Hochschule für angewandte Wissenschaften
Würzburg-Schweinfurt

Bachelor’s Thesis

For the Bachelor of Engineering in Mechatronics

# Automated Cell Counting for Laboratory Automation in Life Sciences

In Cooperation with
Fraunhofer Institute for Silicate Research
Germany.

Okerulu, Chukwudi Ikechukwu
Matriculation Number: 4017032

July 30, 2021

Supervisors

Prof. Dr. Jan Hansmann
Faculty of Electrical Engineering
Automation & Process measurement

Prof. Dr.Tobias Kaupp
Faculty of Electrical Engineering
Digital Production & Robotics

Mr. Lukas Königer
Fraunhofer ISC
Bioprozesstechnik

Okerulu, Chukwudi:
*Automated Cell Counting for Laboratory Automation in Life Sciences*
Bachelor Thesis Mechatronics Engineering
Hochschule für angewandte Wissenschaften Würzburg - Schweinfurt
Thesis period: 01.03.2021 – 30.07.2021

## DECLARATION

I hereby declare that except for where specific reference is made to the work of others, the Dissertation has been written by me and that it is a record of my research work. It has not been presented in any previous application for a certification or degree. And all sources of information are expressly acknowledged by means of references.

**Okerulu, Chukwudi Ikechukwu**
**July 2021**

# ACKNOWLEDGEMENT

First of all, my profound gratitude goes to my supervisor at the University of Applied Science, Wurzburg Schweinfurt, Prof Jan Hansmann. For his confidence in me and giving me the chance to explore such an exciting field of research and providing help and advice whenever I needed it. I also want to thank my Industry supervisor, Mr. Lukas Königer, and my thesis committee Ms. Dalia Mahdy, and Mr. Marius Gensler. Your comments and industry insights; have been very beneficial in completing this manuscript; I learned a lot from you in the past six months, and I immensely enjoyed my time with you.

I want to thank Prof. I.O Okerulu and Ms. Dalia Mahdy for proofreading this text. I am very grateful.

Finally, I would like to thank my family for always being there for me, supporting me throughout my years of study, and through the process of researching and writing this thesis. Important family events were sacrificed so I could focus on completing my education.

Thank you for giving me the strength during my lowest moment; this would not have been possible without every single one of you.

# ABSTRACT

Cell counting has been a vital aspect of life sciences and biological research to date. However, the procedure is done manually and can be tedious, with Hemocytometer (Neubauer Chamber) being the most frequently used manual cell counting method. Therefore, there needs to be research carried out to address this challenge. Object detection using machine learning was identified as the procedure to achieve this aim. The Hemocytometer method provides a geometrical grid that specifies the region of interest of the count. This paper proposes using a Convolutional Neural Network-based algorithm to automate cell counting and compare it with the conventional method. The Faster R-CNN model was identified from the literature review as the most suitable and efficient algorithm for detecting cells. This proposed technique employs deep learning and image processing to isolate the chamber grid patterns, recognize each region of interest, and perform the count. It was also necessary to evaluate the results, establishing the accuracy of the proposed method, and creating a Graphical User Interface (GUI) for the best cell counting model for optimal use of the technique in a laboratory. A series of experiments were conducted, and the result shows a accuracy of 95% based on the dataset and result of all experiments. The tendency of overfitting was mitigated with an early stoppage.

# Table of Contents

# Nomenclature

## Acronyms / Abbreviations

| | |
|---|---|
| ANN | Artificial Neural Network |
| AP | Average Precision |
| BN | Batch Normalization |
| CM | Convolutional Max pooling |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| FPN | Feature Pyramid Network |
| FPS | Frames Per Second |
| GPU | Graphics Processing Unit |
| GUI | Graphical User Interface |
| IoU | Intersection over Union |
| mAP | Mean Average Precision |
| ReLU | Rectified Linear Unit |
| RCNN | Regional Convolutional Neural Network |
| SSD | Single Shot Detector |
| YOLO | You Only Look Once |

# List of Figures

# List of Tables

# 1. Introduction

## 1.0 Introduction

The artificial regeneration and generation of human and animal cells, tissues, and organs to study, replace, or restore their functionality has been the frontier of medical research for the past decade. This field, known as regenerative medicine, holds the capacity to engineer damaged tissues and organs by stimulating the body's ability to repair tissues or organs. Regenerative medicine also includes growing the said tissues and organs to be implanted in the laboratory through the imitation of nature. Tissue engineering has the potential to confront the transplantation crisis caused by the shortage of donor tissues and organs and to address other important but unmet patient needs(1).

An underlying premise is exploiting our understanding of the basic biology of cells, tissues, and organs, including their development and functionality. We can develop methods for controlling these biological processes and based on this ability; strategies can finally be developed for the engineering of living cells or fostering cell repair or regeneration.

The starting point of regenerative medicine is the consideration of cells since it is the basic structural, functional, and biological unit of all tissues and organs. Inevitably, most challenges in regenerative medicine are also faced in cell technology. These challenges include cell sourcing, culturing, and manipulation. Only when we have met these challenges can we address the issue of tissues and organ engineering.

The challenge of cell culturing is tackled with the fact that cells are the basic building blocks of living organisms because they can survive in isolation. This process generally refers to the isolation of cells from an animal or plant and their subsequent growth in a favorable environment composed of necessary nutrients, ideal temperature, pH, and humidity to allow the growth and proliferation of cells. Cell culturing importance in cellular and molecular biology cannot be overstated. It provides a platform for studying the physiology and biochemistry of cells which offers a representative understanding of what is happening within the tissue or organ of study. This makes cell culture significantly important in drug screening and development, vaccine development, and diagnosis of conditions and diseases. Since the growth or decay rate of the cells in any environment, natural or artificial, makes available valuable insights into the behavior of the cell in question, the need to ascertain the growth rate introduced the concept of cell counting, a method of quantification of cells in life sciences.

Some known cell counting methods rely on electronic appliances, and some are primitive and do not require any special equipment. Examples of these methods are Manual Cell Counting, Automated Cell Counting, and Indirect Cell Counting.

Usually, scientists' resort to manual cell counting by utilizing the Neubauer Chamber, otherwise known as a Hemocytometer, and microscopes. The Neubauer chamber encloses a section of the solution in a grid of well-known dimensions and, with the help of a microscope which procures pictures of the enclosed grid as shown in the figures below. By counting the cells in each box and the number of boxes or squares used in the cell count (Fig. 1), the concentration per milliliter of solution can be estimated.

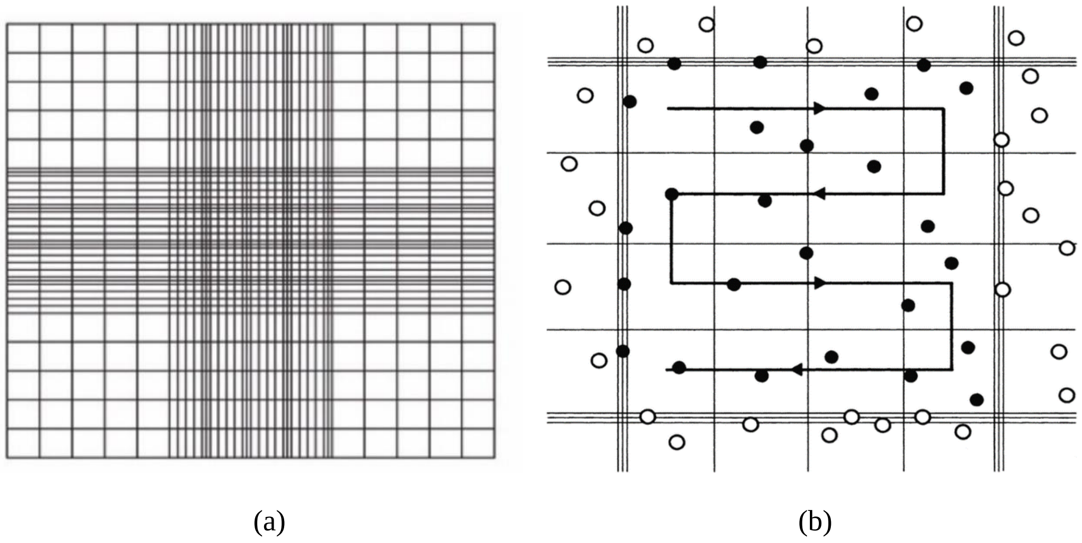


*Figure 1: The Neubauer chamber grid (a) and Cell counting using a Neubauer Chamber (b)*(2)

This method can be tedious and requires proper training and experience to use. Even with experience, it is still error prone as the environment, and the physical capacity, eyesight, stress level of the scientist in question can affect the count, making the procedure mostly an estimate.

Therefore, the importance of creating a fast and reliable method of performing cell counting cannot be exaggerated. Recently, with the advances of Machine Learning and object detection, new innovative ways of cell counting favor the use of Deep Neural Network methods due to their adaptability in various Research areas. This method is also not without drawbacks, but the advantages it offers outweigh the question of automation taking away jobs from people. Instead, it enables scientists to focus more on the necessary work of solving problems in life sciences.

Object detection and localization have seen much progress in the machine learning community. For example, Support vector machine (SVM), Region-based Convolutional Neural Networks (R-CNN)(3), Fully Convolutional Networks (FCN)(4), and You Only Look Once (YOLO)(5) have become the state-of-the-art algorithms for the object detection problem. Though there are many algorithms, it is important to choose a suitable algorithm.

## 1.1 Problem Statement

Cell counting has been a vital aspect of life sciences and biological Research to date. However, the procedure is done manually and can be tedious, with Hemocytometer (Neubauer Chamber) being the most frequently used manual cell counting method. This method provides a geometrical grid that specifies the region of interest on a static feed for the count. Therefore, with the rise in Machine learning studies and its ability to emulate the brain capacity to process information, capturing structures and patterns, there needs to be Research carried out to evaluate the state-of-the-art deep learning models and identify the best model for the detection of cells in a static or dynamic feed. Some of the popular deep learning models for object detections are Faster R-CNN, YOLO, and VGG16. These models differ in terms of their architecture and performance; therefore, it is important to identify the suitable deep learning model for the desired purpose.

