

DEEP LEARNING CS6005

Mini Project Assignment

Detecting Malaria cells using Convolutional Neural Network

Date : 19-04-2021

Praveen RS

2018103577

P Batch

Problem Statement

Malaria is the deadliest disease in the earth and big hectic work for the health department. The traditional way of diagnosing malaria is by schematic examining blood smears of human beings for parasite-infected red blood cells under the microscope by lab or qualified technicians. This process is inefficient and the diagnosis depends on the experience and well knowledgeable person needed for the examination.



Hence, we can use highly accurate classification of malaria-infected cells using deep convolutional neural networks. First, we describe image processing methods used for segmentation of red blood cells from whole slide images. We then discuss the procedures of compiling a pathologists-curated image dataset for training a deep neural network, as well as data augmentation methods used to significantly increase the size of the dataset, in light of the overfitting problem associated with training deep convolutional neural networks. We will then compare the classification accuracies obtained by deep convolutional neural networks through training, validating, and testing with various combinations of the datasets.

Dataset Details

The dataset contains 2 folders and a total of 27,558 images.

- Infected
- Uninfected

This Malaria Cells Images Dataset is taken from the official NIH Website: <https://ceb.nlm.nih.gov/repositories/malaria-datasets/> . The images were manually annotated by an expert slide reader at the Mahidol-Oxford Tropical Medicine Research Unit in Bangkok, Thailand. The de-identified images and annotations are archived at NLM. A level-set based algorithm is applied to detect and segment the red blood cells.

Infected Blood Cells:



Uninfected blood cells:



Modules

1.Loading the dataset and Pre-Processing

The dataset is loaded and appended into a NumPy array. One image of each class is plotted as output.

Now, we scale the dataset. The image pixel values were scaled between 0 and 1 by dividing by 255. The data is segregated into two folders: train and test.

2. Building the CNN Model

The order in which the layers are added to model are as follows:

- 1) A 2D Convolution layer with 16 filters of kernel size (3,3) with a default stride of (1,1) and no padding. ReLU activation layer is used. Further on, Maxpooling of size (2,2) and stride of (2,2) is used. Also, a Dropout of 0.2 is added.
- 2) A 2D Convolution layer with 32 filters of kernel size (3,3) with a default stride of (1,1) and no padding. ReLU activation layer is used. Further on, Maxpooling of size (2,2) and stride of (2,2) is used. Also, a Dropout of 0.3 is added.
- 3) A 2D Convolution layer with 64 filters of kernel size (3,3) with a default stride of (1,1) and no padding. ReLU activation layer is used. Further on, Maxpooling of size (2,2) and stride of (2,2) is used. Also, a Dropout of 0.3 is added.
- 4) The output from the previous layer is flattened.
- 5) A Fully Connected Dense layer with 64 units and a dropout of 0.5 is added with ReLU activation.
- 6) A Fully Connected Dense layer with 1 unit is used as output layer with Sigmoid activation.
- 7) Model is built with loss as Binary cross-entropy, optimizer is Adam and metrics as Accuracy.


3. Training and Visualizing Results

Early stopping is used with patience as 2 with monitoring validation loss. The model is trained with 20 epochs and batch size of 16. Further the model is evaluated with accuracy metrics. Further on, model accuracy graph and model loss graph are plotted for visualization.

Parameters in the CNN Model

Optimizer	Adam
Loss	Binary Cross-entropy
Epochs	20
Batch size	16

CNN Model Summary

 ML

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 16)	448
max_pooling2d (MaxPooling2D)	(None, 63, 63, 16)	0
dropout (Dropout)	(None, 63, 63, 16)	0
conv2d_1 (Conv2D)	(None, 61, 61, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 32)	0
dropout_1 (Dropout)	(None, 30, 30, 32)	0
conv2d_2 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_2 (Dropout)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0

max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 32)	0
dropout_1 (Dropout)	(None, 30, 30, 32)	0
conv2d_2 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_2 (Dropout)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 64)	802880
dropout_3 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65
Total params: 826,529		
Trainable params: 826,529		
Non-trainable params: 0		

