This project deals with the detection of Malarial cells and differentiates the parasitized cells from the uninfected cells.

# Data Extraction

The data is easily available at [Kaggle](https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria). You need to have a Kaggle account in order to download the dataset. The dataset contains a total of 27,558 images (335 MB in size) equally divided into 2 categories: Parasitized (or infected) and Infected.

# Data Cleaning

The data we extracted isn’t ready to be implemented. So, we need to clean it. First, we define the location where we’ve kept all our data in the DATADIR variable.

* Next, we define the CATEGORIES in which we keep the ‘categories’ we need to predict. The images in the dataset are not all of the same dimension and for the model that I’m going to use to work, we need to resize all our images. So, I defined IMG\_SIZE to 100 which is for (100 \* 100 images). Feel free to change this if you’re low on memory (Try 50 or 64 if you’ve RAM <= 8GB). We also define the ‘**training\_data**’ variable where we are going to save all our data.
* Next, we go through our categories and we provide the path to our data in the ‘**path**’ variable. We create a **class\_num** variable to save the categories for our dataset, i.e., 1 for parasitized and 0 for uninfected.
* Now, we read all the images using Python’s Opencv library and then we resize the image that we talked about before. If your system’s a little bit slow, add:

img\_array = cv2.imread(img, cv2.IMREAD\_GRAYSCALE)

This converts your images into grayscale, so you get a dimension of (100 \* 100) instead of (100 \* 100 \* 3). That saves like a lot of memory and time.

* Next, we add that in our training data along with its class. We do this for all the images in the dataset and then we shuffle our training data to mix both categories.

# Saving the Data

We cleaned our data. What’s next? Since the data cleaning is a large and tedious process, we save our **training\_data** in a pickle file. But first, we need to separate our features from our labels.



Here, we define two variables X, for the features and y, for our labels.

Next, we use pickle files to store our data so that we can load them easily later.

# Training Our Model

## VGG-1 Model

Now, it’s time to create our model. Here, we’re going to use Convolutional Neural Networks to build our model.

So, first we’ll create a baseline model on which we’re going to compare our other models. This model will have only one Convolutional layer.



This is a basic CNN model which provides us an accuracy of > 74 % on our test set.

If we use a VGG-2 model, we obtain an accuracy of > 88 % whereas for VGG-3 and VGG-16, we get an accuracy of 95 %.

# Conclusion

We used Convolutional Neural Networks in this project. The thing about CNN is it performs way better if it has more data, like a minimum of 300K to 400K images. That way, we can achieve an accuracy of more than 98 %.