the Rapid Diagnostic Tests are very good at providing confirmation in a relatively short time frame, but they are unable to provide parasitemia (amount of parasites in the bloodstream). Lab-Based Microscopy is very good at providing confirmation, parasitemia, and even speciation, but it takes a significant amount of time.This means that there is a need for a diagnostic system that can provide at least confirmation and parasitemia in a short timeframe, which is what this project intends to do using a machine learning approach.[1] The model used in this project will take in the five reproductive stages mentioned above as well as two additional categories (artifacts and white blood cells) as input for training and testing.

Related Work

Vijayalakshmi and Rajesh Kanna introduced a novel deep neural network model for identifying infected falciparum malaria parasites using a transfer learning approach achieved by unifying a Visual Geometry Group (VGG) network and Support Vector Machine (SVM), resulting in a classification accuracy of 93.1%.[2] For this project, however, we decided to use standard approaches to image data classification due to time constraints.

Standard Method

Convolutional Neural Networks are the standard approach for image classification

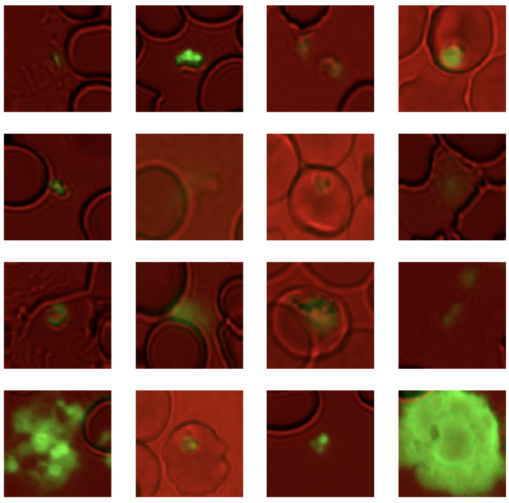
problems. This is the method used in this project. More specifically, we compared a custom network to AlexNet. In the next sections, we will discuss the approaches taken by researchers that go beyond the fundamentals.

Other Methods

Industry deep learning models for biological image classification often require a balance between accuracy and computational cost, which leads to the combination of texture-based feature extraction techniques and more traditional techniques such as decision tree induction algorithms, neural networks, nearest neighbors, and Support vector machines, just like the one used by Vijayalakshmi (et. al.).[2][3]

Data Sets, Features, and Preprocessing

An initial dataset of around 1000 images belonging to 7 different classifications was provided by Dr. Paul Gordon to derive the train/test data. The classifications correspond to the five stages of the reproductive cycle of plasmodium (merozoites (extracellular parasite), ring (early) trophozoites, mature (late) trophozoites, schizont, and ruptured schizont) as well as two additional categories corresponding to artifacts and white blood cells. It is clear that these images are much more complex than the cow laying and cow standing images. For the cow standing/cow laying datast, it was quite easy for the model to identify the images given that standing cows have exceptional vertical features (standing legs) as compared to the cows laying. However, by looking at the malaria dataset (below), it can be seen that the features that differentiate one class from another go way beyond shapes. For example, in order to differentiate an extracellular parasite from a ring or mature trophozoite, it is important to note that the key characteristic is that the bright green spot representing the parasite is located at a convex position relative to the cell membrane, whereas either one of the ring or mature trophozoite is located at a concave position with respect to the cell membrane. Of course this cannot be the only feature taken into account to differentiate given that the ring and mature trophozoites are each a class of their own, and would likely be differentiated by the difference in size and intensity of the green spot (again, representing the parasite).



The images were normalized from -1 to 1 using MobileNet's ImageDataGenerator and setting the preprocessing function to preprocess input. Because the data of the classifications was very unbalanced (two

categories would hold as many as 200 images, whereas the rest did not exceed 70 images), data augmentation was necessary, and thus the data underwent preprocessing for the following parameters and their respective ranges:

Parameter	Range/Mode
Shearing	0 - 15 radians
Zoom	0 -15 radians
Rotation	0 - 15 radians
Fill	Nearest

Such parameters are for the most part, intuitive. These preprocessing techniques allowed the model to create more data from the original and ultimately, become more accustomed to it.

Preliminary Model Architectures

As mentioned earlier, Convolutional Neural Networks are the go-to option for image classification problems. In previous works, a simple binary sequential network with MobileNet as the base model and three dense hidden layers each with a Rectified Linear Unit activation function and an output layer with the Softmax activation function was used. This model was used to classify images of standing and laying cows and served as the foundation for the baseline model of this project.

