IBM Coursera
Advanced Data Science Capstone

Malaria Cell image classification

Roshan Thilakarathne

Use case

- Every 2 minutes, a child dies of malaria. And each year, more than 200 million new cases of the disease are reported – World Health Organization

- Caused by *Plasmodium* parasites that infect the red blood cells

- Common practice is manual identification of parasitized (infected) cells in microscopic thin-film.

- Proposing a deep learning convolution neural network (CNN) to classifying infected and uninfected cells and comparing the model performance with traditional models

Data Set

- Dataset is taken from NIH – U.S. National Library of Medicine
- Description of the dataset https://ceb.nlm.nih.gov/repositories/malaria-datasets/
- FTP link to the dataset ftp://lhcftp.nlm.nih.gov/Open-Access-Datasets/Malaria/cell_images.zip
- The dataset contains a total of 27,558 cell images with 13,779 parasitized images and 13,779 uninfected images

Models

- Deep Convolution Models

  CNN – 9 layers

  CNN – 12 layers

  CNN with image augmentation – 10 layer

  CNN with Histogram Equalization – 10 layer

- Non Deep Learning Models

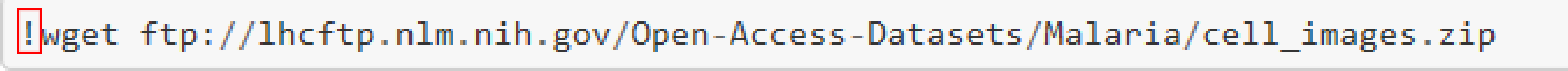
  Basic Neural Network

  Support Vector machine classifier

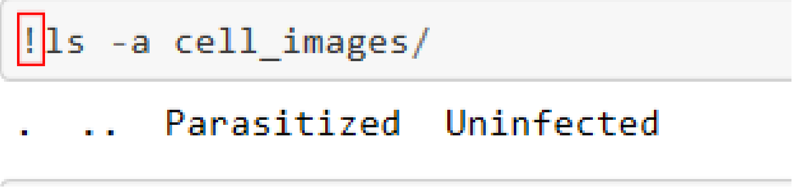
  K-nearest Neighbors classifier

ETL & Feature Engineering

- https://github.com/roshanthi/IBM-Coursera-/blob/master/Malaria_cell_classification_%20ETL.ipynb



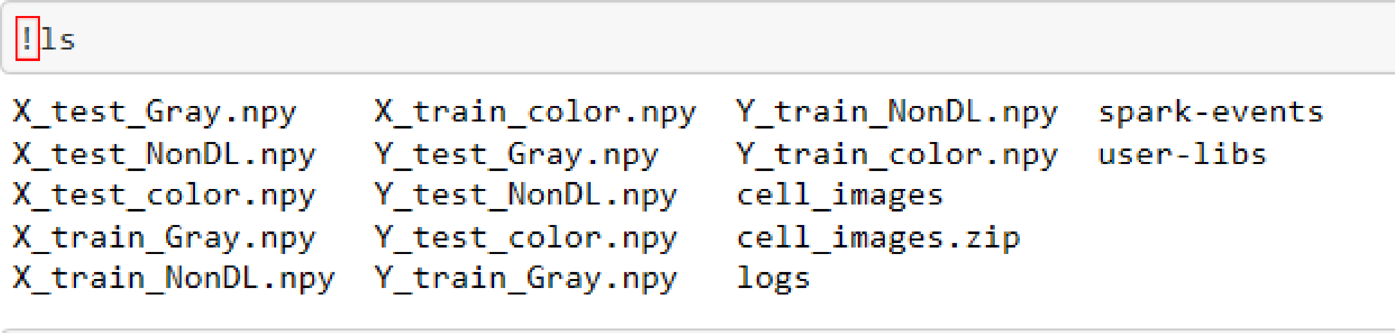
```
!wget ftp://lhcftp.nlm.nih.gov/Open-Access-Datasets/Malaria/cell_images.zip
```



```
!ls -a cell_images/
```

```
.  ..  Parasitized  Uninfected
```

- https://github.com/roshanthi/IBM-Coursera-/blob/master/Malaria_cell_classification_%20Feature_eng.ipynb