## 1.2 Aim and Objectives

This thesis aims to automate the Manual Cell counting method Hemocytometer also known as Neubauer Chamber, using deep learning models. And the appraisal of the model's performance.

The Following objectives have been identified to attain the aim of this thesis work:

- Identify suitable and highly efficient deep learning models for the detection/recognition of cells (Living cells and dead cells).
- Evaluate and compare the classification performance of the selected deep learning models and the conventional method of cell counting and present the results.
- Creating a Graphical User Interface (GUI) for the deep learning method, which gives the best cell counting benchmark for optimal use in a laboratory.
- Integrating the software with the Robotics cell culturing system

# 2. Theoretical Background and Related Work

## 2.1 Cell Culturing

Cell culture is generally regarded as a technique by which cells are cultivated outside a living organism under controlled conditions (e.g., temperature, pH, nutrient, and waste levels) (6). Today's most common cell culture practice is to grow cells using multi-well microplates or Petri dishes as culture vessels. Cell culture systems are usually designed according to the specific requirements of target cells and application, but the process follows a general procedure. There are two procedures for obtaining cells: from a cell bank or by isolating cells from a given tissue. When starting culture from cells obtained from a cell bank, one needs to go through the procedures of "thawing", "expansion", "cell harvesting", and "cell observation. When using tissue collected from a donor, unnecessary tissues are usually removed if it is attached. There are two most important methods to isolate cells from the tissue, explant culture and enzymatic method. In enzymatic methods, isolation of cells from the tissue of interest is done using a proteolytic enzyme solution(7). Then proceed with the steps of "cell expansion", "cell harvesting", and "cell observation". During the cell observation, the cell vessel is viewed using an optical microscope or other observation devices to check the viable cells, presence of foreign objects, and cell number. The cell counts are performed on the cultured cells to determine the exact number of viable cells in each cell suspension.

## 2.2 Cell Counting Process

No Matter the method used to culture or separate the cells, the number of cells recovered in each fraction needs to be counted, and their viability assessed. The Cell counting process is an essential practice in the study of eukaryotic cells for different purposes, such as the management of cell cultures in biological Research, the titration of cell populations in diagnostics, and in-process controls in industrial bioprocesses(8).

There are quite a few techniques devoted to cell counting with varying degrees of complexity. They can be classified as Direct cell counting and Indirect cell counting, which usually involve manual and automated cell counting.

**Manual cell counting:** Among the well-known methods developed so far for cell counting, manual counting with a hemocytometer has been the generally used method owing to its adaptability and low cost. The hemocytometer, or more accurately the 'cell counting chamber', is perhaps the simplest method of counting cells. The apparatus consists of a microscope slide with a grid specifically devoted to cell counting, which is thicker than normal with a precision ground surface that contains a precise volume of liquid when covered with a precision ground coverslip (Fig. 2). Scribed onto the surface are a series of lines that form a grid, as shown in (Fig. 2). The grid (Neubauer type) usually consists of 9 large squares, each 1 x 1 mm holding a volume of 0.1 mm$^3$. Each large square is subdivided into 16 smaller squares. The 16 squares of the large center square are further divided into 16 smaller squares(9).

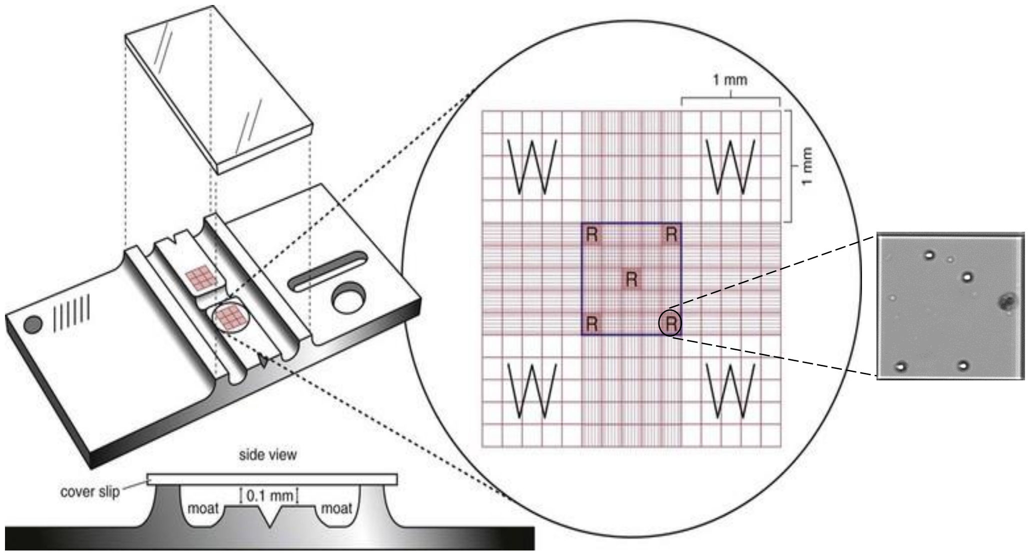


*Figure 2:The View of the Neubauer chamber grid.*

Newtons' rings are visible with the coverslip in place, and cells in suspension are drawn under the coverslip by capillary action. Contained within each large square of the grid is 0.1 mm$^3$ of cell suspension. By viewing the slide under a standard light microscope, both the lines and the cells within the grid can be seen (9). Because of the errors involved when counting cells on a hemocytometer, dilution of the original suspension may have to be carried out to obtain a cell density of about 5 x 10$^5$ - 2 x l0$^6$ cells/ml. The number of cells counted can then be multiplied by the appropriate dilution factor (9). The formula below is used to compute the number of cells present further.

$$The\ Average\ number\ of\ Cells\ = \frac{Total\ No\ of\ cells\ in\ each\ Counted\ box}{No\ of\ Boxes\ Counted} \qquad (2.0)$$

$$Viability = \frac{Living\ Cells}{Living\ Cells\ + Dead\ Cells} \qquad (2.1)$$

$$Cell\ Concentration = Average\ number\ of\ Cells\ *\ DilutionFactor\ *\ 10^4 \qquad (2.3)$$

This approach suffers from several drawbacks. Firstly, the experiment is considered subjective due to inter-observer variability (two different analysts can count differently at different times) (10) (11) (12). Secondly, it is a repetitive and tedious task, which increases the risk of making mistakes.

**Automated cell counting with dedicated machines:** Flow cytometers and Coulter counter are machines for automated cell counting. They are considered the best method for cell counting because they do not suffer from the issue of manual counting. However, they are often very expensive to procure.

**Automated cell counting by image analysis:** This particular technique is the focus of this thesis, as it combines the power of computers and major breakthroughs in image processing and machine learning to count cells. It can be categorized as a computer-aided process.

## 2.3 Artificial Intelligence

Machine Learning, a subset of Artificial Intelligence (AI), is the science and art of programming computers so they can learn from data (5). The field was born in 1956 at Dartmouth college by John McCarthy(13). In recent years, due to developments, the definition progressed thanks to Arthur Samuel, who coined the term “Machine learning” in his studies. Arthur Samuel defined machine learning as the ability of computers to learn without explicit programming(14). Various computer programs which relied on explicit programming by humans are now programmed with little or no human intervention. With the new generation of graphic cards, this technique has launched a breakthrough in different fields involving data processing.

### Types of Machine Learning Systems

Learning is a process of acquiring new behavior, value, skills, and understanding through a specific process and channel. This process in Machine Learning systems can be classified according to the type of supervision required during training.

- Supervised Learning
- Semi supervised Learning
- Unsupervised Learning
- Reinforcement Learning

## Supervised Learning

Supervised Learning, the most basic classification of machine learning algorithms, requires direct supervision, as the name suggests. In this type of learning, the training set you feed to the algorithm includes the desired solutions, called labels (5). These labels annotated by humans hold the classes and location of the objects of interest. A typical supervised learning task is classification; annotated pictures of dogs and cats are fed to the algorithms, which learn from the data set and predict the annotations of new data not previously seen. Some of the most significant supervised learning algorithms are:

- Decision Trees
- Linear Regression
- Logistic Regression
- Support Vector Machine (SVMs)
- Random Forest
- K-Nearest Neighbors
- Neural Networks

## Unsupervised Learning

In unsupervised learning, the training data is unlabeled, forcing the algorithms to learn and identify the properties of a class without human intervention. The system tries to learn with little or no guidance (5). Some of the most important algorithms used here are:

- Clustering (K- Means and Hierarchical Cluster Analysis)
- Anomaly detection and Novelty detection
- Visualization and dimensionality reduction.

## Semi supervised Learning

Since labeling data is usually time-consuming, this learning method uses algorithms that can deal with partially labeled data. Most semi supervised learning algorithms are combinations of unsupervised and supervised algorithms (5).

**Reinforcement Learning**

This type of learning trains the network continually using trial and error; the machine learns from experience based on a reward and penalty system. It must then learn by itself the best strategy to get the most reward over time (5).

## 2.4 Artificial Neural Network

Nature has inspired countless of our inventions. A look into the brain's architecture and its learning process inspired the creation of intelligent machines, which launched in artificial neural networks (ANN). A computational model inspired by the biological neurons found in our brains. Every biological neuron connects with the neighboring neurons and communicates with each other. The architecture of biological neural networks is still a subject of active Research, but some parts of the brain have been mapped. And it seems that neurons are often organized in consecutive layers (5). Artificial neural networks (Fig. 3) try to mirror the working of biological neural networks but like planes that were inspired by birds but did not have the whole attributes of a bird. Similarly, Artificial Neural Network has its difference when compared with their biological counterpart. An artificial neural network consists of a given set of neurons connected to each other and is grouped into layers to replicate the neural function of our brain (15).