Baseline Model or Algorithm

The option was given to use a random classifier as a baseline model, but instead, a modified version of the cow laying/cow standing model was used. The only difference between the latter and the former is the addition of 4000 neurons per layer. This was done with the purpose of allowing the network to identify certain features in the images. The number of epochs was increased from 5 to 15 in order to improve feature extraction while avoiding overfitting.

Comparative Model (Model 2)

Meet AlexNet, one of the most popular convolutional neural networks out there. It contains five convolutional layers, some of which are followed by max-pooling layers and use the rectified linear unit activation function. The last three are fully connected layers. It also contains dropout layers weaved in between so as to avoid overfitting.

Layer (type)	Output Shape	Param #
conv2d_20 (Conv2D)	(None, 55, 55, 96)	34944
batch_normalization_20 (Batc	(None, 55, 55, 96)	384
max_pooling2d_12 (MaxPooling	(None, 27, 27, 96)	0
conv2d_21 (Conv2D)	(None, 27, 27, 256)	614656
batch_normalization_21 (Batc	(None, 27, 27, 256)	1024
max_pooling2d_13 (MaxPooling	(None, 13, 13, 256)	0
conv2d_22 (Conv2D)	(None, 13, 13, 384)	885120
batch_normalization_22 (Batc	(None, 13, 13, 384)	1536
conv2d_23 (Conv2D)	(None, 13, 13, 384)	147840
batch_normalization_23 (Batc	(None, 13, 13, 384)	1536
conv2d_24 (Conv2D)	(None, 13, 13, 256)	98560
batch_normalization_24 (Batc	(None, 13, 13, 256)	1024
max_pooling2d_14 (MaxPooling	(None, 6, 6, 256)	0
flatten_4 (Flatten)	(None, 9216)	0
dense_12 (Dense)	(None, 4096)	37752832
dropout_8 (Dropout)	(None, 4096)	0
dense_13 (Dense)	(None, 4096)	16781312
dropout_9 (Dropout)	(None, 4096)	0
dense_14 (Dense)	(None, 7)	28679

Total params: 56,349,447
Trainable params: 56,346,695
Non-trainable params: 2,752

Final Model

The final model was AlexNet combined with preprocessing of the images and we are currently exploring other methods used by experts in the field for implementing additional feature engineering and making the data easier to analyze.

Experiments and Results

Prior to the implementation of AlexNet, a custom model was created through trial and error. Through this empirical approach, it was discovered that for this dataset, after a certain number of neurons per hidden layer, the training accuracy reached a plateau, and so the number of neurons was set to 5024, and although this could potentially make the model computationally expensive, the training time can significantly be reduced with a Graphics Processing Unit (GPU). The number of epochs was set to 30 for the same reason. After reaching 30 epochs, the training accuracy would stay within the (65 - 74)% range, which is not optimal. The Adam optimizer was not altered, given that better performing optimizers tend to significantly increase the number of parameters by an order of magnitude which again, makes the system computationally expensive overall.[4]

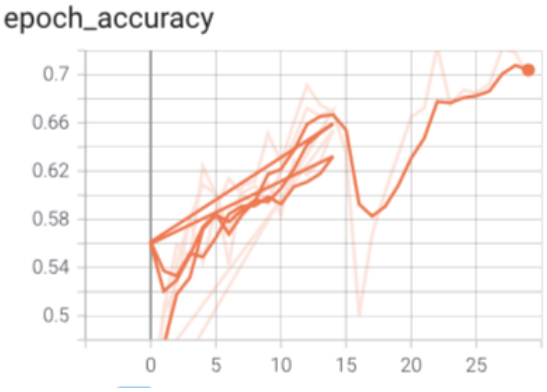
Training

The default learning rate for the Adam optimizer is 0.001, and the total amount of epochs was set to 30. The training data is preprocessed by reshaping them to the shape 227 x 227 x 3, where the first two terms are the pixel size and the third term represents the three color channels (RGB). The data is then normalized to be between -1 and 1 and ultimately shuffled. The batch size is set to 16 and the class mode is set to categorical. A 50% dropout rate is applied in between the last three dense (fully connected) layers.