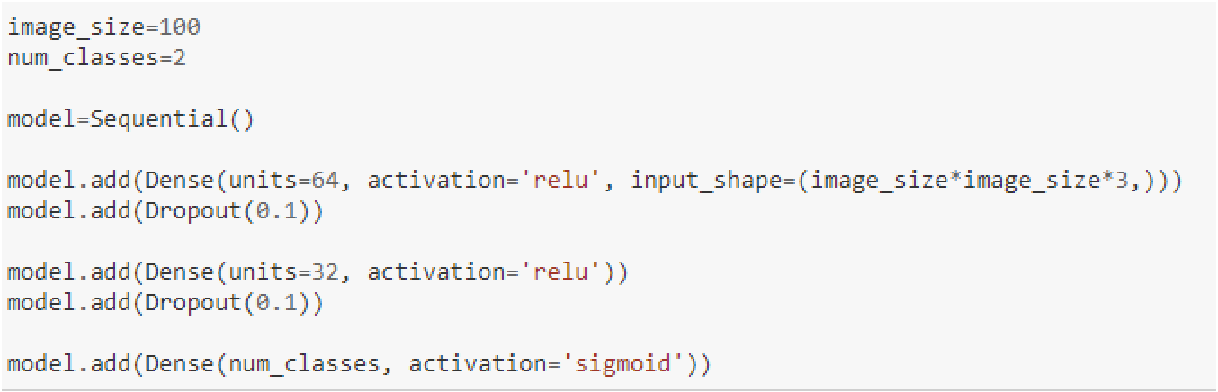
```
!ls
```

```
X_test_Gray.npy     X_train_color.npy  Y_train_NonDL.npy  spark-events
X_test_NonDL.npy    Y_test_Gray.npy    Y_train_color.npy  user-libs
X_test_color.npy    Y_test_NonDL.npy   cell_images
X_train_Gray.npy    Y_test_color.npy   cell_images.zip
X_train_NonDL.npy   Y_train_Gray.npy   logs
```