Coding Snapshots

Importing necessary libraries

```
[1] ▶ ML
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dropout, Flatten, Dense, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping

[2] ▶ ML
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Displaying Uninfected and Infected Cell tissues

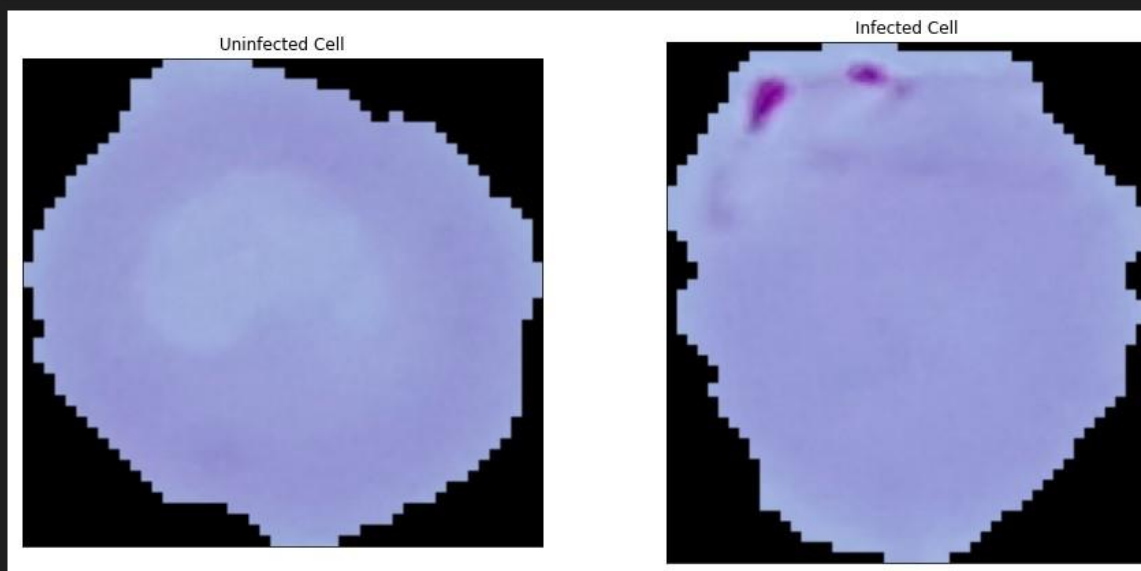
```
[3] ▶ ML
import cv2

upic='../input/cell-images-for-detecting-malaria/cell_images/Uninfected/C100P61ThinF_IMG_20150918_144104_cell_131.png'
apic='../input/cell-images-for-detecting-malaria/cell_images/Parasitized/C100P61ThinF_IMG_20150918_144104_cell_164.png'
plt.figure(1, figsize = (15 , 7))
plt.subplot(1 , 2 , 1)
plt.imshow(cv2.imread(upic))
plt.title('Uninfected Cell')
plt.xticks([], plt.yticks([]))

plt.subplot(1 , 2 , 2)
plt.imshow(cv2.imread(apic))
plt.title('Infected Cell')
plt.xticks([], plt.yticks([]))

plt.show()
```

plt.show()



```
[4] ▶ ML
width = 128
height = 128
```


Dividing Dataset into two folders train and test

```
[5] ▶ ML
datagen = ImageDataGenerator(rescale=1/255.0, validation_split=0.2)
```

Preparing train and test Image Generator

```
[6] ▶ ML
trainDatagen = datagen.flow_from_directory(directory='../input/cell-images-for-detecting-malaria/cell_images/cell_images/',
                                           target_size=(width,height),
                                           class_mode = 'binary',
                                           batch_size = 16,
                                           subset='training')

print(len(trainDatagen))

Found 22048 images belonging to 2 classes.
1378
```

```
[8] ▶ ML
valDatagen = datagen.flow_from_directory(directory='../input/cell-images-for-detecting-malaria/cell_images/cell_images/',
                                          target_size=(width,height),
                                          class_mode = 'binary',
                                          batch_size = 16,
                                          subset='validation')

print(len(valDatagen))

Found 5510 images belonging to 2 classes.
345
```

Preparing the model

```
[10] ▶ ML
model = Sequential()
model.add(Conv2D(16,(3,3),activation='relu',input_shape=(128,128,3)))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.2))

model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.3))

model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1,activation='sigmoid'))
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 126, 126, 16)	448
max_pooling2d (MaxPooling2D)	(None, 63, 63, 16)	0
dropout (Dropout)	(None, 63, 63, 16)	0
conv2d_1 (Conv2D)	(None, 61, 61, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 32)	0
dropout_1 (Dropout)	(None, 30, 30, 32)	0
conv2d_2 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_2 (Dropout)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0

max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 32)	0
dropout_1 (Dropout)	(None, 30, 30, 32)	0
conv2d_2 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_2 (Dropout)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 64)	802880
dropout_3 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65
=====		
Total params: 826,529		
Trainable params: 826,529		
Non-trainable params: 0		


```
[12] > M1
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

[13] > M1
early_stop = EarlyStopping(monitor='val_loss',patience=2)
```