Model Evaluation

All the models in this project were evaluated based on the ability to successfully categorize the images given as one of the seven classes. As aforementioned, the highest achieving model was AlexNet with a 70% accuracy, compared to a 64% accuracy by the custom model.

```
Epoch 1/15
63/63 [==============================] - 132s 2s/step - loss: 0.9723 - accuracy: 0.7284
Epoch 2/15
63/63 [==============================] - 131s 2s/step - loss: 0.9224 - accuracy: 0.7448
Epoch 3/15
63/63 [==============================] - 131s 2s/step - loss: 0.9365 - accuracy: 0.7161
Epoch 4/15
63/63 [==============================] - 132s 2s/step - loss: 0.8893 - accuracy: 0.7373
Epoch 5/15
63/63 [==============================] - 137s 2s/step - loss: 0.8541 - accuracy: 0.7800
Epoch 6/15
63/63 [==============================] - 133s 2s/step - loss: 0.8072 - accuracy: 0.7666
Epoch 7/15
63/63 [==============================] - 133s 2s/step - loss: 0.7838 - accuracy: 0.7685
Epoch 8/15
63/63 [==============================] - 133s 2s/step - loss: 1.1317 - accuracy: 0.7493
Epoch 9/15
63/63 [==============================] - 132s 2s/step - loss: 0.8230 - accuracy: 0.7472
Epoch 10/15
63/63 [==============================] - 132s 2s/step - loss: 0.7113 - accuracy: 0.7929
Epoch 11/15
63/63 [==============================] - 133s 2s/step - loss: 0.7597 - accuracy: 0.7911
Epoch 12/15
63/63 [==============================] - 133s 2s/step - loss: 0.5776 - accuracy: 0.8180
Epoch 13/15
63/63 [==============================] - 133s 2s/step - loss: 0.6379 - accuracy: 0.8203
Epoch 14/15
63/63 [==============================] - 134s 2s/step - loss: 0.5701 - accuracy: 0.8209
Epoch 15/15
63/63 [==============================] - 133s 2s/step - loss: 0.6021 - accuracy: 0.8172
```



Parameter	Testing Accuracy (Percentage %)
Artifacts	100
Extracellular Parasite	50
Mature Trophozoites	63
Ring Trophozoites	0
Ruptured Schizont	33
Schizonts	15
White Blood Cells	100

Discussion

Clearly, the model feature identification process is not tailored for this sort of application. In past works, this had been useful for identifying binary classifications whose differences in features were prevalent and clear, such as the cow standing/cow laying dataset. The features in this dataset, however, go beyond shapes. In order to be able to identify the images, one must recognize not only the shapes involved but also the differences in intensity, as well as the position of certain features. For example, the relationship must be identified between the position of a high-intensity point with respect to other features, such as a convex or concave curve in order to determine whether this is an extracellular or intracellular parasite. These features may be very obvious to a human, but in order to accomplish the same result in a machine learning model, additional steps must be taken aside from feeding the input to the network. Some suggestions for this will be introduced in the future work section.

Conclusions

We compared a custom machine learning model to AlexNet, who once more

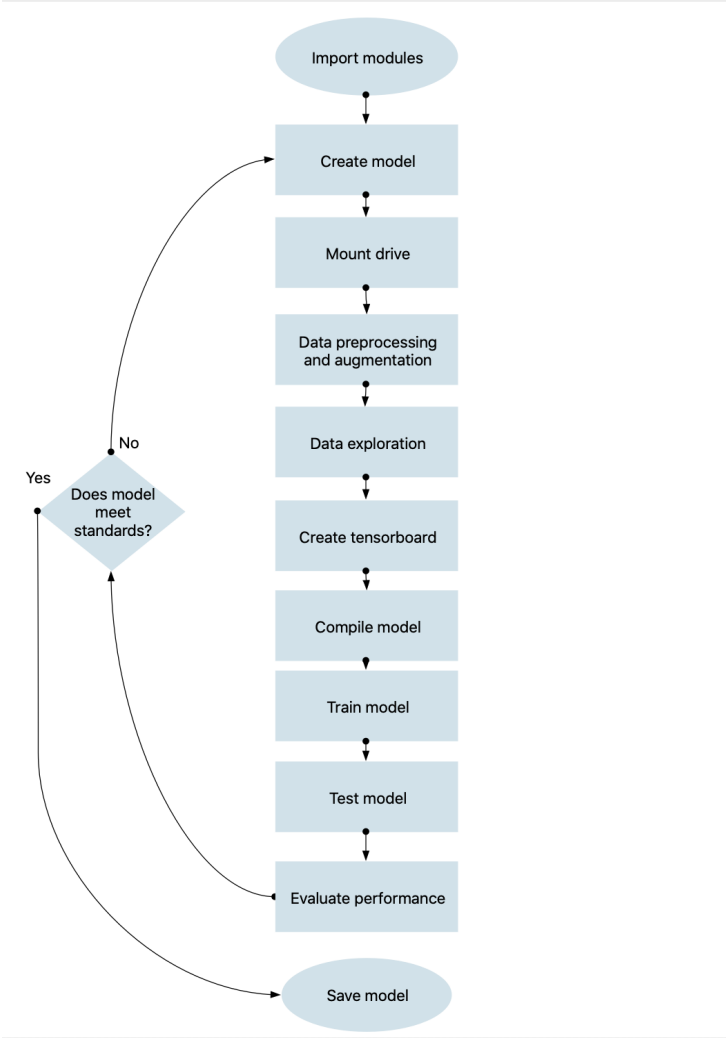
confirmed its reputation for being one of the best general application CNNs out there, having reached a 70% training accuracy for the complex malaria dataset. AlexNet utilizer rectified linear unit activation functions, as opposed to tanh. This gives AlexMet an edge with respect to training time. AlexNet also has the advantage of allocating space to multiple GPUs, which can significantly decrease the training time. The goal of this project was to achieve an 70% classification accuracy, which is supposed to simulate human classification ability. Certainly, the goal was achieved for the training data, but not so for the testing data, so we will keep on exploring options to optimize the model and process the data so that it is easier to classify.