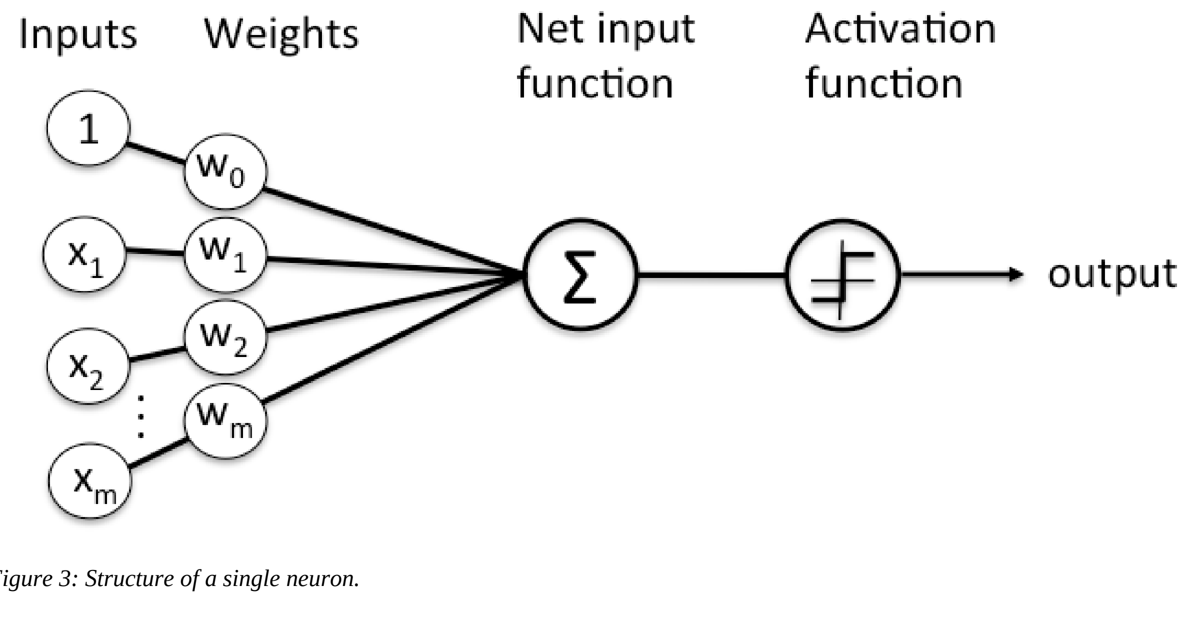


*Figure 3: Structure of a single neuron.*

## Back propagation

Though the artificial neural network has been in existence since the 1940s, they become popular due to the inception of backpropagation by Rumelhart et al.(16). Backpropagation is the calculation of the gradient that proceeds backward through the network, with the weight of the last layers being calculated first, the second to the last, and so on. With the backpropagation, a network is given with an error function, and it calculates the gradient of the error function with respect to the weight of the neural network. The learning process of a neural network can be summarized as the search process of a set of weights w that eventually leads to a decrease in loss function $L$ ($\boldsymbol{H}$, w) for a set of input-output pair $\boldsymbol{H}$**.** The loss function calculates the difference between the predicted output $a_i$ and its actual value $A_i$. Mean squared error is a classic error function for back-propagation.

$$L(\boldsymbol{H}, w) = \frac{1}{2N} \sum_{i=1}^{N} (ai(H, w) - Ai)^2 \tag{2.4}$$

## Activation Function

The activation function is a mathematical equation that outlines the output of a perceptron given its input. In most applications, this function is non-linear because if it is, the output to the neural network would be a linear function which would only be suitable for linear classification/regression problems (5). Moreover, most of the time, neural networks want to compute something more complicated than that. This is particularly significant in Deep Learning approaches when the goal is to make sense of complex images with high dimensionality. An important feature that needs to be considered is that it must be differentiable to perform back-propagation optimization for gradient error calculations. There are several activation functions, but the use of each of them depends heavily on the goal of each layer in the ANN as they have different properties. Some common activation functions are Sigmoid, Tanh, ReLU, and SoftMax.

## Batch Normalization

Batch Normalization is a technique that organizes the update of multiple layers in the model by rescaling the data to put all the data points on one scale. It is used to stabilize the learning process and radically reduce the training epochs required (17). An unnormalized data can cause problems to the network, making it drastically harder to train and decrease the training speed. This technique was developed by Sergey Ioffe and Christian Szegedy. It consists of adding an operation that zero-centers and normalizes each input before and after the activation function of each layer.

## 2.5 Convolutional Neural Networks

Several types of artificial neural networks are considered important: Feed-forward neural network, recurrent neural network, convolutional neural network, Etc. Among these listed types of networks, the convolutional neural network, a special kind of Feed-Forward neural network has found application in image/video processing and natural language processing, where it is proven to be better than many previously established benchmarks (18). The Convolutional neural network was modeled after the functionality of the visual cortex of the brain, which processes visual information (19).

Convolutional neural network (CNN) is named after the mathematical operation convolution: The mathematical operation on two given functions produces a third function in a continuous or discrete domain.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(x).\ g(t-x)dx \qquad (2.5)$$

In image processing, the function *f* is the intensity of a given pixel, and the *g* is a weighing function also known as a kernel and depending on the value of the kernel, different image manipulation operations, for example, blurring effect and intensity, can be applied on the image (Fig. 4).

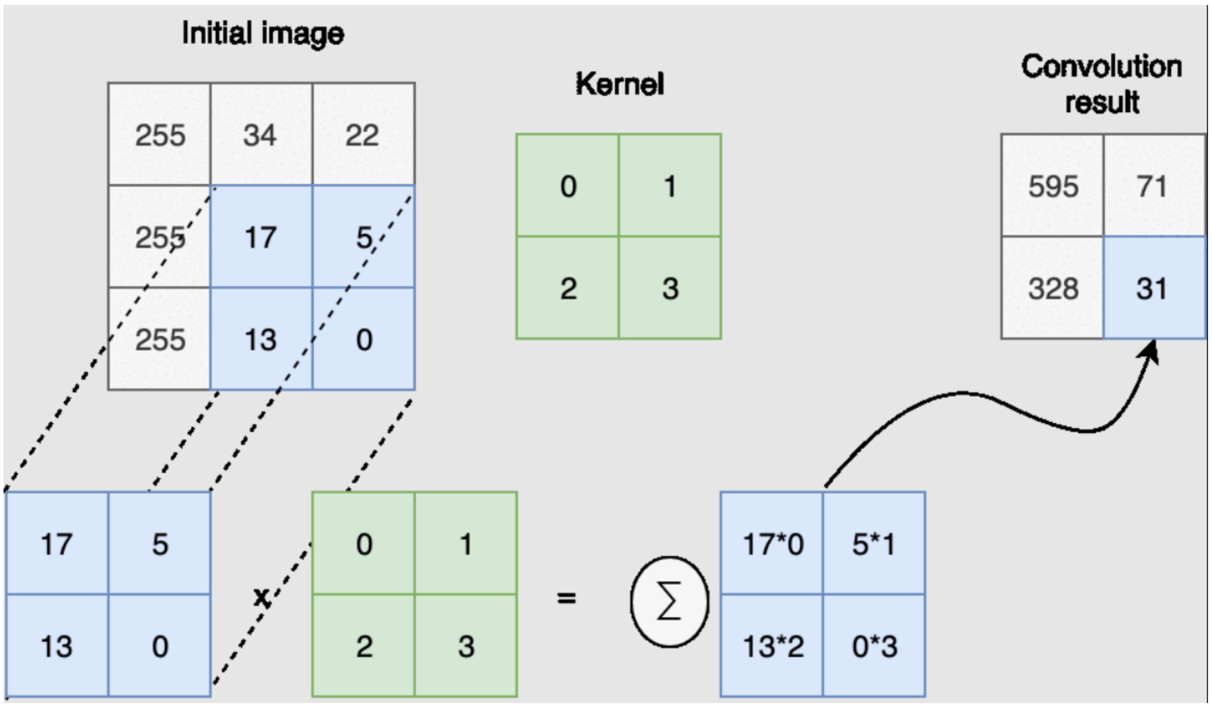


*Figure 4: Convolution Process.*

**The Convolutional Layer**

The core building block of the Convolutional neural network is the convolutional layer. This layer consists of a set of filters with learnable parameters used to extract features from the input data. These parameters are known as the weights and biases of the network. The layers are built up ideally to first detect a set of patterns in the input image, such as edges, and the second layer detects nested patterns and so on (19). The Convolutional layers method of learning features is like that of a multi-layer perceptron network, also known as Artificial Neural Network (ANN), which learns through backpropagation.

The convolution of an image is done by sliding a kernel or filter with a fixed size over the matrix value of the input image. The elements are combined through matrix multiplication of the kernel and the area it overlaps in the input image. Other parameters like the zero paddings, which add zeros to the outer part of the input image, and stride, which determines how the filters slide between steps, also exist.

**Why Convolutional Neural Network**

Rumelhart et al. (16) stated that the artificial neural network has been in existence for a while, and there are reasons why Convolutional neural networks have become the go-to network for object detection and classification.

- **Convolutional Neural Networks have few memory requirements:** Most neural networks do not scale well for multi-channel images. A typical input of 100 x 100 colored images would produce 100 * 100 * 3 = 30,000 weights. CNN takes advantage of this and reduces the dimensionality of any input while preserving its features.

- **They are easier and better to train:** Because of CNN’s low architectural complexity, the time spent on training is significantly decreased compared with other neural networks. And because of the lower number of parameters, noises introduced to the model are significantly low, which leads to a high-performance network.