The training stops at the 3rd epoch itself since we have used the 'early_stop' callback to prevent the overfitting of the data.

```
[14] > M1
history = model.fit_generator(generator = trainDatagen,
                             steps_per_epoch = len(trainDatagen),
                             epochs =20,
                             validation_data = valDatagen,
                             validation_steps=len(valDatagen),
                             callbacks=[early_stop])

Epoch 1/20
1378/1378 [=====] - 146s 106ms/step - loss: 0.3657 - accuracy: 0.8476 - val_loss: 0.1695 - val_accuracy: 0.9363
Epoch 2/20
1378/1378 [=====] - 48s 35ms/step - loss: 0.1771 - accuracy: 0.9469 - val_loss: 0.1971 - val_accuracy: 0.9376
Epoch 3/20
1378/1378 [=====] - 48s 35ms/step - loss: 0.1560 - accuracy: 0.9535 - val_loss: 0.2200 - val_accuracy: 0.9303

[20] > M1
history.history['accuracy']

[0.8476052284240723, 0.9468886256217957, 0.9535105228424072]
```

Visualization

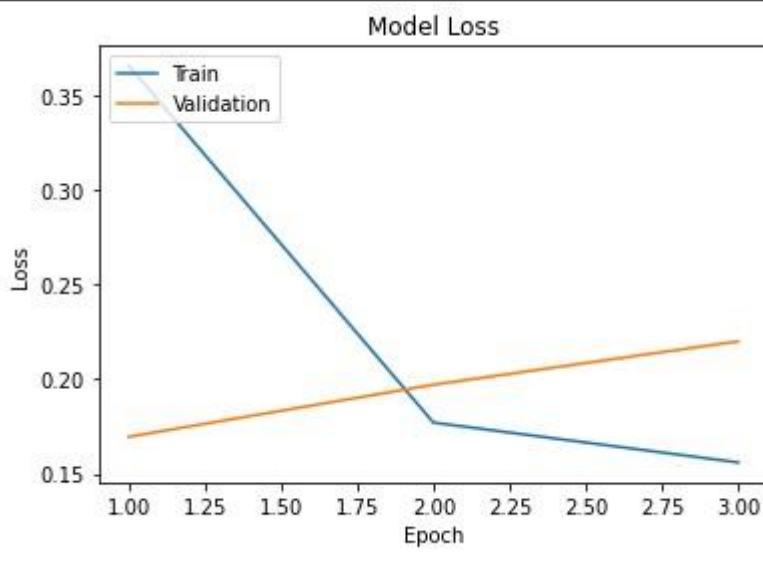
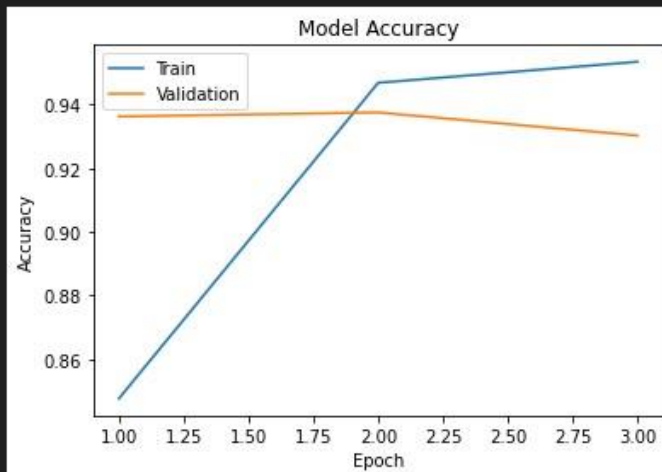
Plots

```
[21] > M1
def plotLearningCurve(history,epochs):
    epochRange = range(1,epochs+1)
    plt.plot(epochRange,history.history['accuracy'])
    plt.plot(epochRange,history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train','Validation'],loc='upper left')
    plt.show()

    plt.plot(epochRange,history.history['loss'])
    plt.plot(epochRange,history.history['val_loss'])
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train','Validation'],loc='upper left')
    plt.show()
```

[22] ▶ M1

```
plotLearningCurve(history,3)
```



Results

We can observe that model obtains an accuracy of 93.03% on the test data (validation) and a loss of 0.22.

```
1378/1378 [=====] - 48s 35ms/step - loss: 0.1771 - accuracy: 0.9469 - val_loss: 0.1971 - val_accuracy: 0.9376  
Epoch 3/20  
1378/1378 [=====] - 48s 35ms/step - loss: 0.1560 - accuracy: 0.9535 - val_loss: 0.2200 - val_accuracy: 0.9303
```

References

G. Shekar, S. Revathy and E. K. Goud, "Malaria Detection using Deep Learning," 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184), Tirunelveli, India, 2020, pp. 746-750, doi: 10.1109/ICOEI48184.2020.9143023.