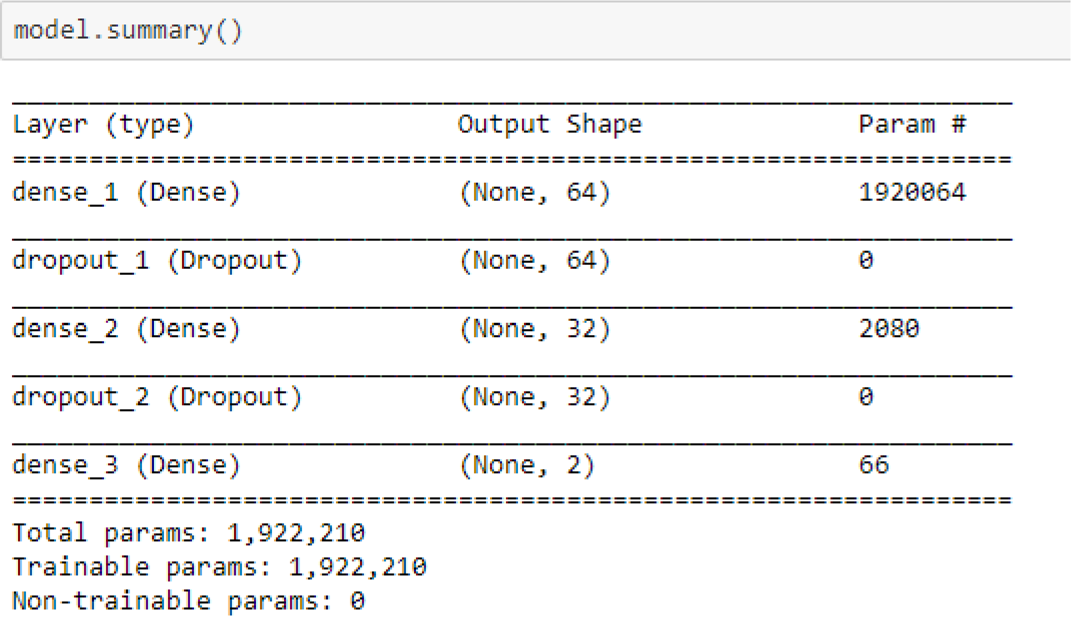
Non - Deep Learning Models

Basic Neural Network

https://github.com/roshanthi/IBM-Coursera-/blob/master/Malaria_cell_classification_%20BasicNN.ipynb



```
image_size=100
num_classes=2

model=Sequential()

model.add(Dense(units=64, activation='relu', input_shape=(image_size*image_size*3,)))
model.add(Dropout(0.1))

model.add(Dense(units=32, activation='relu'))
model.add(Dropout(0.1))

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



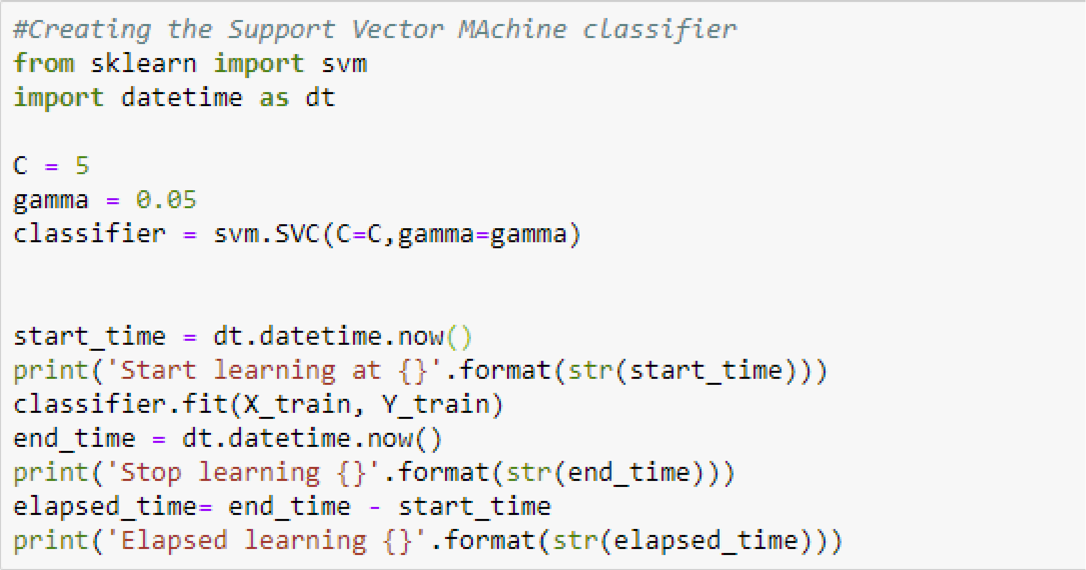
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                1920064
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 66
=================================================================
Total params: 1,922,210
Trainable params: 1,922,210
Non-trainable params: 0
```

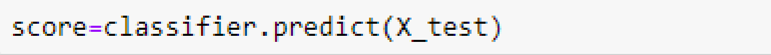
Non - Deep Learning Models

https://github.com/roshanthi/IBM-Coursera-/blob/master/Malaria_cell_classification_NonDL_Models.ipynb

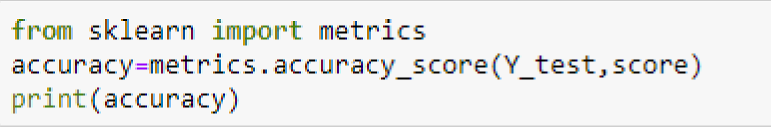
Support Vector Machine



```
#Creating the Support Vector MAchine classifier
from sklearn import svm
import datetime as dt

C = 5
gamma = 0.05
classifier = svm.SVC(C=C,gamma=gamma)


start_time = dt.datetime.now()
print('Start learning at {}'.format(str(start_time)))
classifier.fit(X_train, Y_train)
end_time = dt.datetime.now()
print('Stop learning {}'.format(str(end_time)))
elapsed_time= end_time - start_time
print('Elapsed learning {}'.format(str(elapsed_time)))
```

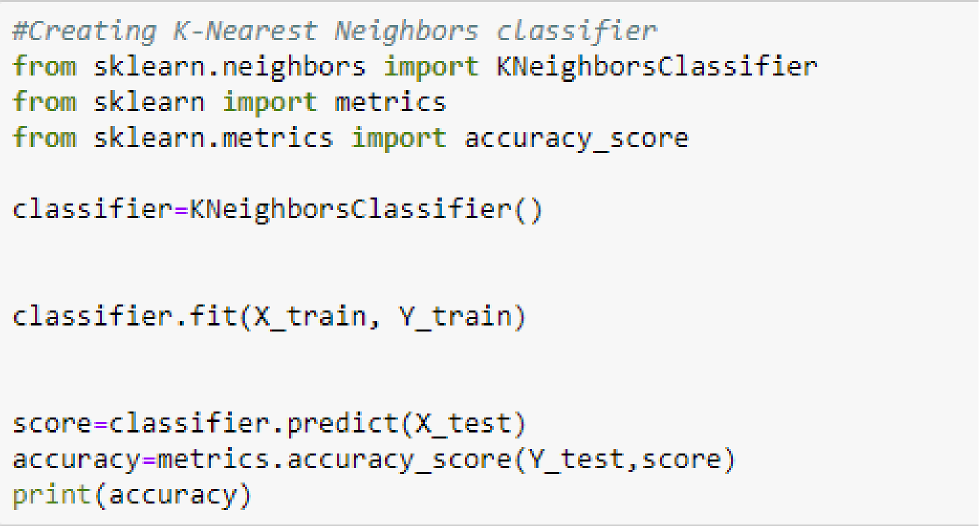


```
score=classifier.predict(X_test)
```



```
from sklearn import metrics
accuracy=metrics.accuracy_score(Y_test,score)
print(accuracy)
```