## 2.6 Machine Learning Process

There are many aspects to consider while trying to solve a machine learning problem (20). With the dataset being the most crucial aspect, there is a need to gather and clean the dataset, paying close attention to the possibility of biases that can easily be part of the data. As the solution is mostly dependent on the data at hand, the data should also be observed for possible outliers. Depending on the problem and data available, a suitable machine learning algorithm can be chosen or, in the case of implementing from scratch, a general understanding of the work of specific networks that makes up an algorithm. During the training of the network, however, different problems like over representing a class, where the model finds it difficult recognizing a scare class or overfitting, which means the algorithm successfully memorized the data it was trained on rather than learned to generalize from it properly could arise. Adding more data or using data augmentation always helps to mitigate overfitting, and it is an important step in preparing a dataset for training (21) (22). It is good practice to split the dataset into the training set, validation set, and testing set to help with the model's accuracy (23). The training dataset are used to train the model, while the test and the validation set is used to evaluate performance and test hyperparameter optimization. Hyperparameters are properties of the algorithm that influences the speed and result of the learning process. The two most important hyperparameters are the learning rate and the decay. Transfer learning which involves using an object detection solution in another domain can also be used to speed up the learning, improve the accuracy (24) and overall performance of a model.

Machine learning generally relies on other libraries; the five major dependencies are listed below.

- **TensorFlow:**

  TensorFlow is a software framework for numerical computations based on data flow graphs. The open-source software is figurative a math library and was designed primarily for machine learning applications such as neural networks. The name TensorFlow is derived from the operations the neural networks perform on multidimensional data arrays known as tensors. It was released in 2015 under the Apache License 2.0 by Google for both Research and production. C++ programming language is the core of TensorFlow, and it has Python as the most developed frontend interface for expressing and implementing computation graphs. Tensorflow can run on multiple CPUs and GPU. It can be deployed on various platforms such as 64-bit Linux, macOS, Windows, Raspberry Pi single-board computers, and mobile computing platforms including Android and iOS.

- **NumPy:**

  NumPy is a Python programming language open source project to enable numerical computing. It was released in 2015 under the BSD license; it is fast, versatile, and offers comprehensive high-level mathematical functions.

- **OpenCV:**

  OpenCV is an open-source library with numerous algorithms primarily aimed at real-time computer vision. The package in the library includes the following modules: Video Analysis, Object Detection, Image Processing, Etc. The library is a cross-platform library under the open-source BSD license, with MATLAB, Python, C++, and Java interfaces, supporting Linux, Windows, Android, and macOS.

- **Pillow:**

  Python Imaging Library is a Python programming language library that adds image processing capabilities to the Python interpreter by opening, editing, and saving different formats of image files. It also has support across different operating systems like Windows, macOS, and Linux.

- **Matplotlib:**

  Matplotlib is a broad Python programming language plotting library for creating static, dynamic, and interactive visualizations. It also provides an API to use general-purpose Graphical User Interface toolkits such as Tkinter, PyQt, or wxPython to embed plots.

## 2.7 Computer Vision

There is a need to identify objects in various fields and track them efficiently, making computer vision an important task and a core problem in machine learning. The techniques involved, however, depend mainly on the application domain ranging from the analog implementation using the human vision to the digital system that can emulate the human visual system and sometimes surpass it (25). In recent years, the development of neural networks and deep learning has contributed to computer vision advances, achieving superior performance for various tasks such as object detection, image classification, and semantic segmentation (4) (26) (22). Image classification (Fig. 5 (a)) seeks to recognize categories of objects in an image. On the other hand, object detection not only identifies objects but also predicts the location of each object using a bounding box (Fig. 5 (b)). In contrast, segmentation aims to predict the pixel classifiers to assign to a specific category providing a better understanding of the image (Fig. 5 (c) (d)). Another method exists which combines object detection and segmentation to give rise to instance segmentation. These methods are quite different from conventional image processing, which mainly manipulates or enhances pixels and data on the image.

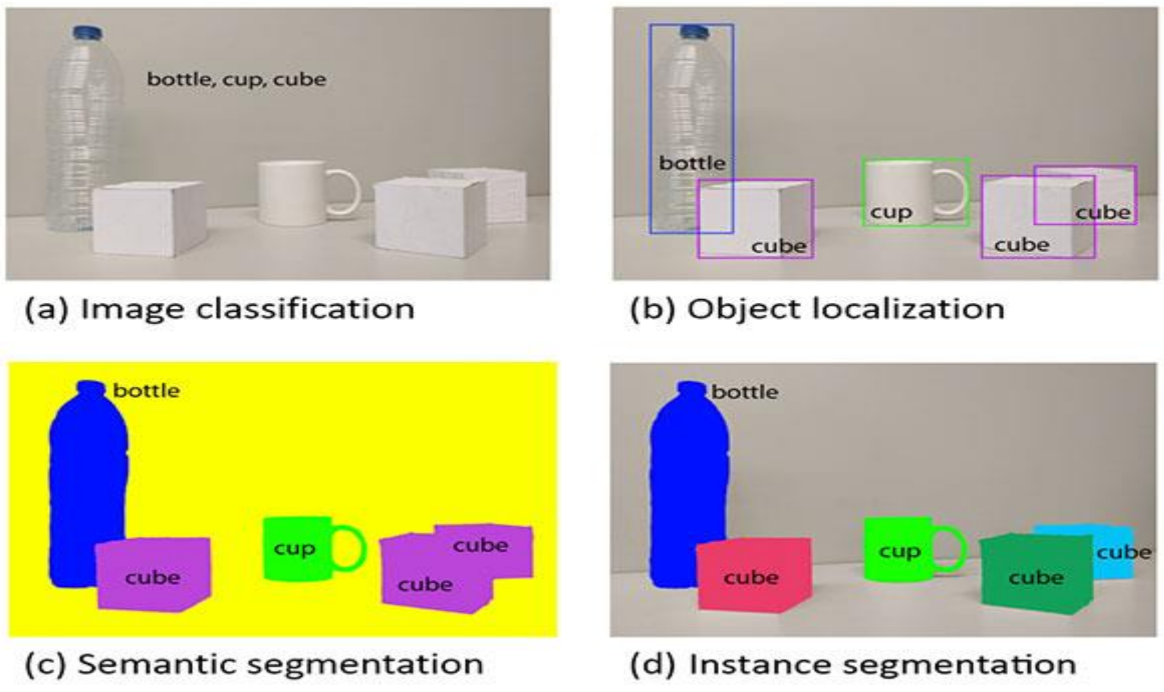


*Figure 5: The Difference between the output of image classifiers. (a) Image classification, (b) Object localization, (c) Semantic segmentation and (d) instance segmentation.*

## 2.8 Object Detection

The task of recognizing and localizing multiple objects in an image is called object detection. Recently, object detection has attracted a significant amount of attention due to its wide range of applications, with a complicated yet understandable approach. However, years ago, the simple and common approach was to take a Convolutional Neural Network trained to classify and localize a single object and slide it across the image of interest (5). This technique is relatively straightforward, but it requires sliding the Convolutional Neural Network many times along different regions so that it can be quite slow. The Fully Convolutional Neural Network introduced by Jonathan Long et al. (27) presented a much faster way to slide the Convolutional Neural Network across an image. Further developments in deep learning and graphical processing unit (GPU) computing power brought about different state-of-the-art algorithms for object detection.

## 2.8.2 State of the Art Object Detection

There has been tremendous Research on object detection using deep learning, mostly on different ways to use various architectures of Convolutional Neural Networks. These state-of-the-art detectors are designed to achieve high accuracy, are domain-specific, and can be categorized into two-stage detectors and one-stage detectors. The modern one-stage detectors usually implement a fully convolutional architecture, and the network's outputs are classification box offsets and probabilities at each position on the image. The One-stage detectors perform classification and regression on dense anchor boxes without generating a sparse Region of Interest set (28). While two-stage detectors adopted a more complicated pipeline that uses the Region proposal network, filters out the regions with a high probability of containing the object of interest. In these detectors, sparse region proposals are generated in the first stage and are further regressed and classified in the second stage (28).

### 2.8.2.1 One-stage Detectors

**Region Convolutional Neural Network (R-CNN)**

R-CNN, as proposed by Girshick et al. (3), is a region-based CNN detector. The authors work was the first to show that CNN can generate high object detection performance. The RCNN detector consists of four modules (Fig. 6): the first module generates category-independent region proposals, the second module extracts a vector of features of fixed length from each region proposal, the third module is a set of class-specific linear support vector machines used for classifying objects in an image, the final module is a bounding-box regressor for bounding box prediction. To obtain details, the authors' first chose the method of selective search to generate region proposals.

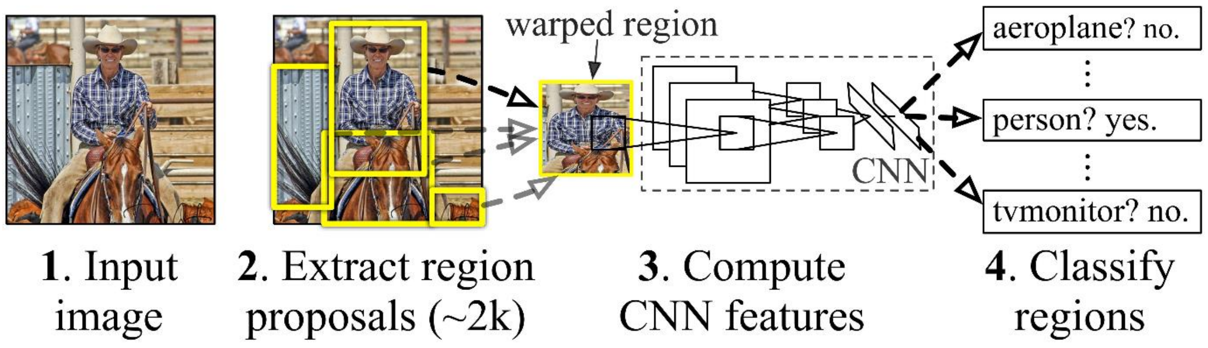


*Figure 6: Structure of an RCNN model*

A Convolutional Neural Network is then used to extract a 4096-dimensional feature vector from each proposed region (29). As illustrated in (Fig. 6) taken from the original paper.