Future Work

Currently, there are many advanced Computer Vision techniques used for multiple applications, and each one of them will be explored to discover whether they can be exploited for our application. Some of these techniques are described:

Object Detection uses a technique called Sliding Window , which applies a CNN to many different crops of the image, which is computationally expensive. Region-based Convolutional Neural Networks (R-CNNs) was proposed as a solution to this problem. Using R-CNNs, the Selective Search algorithm searches for “blobby” image regions that are likely to contain objects. The algorithm then runs a CNN on top of each of these proposed regions. The result is then fed to s Support Vector Machine to classify the region and a linear regression to bound the box of the object.Fast Region-based Convolutional neural Networks (Fast R-CNNs) is a much faster descendant of R-CNN, and the main difference is that it requires feature extraction prior to blobby region proposal, thus only running one CNN over the entire image, and it also “replaces SVM with a softmax layer, thus extending the neural network for predictions instead of creating a new model”. The ultimate model that willis called Faster R-CNN, which replaces the Selective Search algorithm with a Region Proposal Network to predict proposals from the features.[5]

Flowchart



References

[1]*Portable Microscopy for Quantitative Malaria Diagnosis and Monitoring at the Point of Care*, Paul Gordon Ph.D., Texas A&M University

[2] Vijayalakshmi A, & Rajesh Kanna B. (2019). Deep learning approach to detect malaria from microscopic images. *Multimedia Tools and Applications, 79*(21-22), 15297-15317. doi:10.1007/s11042-019-7162-y

[3]Carlos Affonso, André Luis Debiaso Rossi, Fábio Henrique Antunes Vieira, André Carlos Ponce de Leon Ferreira de Carvalho, Deep learning for biological image classification,Expert Systems with Applications,Volume 85,2017,Pages 114-122,ISSN 0957-4174.

[4] Doshi, S. (2020, August 03). Various optimization algorithms for training neural networks. https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

[5] Le, J. (2020, January 29). The 5 computer vision techniques that will change how you see the world https://heartbeat.fritz.ai/the-5-computer-vision-techniques-that-will-change-how-you-see-the-world-1ee19334354b