K- Nearest Neighbors



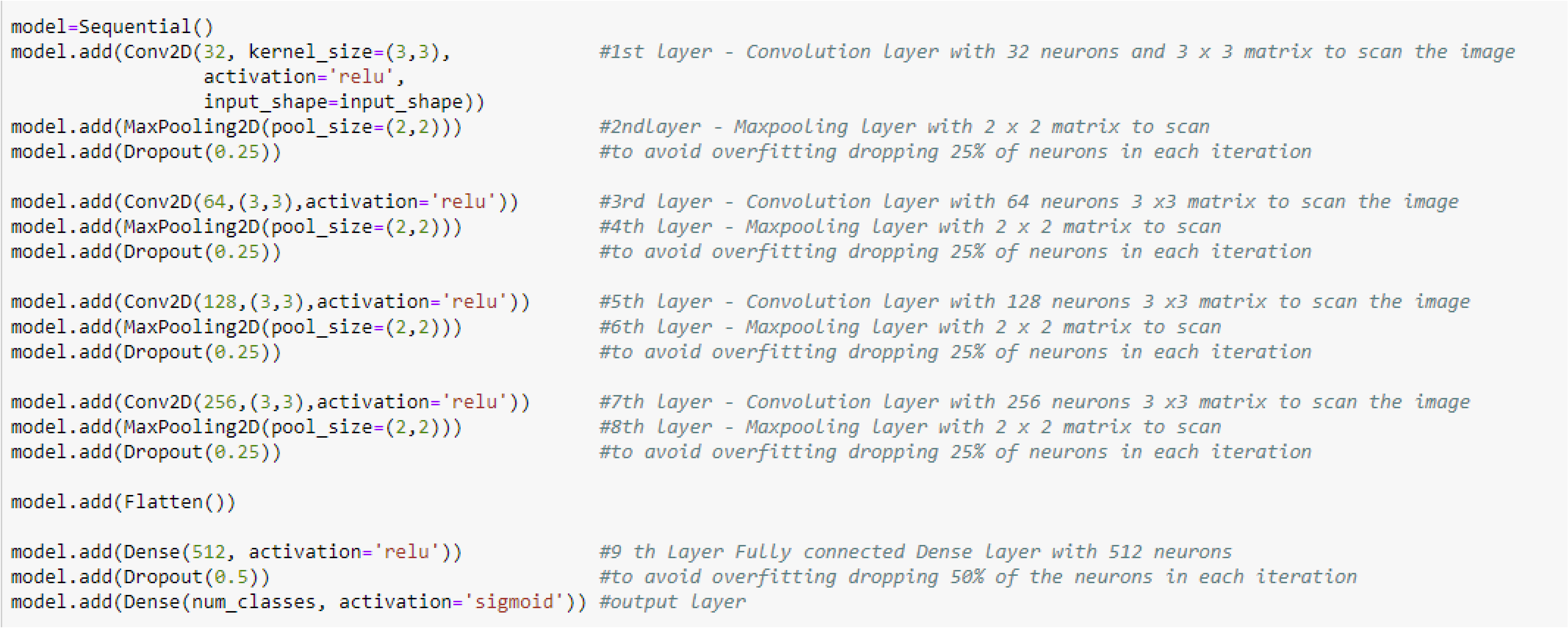
```
#Creating K-Nearest Neighbors classifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score

classifier=KNeighborsClassifier()


classifier.fit(X_train, Y_train)


score=classifier.predict(X_test)
accuracy=metrics.accuracy_score(Y_test,score)
print(accuracy)
```

Deep Learning Models

https://github.com/roshanthi/IBM-Coursera-/blob/master/Malaria_cell_classification_DL_Models.ipynb