**Fast Region Convolutional Neural Network (Fast R-CNN)**

To create architecture more efficient than R-CNN, Ross Girshick proposed a newer version (30). Fast RCNN extracts feature from an input image and then iterates through the Region of Interest (ROI) pooling layer to get the fixed size features as the input for the fully connected bounding box regression layer and classification. Features are extracted once more from the full image and simultaneously sent to CNN for classification and localization. Compared to the R-CNN, Faster R-CNN saves time on CNN processing and disk space to store many features (30). The architecture is illustrated in (Fig. 7)

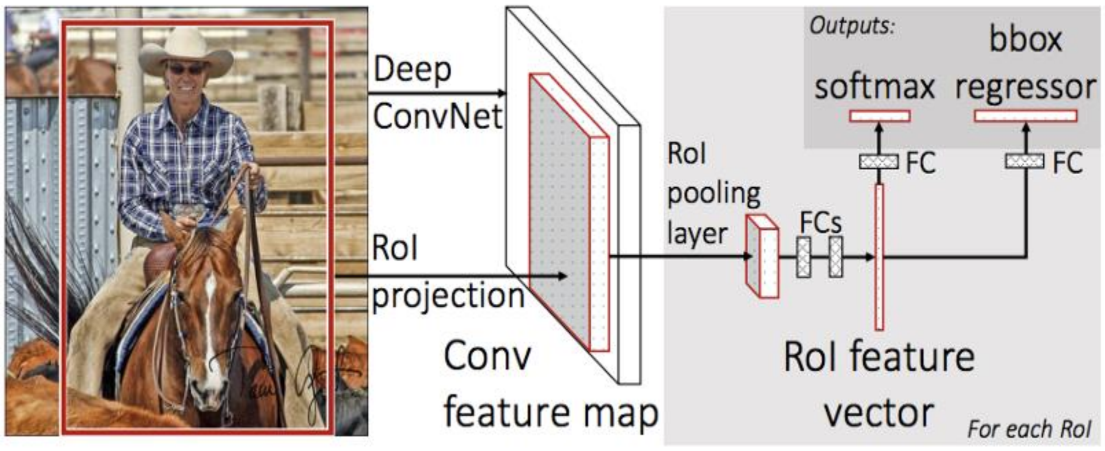


*Figure 7: The Structure of a Fast R-CNN model*

**Faster Region Convolutional Neural Network (Faster R-CNN)**

Like the work done on Fast RCNN, Girshick et al. presented an architecture that uses VGG16 (31). In comparison, Fast RCNN uses selective search, which is slow and requires the same execution time as the detection network. Faster RCNN replaces it with a novel RPN (Region Proposal Network), a fully convolutional network for efficient forecasting of region proposals with a wide range of scales and aspect ratios. RPN increases the creating speed of region proposals because it shares full-image convolutional features and a common set of convolutional layers with the detection network (31). (Fig. 8) illustrates this architecture.

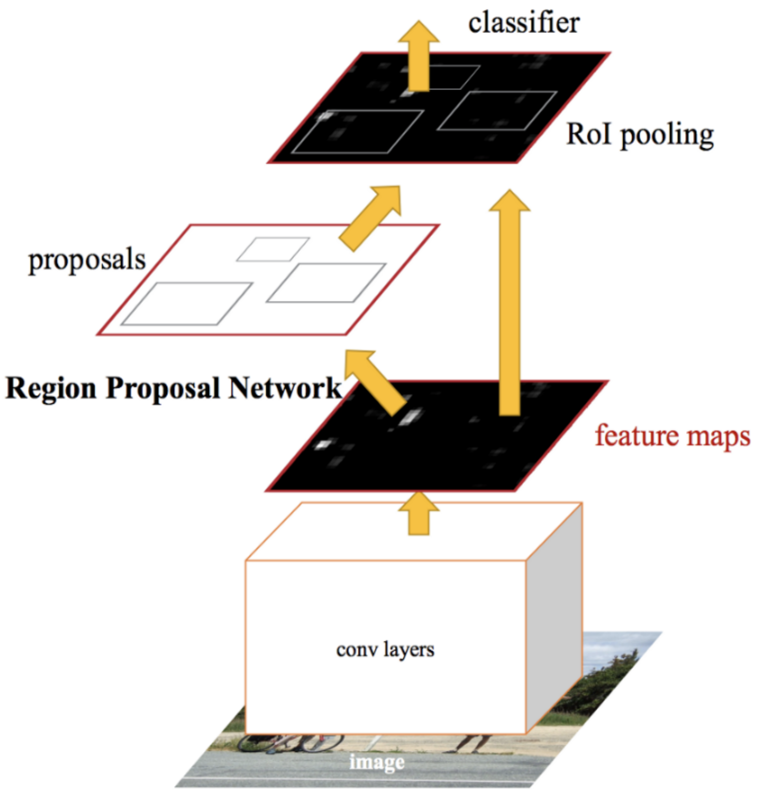


*Figure 8: The Structure of a Faster R-CNN model*

**Mask Region Convolutional Neural Network (Mask RCNN)**

The Mask R-CNN is an extending work to Faster R-CNN in instance segmentation. Although it added a parallel Mask branch, it can be seen as a more accurate object detector. The authors used the Faster R-CNN with a different basic feature extractor known as ResNet (26) to achieve excellent accuracy and processing speed.

## 2.8.3 Two-stage Detectors

### You Only Look Once (YOLO)

Redmon et al. proposed a one-stage object detector. The authors presented the architecture with the major impact of real-time detection on images and webcam at 45 frames per second (32). YOLO framed detection as a regression problem; the architecture extracts feature from input images to predict class probabilities and bounding boxes. The pipeline first divides the input images into an S * S grid; within each grid, an n bounding box is taken, and the confidence score is acquired by multiplying two parts object and intersection over the union. For each of the bounding boxes, the network produces a class probability and offset values for the bounding box. The feature extraction network contains twenty-four [24] convolutional layers followed by two fully connected layers (32). The authors of the YOLO paper presented different versions of YOLO from version two to the recent version five, with each version improving on the performance of the previous (33). Some of the changes were made in the implementation of batch normalization, use of anchor boxes, among others. The YOLO architecture is illustrated in (Fig. 9).

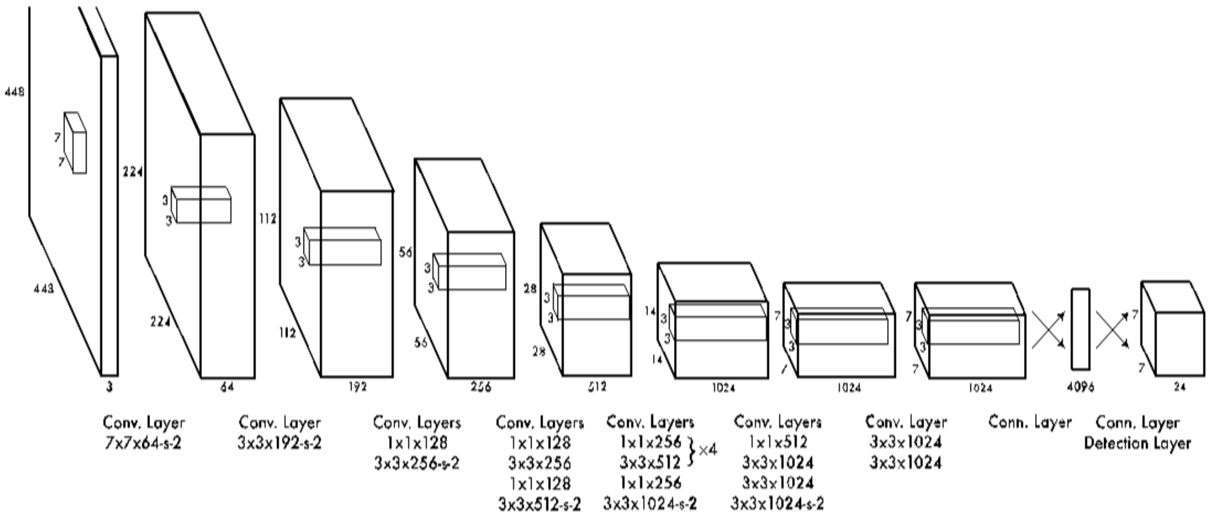

448
448
3
224
224
64
112
112
192
56
56
256
28
28
512
14
14
1024
1024
1024
4096
24
Conv. Layer 7x7x64-s-2
Conv. Layer 3x3x192-s-2
Conv. Layers 1x1x128 3x3x256-s-2
Conv. Layers 1x1x128 3x3x256 1x1x128 3x3x512-s-2
Conv. Layers 1x1x256 3x3x512 }x4 1x1x256 3x3x1024-s-2
Conv. Layers 1x1x512 3x3x1024 3x3x1024 3x3x1024-s-2
Conv. Layer 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer Detection Layer

*Figure 9: The Architecture of YOLO*

## Single Shot MultiBox Detector (SSD)

The authors of the architecture presented a work like YOLO (32) both in the architecture and performance, but with some differences. A Single-shot detector directly predicts category scores and box offsets for a fixed set of bounding boxes at each location in the feature maps with different scales. The SSD also relies heavily on data argumentation which includes random color distortion, cropping, horizontal flipping, and expansion (34). The architecture is illustrated in (Fig. 10).