Source Code

```
import pandas as pd
import numpy as np
import os
from PIL import Image
import matplotlib.pyplot as plt
from google.colab import drive
import tensorflow as tf
import tensorflow.keras
from tensorflow import keras
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense,Dropout
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.mobilenet import preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator

model = keras.models.Sequential([
   keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4),
activation='relu', input_shape=(227,227,3)),
   keras.layers.BatchNormalization(),
   keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
   keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1),
activation='relu', padding="same"),
   keras.layers.BatchNormalization(),
   keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
```

```
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(1,1), strides=(1,1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1),
activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(7, activation='softmax') # Output cells = No.
categories
])
model.summary()

# Import files
drive.mount('/content/drive')
train_path = "/content/drive/My Drive/Colab Notebooks/Project 1/Train"
#train_path = "/content/drive/My Drive/Colab Notebooks/dataset_cows.zip
(Unzipped Files)/dataset_cows/train"

# pre processing for mobilenet (mobilenet images are normalized from -1 to
1)
train_datagen=ImageDataGenerator(preprocessing_function=preprocess_input,
                                 shear_range = 0.15,
                                 zoom_range =.15,
                                 rotation_range =15,
                                 fill_mode='nearest')

file_path = train_path
# create train_generator iterator object
train_generator=train_datagen.flow_from_directory(file_path,
                                                  target_size=(227,227),
```

```
                                                color_mode='rgb',
                                                batch_size= 16,
                                                seed=None,
                                                class_mode='categorical',
                                                shuffle=True)


# Data Exploration
import matplotlib.pyplot as plt

x_image,y_label = next(train_generator) # next function gets the next
batch of 16 images

fig = plt.figure(figsize=(10, 10))  # set the size of the figures for
display
for i in range(x_image.shape[0]):
   plt.subplot(4,4, i + 1)
   plt.axis('off')
   img_out = ((x_image[i]+1)/2)*255  # convert back from imagenet format,
imagenet is 0 mean values ranging from -1 to 1
   img_out = img_out.astype(np.uint8)  # convert to unsigned integer 8-bit
for display
   plt.imshow(img_out, cmap=plt.cm.gray_r, interpolation='nearest')

# Compile and run the model
# Adam optimizer
# loss function will be binary cross entropy
# evaluation metric will be accuracy

model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['a
ccuracy'])


# Epoch ==> one pass through all of the training samples
# Batch ==> subset of samples (loading all samples may use too much
memory)

step_size_train=train_generator.n//train_generator.batch_size
```

```
history = model.fit(train_generator,
          steps_per_epoch=step_size_train,
          epochs=30,
          callbacks=[tensorboard])



# Predict one image at a time
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
image_size = 227 # 102

########### Inputs ###################
# change these values to predict the images in a directory
#data_set = "test"
data_set = "Test"
classification = "White Blood Cells"
#mypath = "/content/drive/My Drive/Colab Notebooks/dataset_cows.zip
(Unzipped Files)/dataset_cows/" + data_set + "/" + classification + "/"
mypath = "/content/drive/My Drive/Colab Notebooks/Project 1/" + data_set +
"/" + classification + "/"
class_names = list(train_generator.class_indices.keys()) # Classification
names list
print(class_names)

def convert_to_array(file_path):
   img1 = load_img(file_path, target_size=(image_size, image_size))
   a_img = img_to_array(img1)
   a_img = np.expand_dims(a_img, axis=0)
   return a_img

from os import listdir
from os.path import isfile, join
onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath,f))]

#print(onlyfiles)

prediction_list = []
j=0
cnt = 0

# keep all the samples in numpy array for plotting
# X_samples =np.ndarray((train_generator.n,224,224,3)).astype('float32')
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

for f in onlyfiles:

    #img = mpimg.imread(mypath+f)
    #imgplot = plt.imshow(img)
    #plt.show()


    input_image = convert_to_array(mypath+f)

    # X_samples[i] = input_image

    # images need to be normalized using the same method that imagenet used
    processed_image_mobilenet= preprocess_input(input_image) #.copy())

    # create a list of predictions for each image in the directory
    prediction_list.append(model.predict(input_image))



    #predicted_class = class_names[np.where(prediction_list[0][0] ==
max(prediction_list[0][0]))[0][0]]
    #print(str(mypath+f), "     Predicted Class: " + predicted_class + " "
+   "   Actual Class: " + classification ) # Print predicted classification
    #i += 1
    #if predicted_class == classification:
    #  cnt +=1

#print(prediction_list[0][0])
#print(prediction_list[1][0])

for i in range(0,len(prediction_list)):
 #print(prediction_list[i][0])
 predicted_class = class_names[np.where(prediction_list[i][0] ==
max(prediction_list[i][0]))[0][0]]
 print(str(mypath+f), "     Predicted Class: " + predicted_class + " " +
"   Actual Class: " + classification ) # Print predicted classification
 j += 1
 if predicted_class == classification:
   cnt +=1
print("\n Accuracy =  " + str(100*cnt/j) + "%")
```

Algorithm Engineering and Machine Learning for Biomedical Engineers Pablo Loo Quintero

Dr. John Hanks BMEN 489

```
%load_ext tensorboard
%tensorboard --logdir "/content/drive/My Drive/Colab Notebooks/Project
1/logs"
```