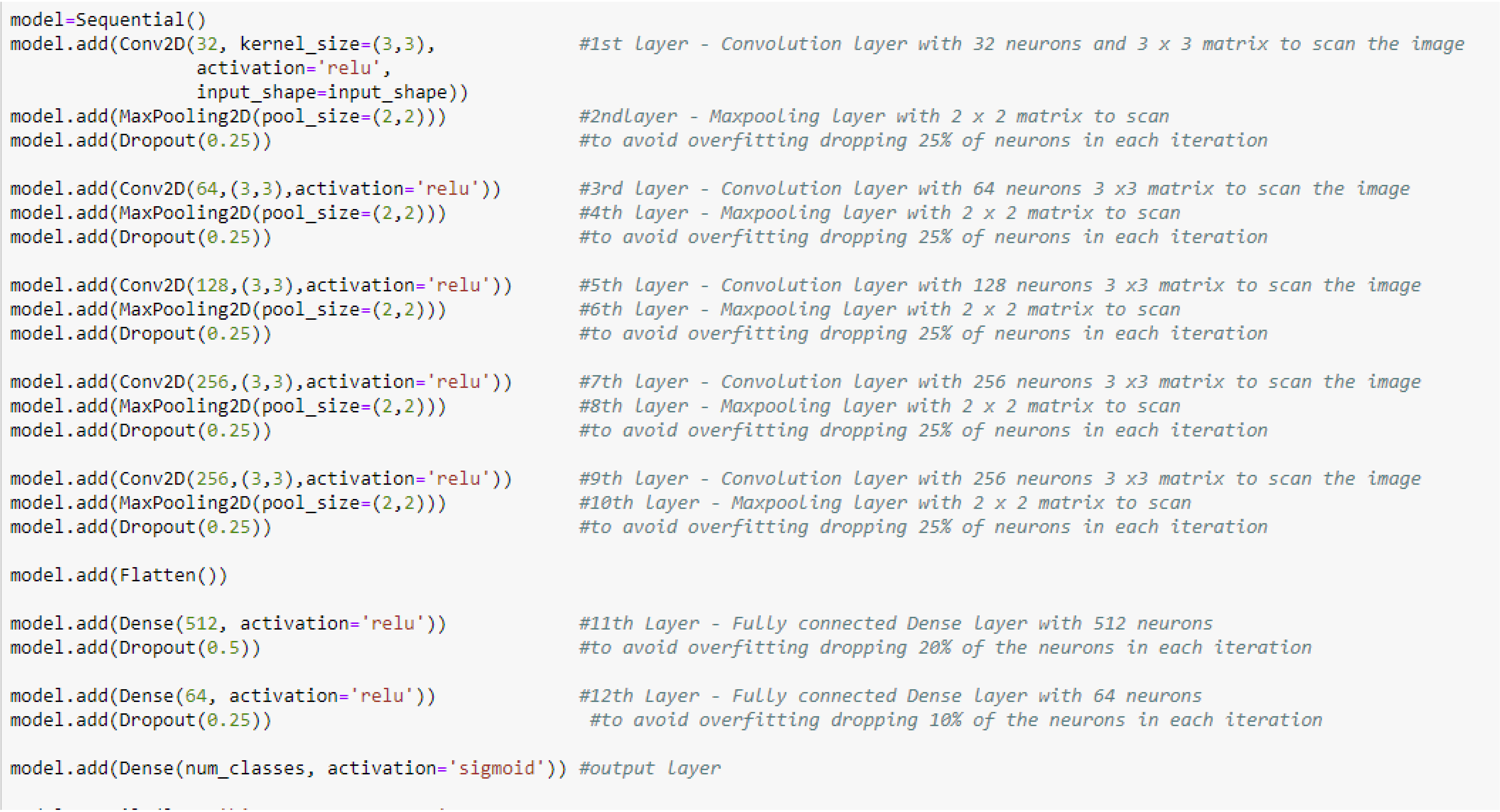
https://github.com/roshanthi/IBM-Coursera-/blob/master/Malaria%20_cell_classification_histogram_eql.ipynb