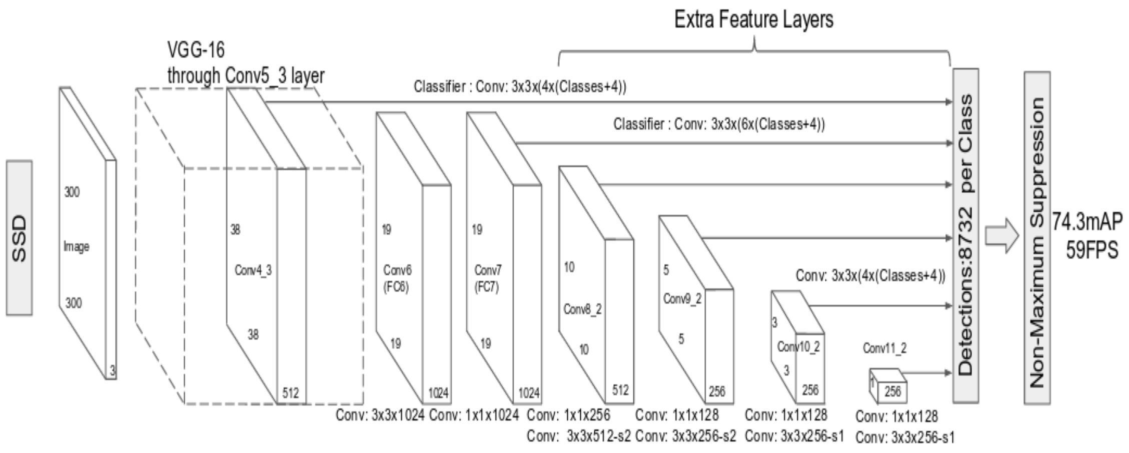


*Figure 10: The Architecture of the SSD model*

## 2.9 Related Work

Adagale SS et al. implemented a simplified Image Segmentation using a combination of Pulse Coupled Neural Network and Template Matching for Blood Cell Counting (35). The steps were dependent on features such as cell shape and size. A template matching algorithm is a powerful tool for image segmentation where a template is generated from the anatomy of the object. This approach is like image classification, except that they are implemented in the spatial domain of the image rather than in a feature space. However, the accuracy of the algorithm decreases whenever there is overlapping.

Ivan V. Grishagin implemented automated cell counting using the ImageJ toolbox (36). The Research mentioned that the standalone cell counting plugins for ImageJ lack focus on cell counting for cell manipulation, batch processing, and manual adjustment of counting parameters. The Researchers went further to use the ImageJ platform for developing an image processing algorithm. The method determines and counts the local intensity maxima in an image by first blurring the image using a filter, thereby eliminating the imperfections of high intensity, and applying a noise threshold that separates the background from the particles of interest, which are then counted using the ImageJ analysis algorithm after converting the RGB image to a grayscale image. A restriction of size and shape of the cell was not imposed on the algorithm as cells tend to coalesce at higher concentrations.

The author points towards the robustness of this approach as it took 2 to 10 times less time than manual counting and yields the same result.

Hernández Carlos Xavier et al. presented a method of using Deep Learning for segmentation and counting within Microscopy Data (37). The authors showed that it was possible to train Convolutional Neural Network architecture to count cells in microscopy images and achieve good accuracy. However, their work recorded some failure cases because of high cell overlaps, irregular cell shapes, and bad focal planes. They suggested that an additional dataset can improve the model’s performance.

Qian Liu et al. developed a system for Automated Counting of cancer cells by Ensembling deep features (38). The writers treated the cell counting challenge as a regression problem using image features extracted by deep learning models. The method involved the development of three deep convolutional neural networks to regress image features. The model showed better performance and true cell counts showing the performance and stability of the Convolutional neural network in cell counting.

Lillian Pride and Shinsuke Agehara set out to highlight the useful Image-Based techniques for manual and automatic counting using ImageJ in a Horticultural Research (39). The authors introduced the ImageJ, but unlike Grishagin, who used the possibility of custom plugins in the open-source image processing program to his advantage and created a plugin, they outlined the advantages and disadvantages of image-based counting and proceeded to use the software to the best of their ability to the desired result.

From the above papers and the knowledge gathered from further Research, it can be said that deep learning with Convolutional Neural Networks is the best-suited algorithm for object detection. Therefore, this thesis will be using a Convolutional Neural Network-based algorithm to recognize cells and evaluate the performance of the chosen algorithm.

# 3. Materials and Methods

## 3.1 Methods

The research focuses on a multimethodological approach that includes the system development method of Nunamaker and Chen (40) and experiments. The experimental method was chosen as the research method to answer the question of the performance of the selected deep learning models as an alternative to the conventional Neubauer chamber and to compare the results. I chose this research method because it has proven to have high efficiency and performance when dealing with quantitative data. Since the primary goal of the experiment is to evaluate the deep learning models for object recognition and tracking of cells in static images, the combination of the experimental methods and literature review is sufficient to prove or disprove a particular hypothesis (40).

To answer the research questions mentioned in the introduction: Identifying suitable deep learning models for cell detection and benchmarking the performance of such models, it is necessary to create several models based on different neural networks to compare the performance. Time plays a significant role in this experiment, and it limited the extent to which I can study the processes. There is a different architecture based on other frameworks. However, I can shorten the time required to examine them individually by studying Zhao Z and co (41) dissertation. I will conduct a series of experiments to establish best practices for dataset labeling and the machine learning process to improve performance. These experiments test the models with the same data to measure the accuracy using the **Maximum a Posteriori** (mAP) metric within a threshold.

## 3.2 Research Process

A research process involves understanding the research domain, asking questions, and applying valid Research methodologies to address those questions (40). Cell detection research is sufficiently broad, but we aim to narrow it down to the application of computer vision, and this allows the use of limited methodologies. The System development research process offers a way to get this task done as it interacts with other research methodologies to form integrated research. (Fig. 11) details the principal parts of system development research.

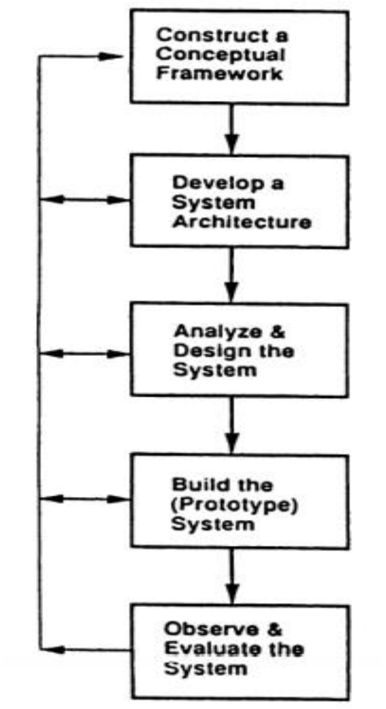

Construct a Conceptual Framework
Develop a System Architecture
Analyze & Design the System
Build the (Prototype) System
Observe & Evaluate the System

*Figure 11: A Process for Systems Development Research*

### Construction of a conceptual framework

As discussed in the earlier sections, the main goal of this research is to automate a cell counting process, and a literature review pointed to the areas of interest for investigation in this thesis. The following research questions make up the framework:

1. What is the most suitable and efficient computer vision process for object detection of cells?

   Since several deep learning models can perform object detection, the research is oriented to identify the best-suited model.

2. How do the models from the computer vision process perform against the Neubauer Chamber?

   The research also evaluates the classification performance of the selected models using relevant metrics.

**Develop a System Architecture**

A system architecture provides a road map for the object detection process. These architecture requirements include the necessary software libraries and tools. The machine learning process can be outlined as follows:

1. Data acquisition
2. Data preprocessing
3. Data Labelling
4. Splitting dataset into training, testing, and validation.
5. Parameter optimization and configuration
6. Transfer Learning
7. Training
8. Evaluation

**Analyze and design the system**

The blueprint for the design of the object detection system consists of an image from the microscope passed into the object detection model, which makes the bounding boxes indicating the location of the cells.

**Build a prototype system**

After the training cell detection model, I designed a software graphical user interface and integrated it with the model to create a stand-only application.

**Observe and evaluate the system**

The system is evaluated on its ability to detect a cell.

## 3.3 Experimental Setup

**Hardware Environment:** A Virtual Machine from Google Research that allows the writing and execution of arbitrary python code through a browser served as the hardware. Based on Jupyter notebook, the virtual machine requires no setup and provides access to computing resources, including GPUs. I used the professional version of Google Collaboratory (Colab Pro) for the research because it provides access to T4 and P100 GPUs and provides high memory.

**Software Environment:** A high-level programming language called Python was selected for the research because it is easy to learn and use, and the fact that it houses most machine learning tools makes it widely used for machine learning algorithms.

CUDA, a computing platform developed by Nvidia for using GPUs in general-purpose applications like deep learning and cuDNN, comes pre-installed in Google Colab, making it faster and more efficient than training on a CPU enabled system. For the Graphical Users Interface, I used PyQt, the most popular Python bindings for the Qt cross-platform C++ framework.

## 3.4 Machine Learning Process

### 3.4.1 Design of the Dataset

Before commencing the training of a machine learning model, I prepared the dataset. The importance of data cannot be overstated as the network needs a pattern to recognize. Therefore, most of the time spent on machine learning projects is on data. The garbage in/garbage out principle applies to most systems; consequently, a researcher must take special care to ensure the model gives the best possible result. The preparation of a dataset can be divided into three parts: data acquisition, data preprocessing, and data labeling.

- **Data acquisition**

A dataset was created by collecting images from the daily operation of the main system consisting of a microscopic view of cultured cells. However, some of the images had a low resolution. They were included in the primary dataset to increase variety and bias reduction since it showed the system’s capability or lack, therefore, to produce a low-quality image.

- **Data preprocessing**

The conventional Neubauer Chamber method the thesis aims to automated has a specific way of counting cells in grids (Fig. 12(a)). To automate this process, I created a python script to crop out the images with the dimension of the Neubauer Chamber grids (Fig .12(b)). To allow for a straightforward assessment, the images should ideally have the same size or same aspect ratio. Additionally, I altered the brightness, hue, contrast, and saturation to increase the dataset.

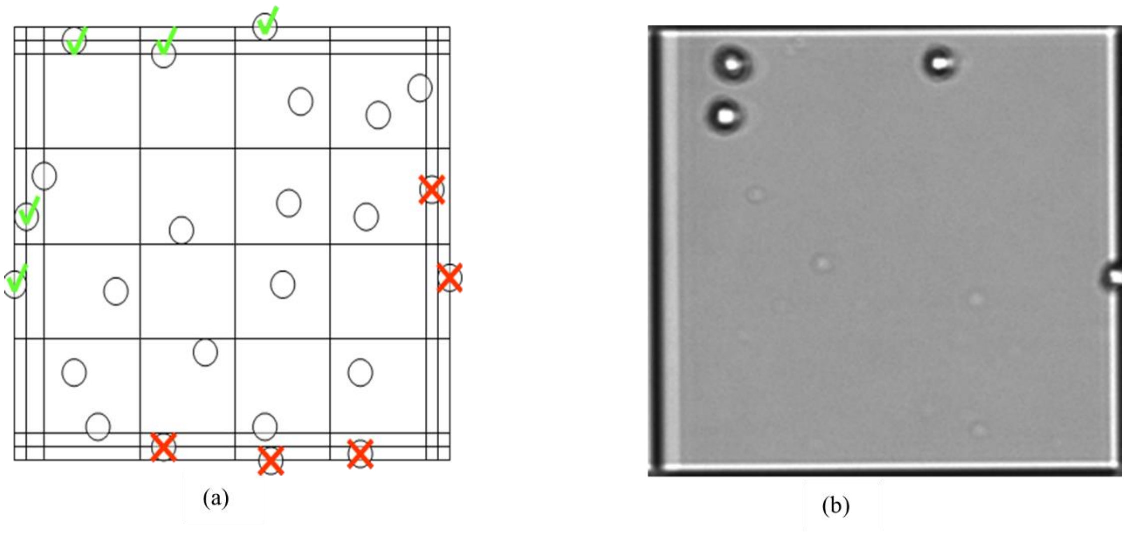
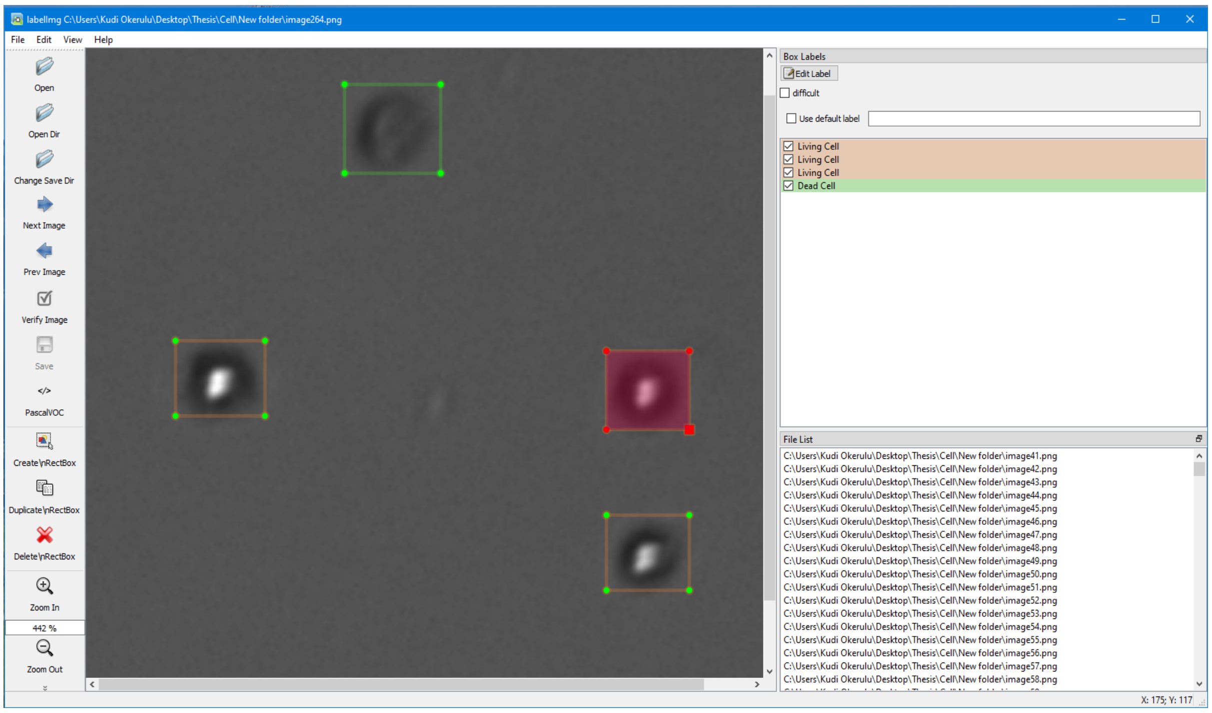


*Figure 12:The Neubauer chamber count process (a) and Cell image with three cells (b).*

- **Data Labeling**

The preprocessed images were labeled manually using the 'LabelImg' tool (*github.com/tzutalin/labelImg*). I labeled each image by drawing a bounding box around the respective classes of the objects (Living cell and Dead cell) as shown in (Fig 13). As a result, an annotated XML file was generated for each image containing the coordinates' location of objects and the Label name. These files enable the algorithm to detect the cells.



☑ Living Cell
☑ Living Cell
☑ Living Cell
☑ Dead Cell

```
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image41.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image42.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image43.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image44.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image45.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image46.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image47.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image48.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image49.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image50.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image51.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image52.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image53.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image54.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image55.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image56.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image57.png
C:\Users\Kudi Okerulu\Desktop\Thesis\Cell\New folder\image58.png
```

*Figure 13:Labeling process using LabelImg*

### 3.4.2 Training and Evaluation

TensorFlow's Object Detection API using Faster R-CNN is the Framework chosen for the training. The literature review showed it to be a powerful tool used to build and deploy image recognition, and it belongs to the two-stage type of object detection models.

- **Parameter optimization and configuration**

I made various changes to the default configuration of the Faster R-CNN files to evaluate the performance, including the learning rate, number of classes, label map, batch sizes, momentum, and decay parameters. The number of classes is two since the algorithm is projected to detect and recognize two classes – Living Cells and Dead Cells.

- **Splitting dataset into training, testing, and validation.** The dataset, which can be found at (*github.com/maxKudi/Cell-Dataset*), was split into two sets: training, and validation with a distribution of 80 and 20 percent of the images, respectively (Fig. 14). This is a method used by various Researchers (42) (43).