1. CNN – 9 layers



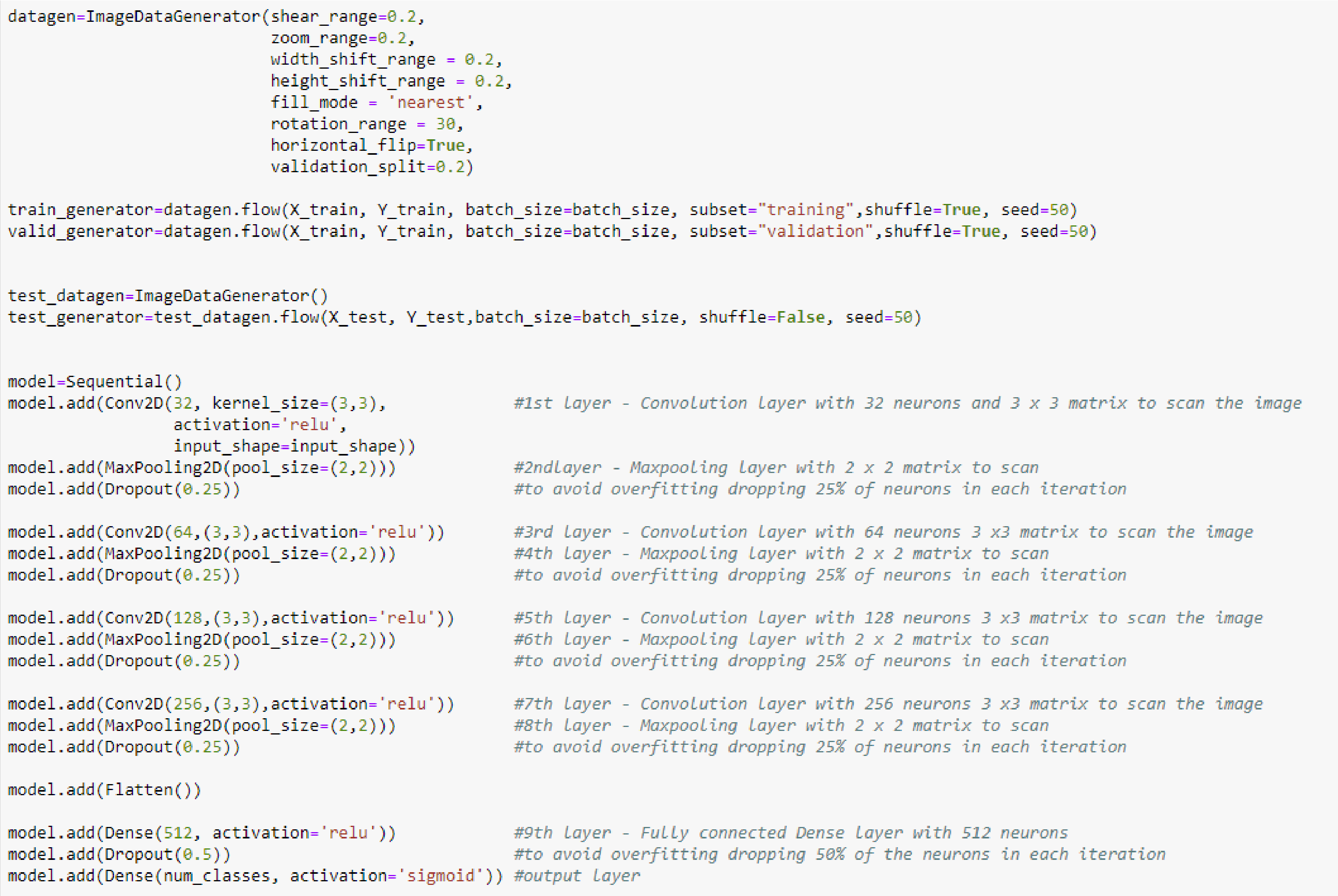
```
model=Sequential()
model.add(Conv2D(32, kernel_size=(3,3),             #1st layer - Convolution layer with 32 neurons and 3 x 3 matrix to scan the image
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2,2)))            #2ndLayer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(64,(3,3),activation='relu'))       #3rd layer - Convolution layer with 64 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))            #4th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(128,(3,3),activation='relu'))      #5th layer - Convolution layer with 128 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))            #6th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(256,(3,3),activation='relu'))      #7th layer - Convolution layer with 256 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))            #8th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Flatten())

model.add(Dense(512, activation='relu'))            #9 th Layer Fully connected Dense layer with 512 neurons
model.add(Dropout(0.5))                             #to avoid overfitting dropping 50% of the neurons in each iteration
model.add(Dense(num_classes, activation='sigmoid')) #output layer
```

2. CNN – 12 layers



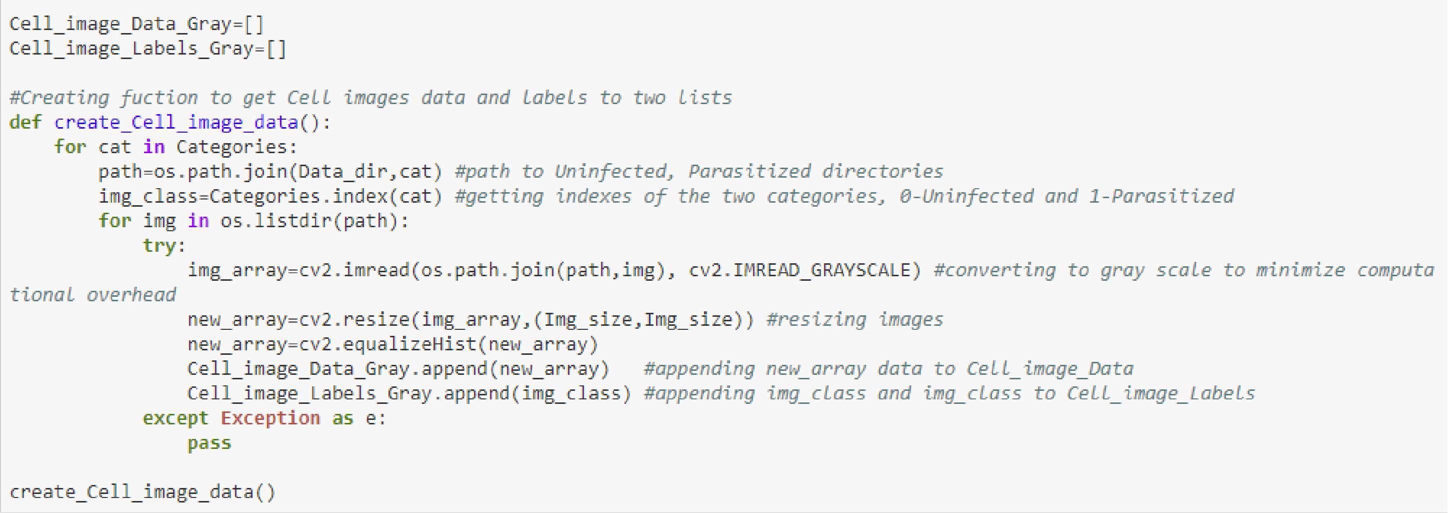
```
model=Sequential()
model.add(Conv2D(32, kernel_size=(3,3),             #1st layer - Convolution layer with 32 neurons and 3 x 3 matrix to scan the image
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2,2)))           #2ndLayer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(64,(3,3),activation='relu'))       #3rd layer - Convolution layer with 64 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))           #4th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(128,(3,3),activation='relu'))      #5th layer - Convolution layer with 128 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))           #6th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(256,(3,3),activation='relu'))      #7th layer - Convolution layer with 256 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))           #8th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(256,(3,3),activation='relu'))      #9th layer - Convolution layer with 256 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))           #10th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Flatten())

model.add(Dense(512, activation='relu'))            #11th Layer - Fully connected Dense layer with 512 neurons
model.add(Dropout(0.5))                             #to avoid overfitting dropping 20% of the neurons in each iteration

model.add(Dense(64, activation='relu'))             #12th Layer - Fully connected Dense layer with 64 neurons
model.add(Dropout(0.25))                             #to avoid overfitting dropping 10% of the neurons in each iteration

model.add(Dense(num_classes, activation='sigmoid')) #output layer
```

3. CNN with image augmentation – 9 layer

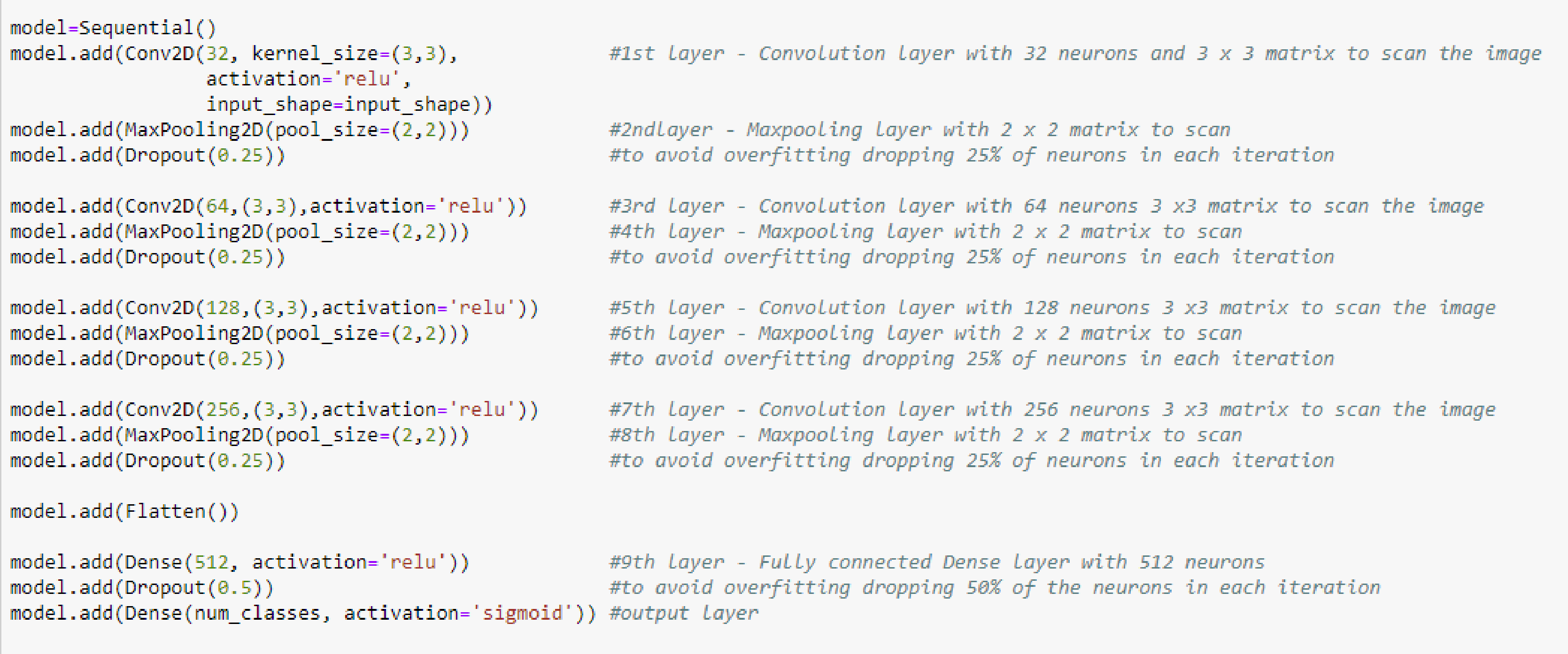


```
datagen=ImageDataGenerator(shear_range=0.2,
                           zoom_range=0.2,
                           width_shift_range = 0.2,
                           height_shift_range = 0.2,
                           fill_mode = 'nearest',
                           rotation_range = 30,
                           horizontal_flip=True,
                           validation_split=0.2)

train_generator=datagen.flow(X_train, Y_train, batch_size=batch_size, subset="training",shuffle=True, seed=50)
valid_generator=datagen.flow(X_train, Y_train, batch_size=batch_size, subset="validation",shuffle=True, seed=50)


test_datagen=ImageDataGenerator()
test_generator=test_datagen.flow(X_test, Y_test,batch_size=batch_size, shuffle=False, seed=50)


model=Sequential()
model.add(Conv2D(32, kernel_size=(3,3),             #1st layer - Convolution layer with 32 neurons and 3 x 3 matrix to scan the image
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2,2)))            #2ndlayer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(64,(3,3),activation='relu'))       #3rd layer - Convolution layer with 64 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))            #4th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(128,(3,3),activation='relu'))      #5th layer - Convolution layer with 128 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))            #6th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(256,(3,3),activation='relu'))      #7th layer - Convolution layer with 256 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))            #8th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                            #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Flatten())

model.add(Dense(512, activation='relu'))            #9th layer - Fully connected Dense layer with 512 neurons
model.add(Dropout(0.5))                             #to avoid overfitting dropping 50% of the neurons in each iteration
model.add(Dense(num_classes, activation='sigmoid')) #output layer
```

4. CNN with Histogram Equalization – 9 layer



```
Cell_image_Data_Gray=[]
Cell_image_Labels_Gray=[]

#Creating fuction to get Cell images data and labels to two lists
def create_Cell_image_data():
    for cat in Categories:
        path=os.path.join(Data_dir,cat) #path to Uninfected, Parasitized directories
        img_class=Categories.index(cat) #getting indexes of the two categories, 0-Uninfected and 1-Parasitized
        for img in os.listdir(path):
            try:
                img_array=cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE) #converting to gray scale to minimize computa
tional overhead
                new_array=cv2.resize(img_array,(Img_size,Img_size)) #resizing images
                new_array=cv2.equalizeHist(new_array)
                Cell_image_Data_Gray.append(new_array)   #appending new_array data to Cell_image_Data
                Cell_image_Labels_Gray.append(img_class) #appending img_class and img_class to Cell_image_Labels
            except Exception as e:
                pass

create_Cell_image_data()
```



```
model=Sequential()
model.add(Conv2D(32, kernel_size=(3,3),             #1st layer - Convolution layer with 32 neurons and 3 x 3 matrix to scan the image
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2,2)))             #2ndlayer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                             #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(64,(3,3),activation='relu'))        #3rd layer - Convolution layer with 64 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))             #4th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                             #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(128,(3,3),activation='relu'))       #5th layer - Convolution layer with 128 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))             #6th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                             #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Conv2D(256,(3,3),activation='relu'))       #7th layer - Convolution layer with 256 neurons 3 x3 matrix to scan the image
model.add(MaxPooling2D(pool_size=(2,2)))             #8th layer - Maxpooling layer with 2 x 2 matrix to scan
model.add(Dropout(0.25))                             #to avoid overfitting dropping 25% of neurons in each iteration

model.add(Flatten())

model.add(Dense(512, activation='relu'))             #9th layer - Fully connected Dense layer with 512 neurons
model.add(Dropout(0.5))                              #to avoid overfitting dropping 50% of the neurons in each iteration
model.add(Dense(num_classes, activation='sigmoid')) #output layer
```

Performance Indicator : Accuracy

Model Name	Test Accuracy
CNN – 9 layers	0.9580
CNN – 12 layers	0.9569
CNN with image augmentation	0.9506
CNN with Histogram Equalization	0.9462
Basic Neural Network	0.5012
Support Vector machine	0.5872
K-nearest Neighbors	0.5867

Summary

- All the CNNs works well for this data set while CNN with 9 layers shows the best accuracy.
- Further tasks :

  Train models longer (increase the no of epochs) and see if it improves the accuracy

https://github.com/roshanthi/IBM-Coursera-