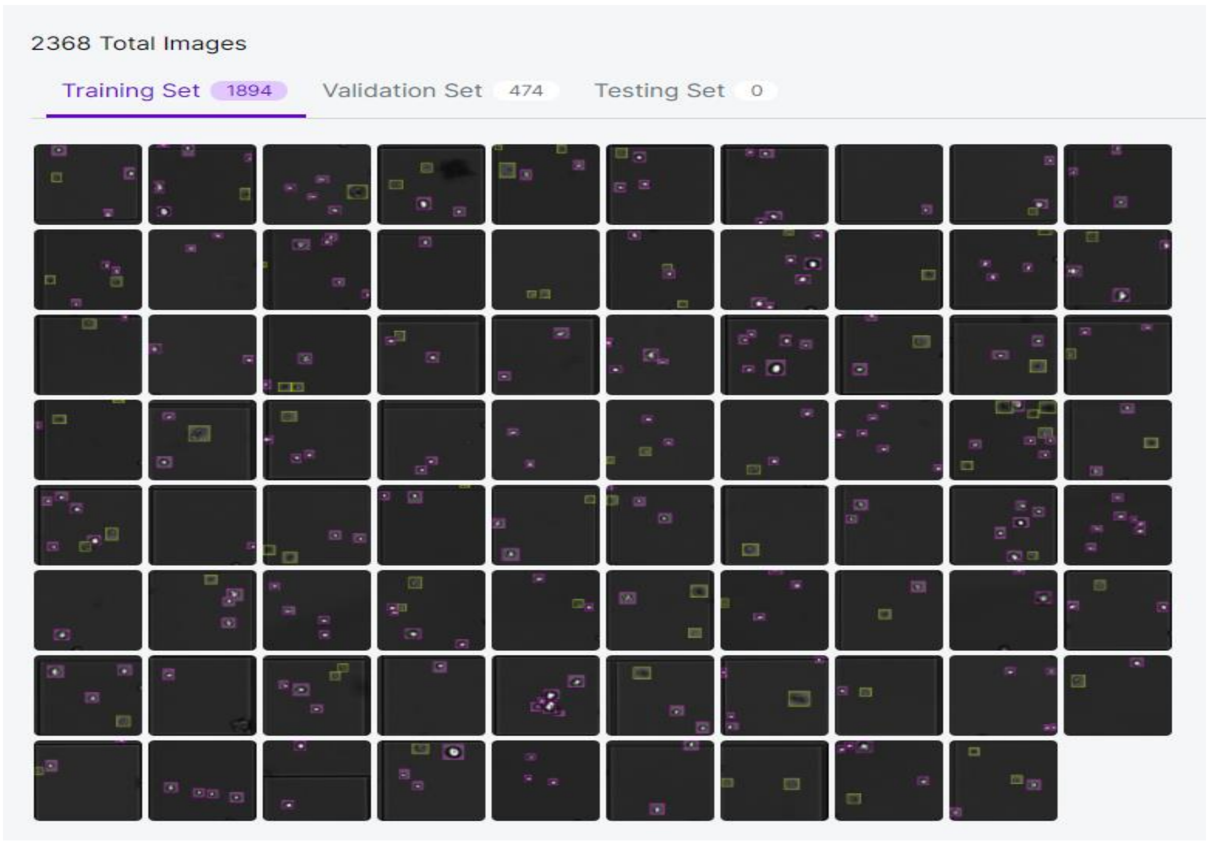
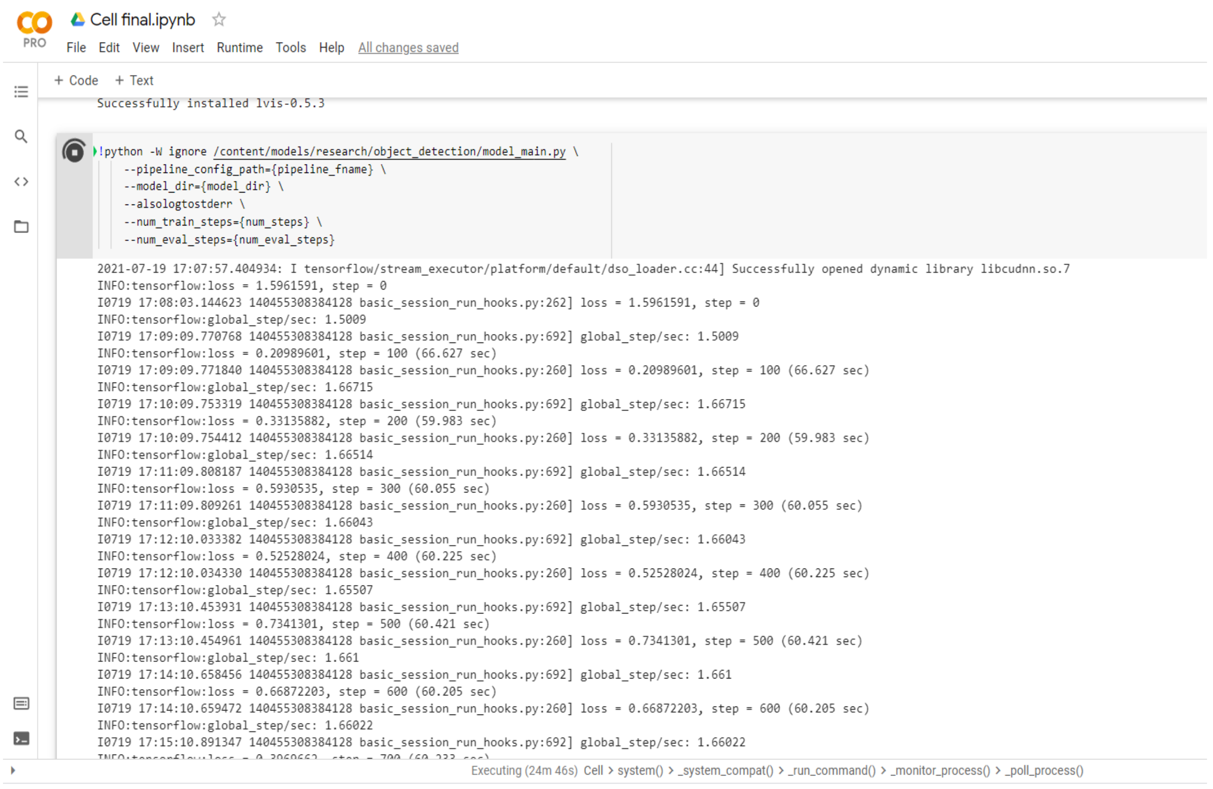


*Figure 14: Dataset*

- **Transfer Learning**

Transfer learning is almost certainly the advantage deep learning models have over shallow models because the models developed from other datasets are reused as the launching point of the training of a new model. Basically, using the knowledge gained while solving task A for task B. To apply transfer learning to our model, the weights from the model architectures Faster R-CNN trained on Microsoft Common Object in Context (MS COCO) dataset (44) are used to transfer the knowledge gained, leading to a decrease in time spent training.

After setting up the configuration and techniques mentioned above, the training process is initialized as shown in (Table 1). In the steps shown in (Fig. 15), I observed that the classification loss starts at a high value and decreases gradually as the model learns.


Cell final.ipynb
PRO
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Executing (24m 46s) Cell > system() > _system_compat() > _run_command() > _monitor_process() > _poll_process()

```
Successfully installed lvis-0.5.3

!python -W ignore /content/models/research/object_detection/model_main.py \
    --pipeline_config_path={pipeline_fname} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --num_eval_steps={num_eval_steps}

2021-07-19 17:07:57.404934: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudnn.so.7
INFO:tensorflow:loss = 1.5961591, step = 0
I0719 17:08:03.144623 140455308384128 basic_session_run_hooks.py:262] loss = 1.5961591, step = 0
INFO:tensorflow:global_step/sec: 1.5009
I0719 17:09:09.770768 140455308384128 basic_session_run_hooks.py:692] global_step/sec: 1.5009
INFO:tensorflow:loss = 0.20989601, step = 100 (66.627 sec)
I0719 17:09:09.771840 140455308384128 basic_session_run_hooks.py:260] loss = 0.20989601, step = 100 (66.627 sec)
INFO:tensorflow:global_step/sec: 1.66715
I0719 17:10:09.753319 140455308384128 basic_session_run_hooks.py:692] global_step/sec: 1.66715
INFO:tensorflow:loss = 0.33135882, step = 200 (59.983 sec)
I0719 17:10:09.754412 140455308384128 basic_session_run_hooks.py:260] loss = 0.33135882, step = 200 (59.983 sec)
INFO:tensorflow:global_step/sec: 1.66514
I0719 17:11:09.808187 140455308384128 basic_session_run_hooks.py:692] global_step/sec: 1.66514
INFO:tensorflow:loss = 0.5930535, step = 300 (60.055 sec)
I0719 17:11:09.809261 140455308384128 basic_session_run_hooks.py:260] loss = 0.5930535, step = 300 (60.055 sec)
INFO:tensorflow:global_step/sec: 1.66043
I0719 17:12:10.033382 140455308384128 basic_session_run_hooks.py:692] global_step/sec: 1.66043
INFO:tensorflow:loss = 0.52528024, step = 400 (60.225 sec)
I0719 17:12:10.034330 140455308384128 basic_session_run_hooks.py:260] loss = 0.52528024, step = 400 (60.225 sec)
INFO:tensorflow:global_step/sec: 1.65507
I0719 17:13:10.453931 140455308384128 basic_session_run_hooks.py:692] global_step/sec: 1.65507
INFO:tensorflow:loss = 0.7341301, step = 500 (60.421 sec)
I0719 17:13:10.454961 140455308384128 basic_session_run_hooks.py:260] loss = 0.7341301, step = 500 (60.421 sec)
INFO:tensorflow:global_step/sec: 1.661
I0719 17:14:10.658456 140455308384128 basic_session_run_hooks.py:692] global_step/sec: 1.661
INFO:tensorflow:loss = 0.66872203, step = 600 (60.205 sec)
I0719 17:14:10.659472 140455308384128 basic_session_run_hooks.py:260] loss = 0.66872203, step = 600 (60.205 sec)
INFO:tensorflow:global_step/sec: 1.66022
I0719 17:15:10.891347 140455308384128 basic_session_run_hooks.py:692] global_step/sec: 1.66022
```

*Figure 15:The Training Process on Google Collaboratory.*

*Table 1: Logic of the training process*

```
1:   Import All Necessary Libraries
2:   Import Dataset
3:   Import Model
4:   Import Pretrained weights
5:   Configure model (Batch size, Steps)
6:   Build model
7:   Add pointers to the TF-Records and Configuration Files
8:   If tf-records and configured files are available
9:       Run the Training process
10:      Investigates models validation using Tensor Board
11:      Save Trained Model
12:  Import Libraries for Inference
13:  Run Model on Testing Dataset
```

### 3.4.3 Model Performance Assessment

After the training of the model, a question arises on how good the model is, and to be able to evaluate the performance of a model, and different evaluation metrics are used.

The most widely used metrics are:

- Overfitting or Underfitting
- Accuracy
- $F_1$ Score (Precision/Recall)

**Overfitting** is the characteristics of a model that shows the model predicting with high accuracy, the dataset used for the training session but frequently making errors when presented with examples that were not seen during training.

**Underfitting** occurs when the model makes mistakes on the training dataset, showing the model's inability to predict the dataset during training. Underfitting can be attributed to the model's simplicity or the lack of information on the dataset's features.

**Accuracy** is provided by the number of correctly classified predictions made by the model divided by the number of predictions. This can be represented as:

$$Correct\ Predictions\ (CP) = True\ Positives\ +\ True\ Negatives$$

$$Total\ Predictions\ (TP) = TruePositives\ +\ TrueNegatives\ +\ FalsePositives\ +\ FalseNegatives$$

$$Accuracy = \frac{CP}{TP}$$

Where a False Negative is defined as the false detection of the class trained, meaning the algorithm failed to detect an object that should be detected. A False Positive is defined as an incorrect detection, also called misclassification of objects. On the other hand, a True Positive is defined as the correct detection of the object trained, while a True Negative is defined as a correct misdetection, meaning nothing was detected when there was nothing to detect.

**$F_1$ Score** is used to measure a test's accuracy. It is good if the overall number of false negatives and positives are low.

$$F1\ Score = \frac{2\ *\ Precision\ *\ Recall}{(Precision\ +\ Recall)} \tag{3.0}$$

The Most Frequently used metrics to assess a model are **Precision** and **Recall**.

The **Precision** is the ratio of the true positives to the overall sum of positive results predicted by the model.

$$Precision = \frac{True\ Positives}{(True\ Positives\ +\ False\ Positives)} \tag{3.1}$$

The **Recall** is the ratio of the true positives to the sum of the true positives and the false negatives predicted by the model.

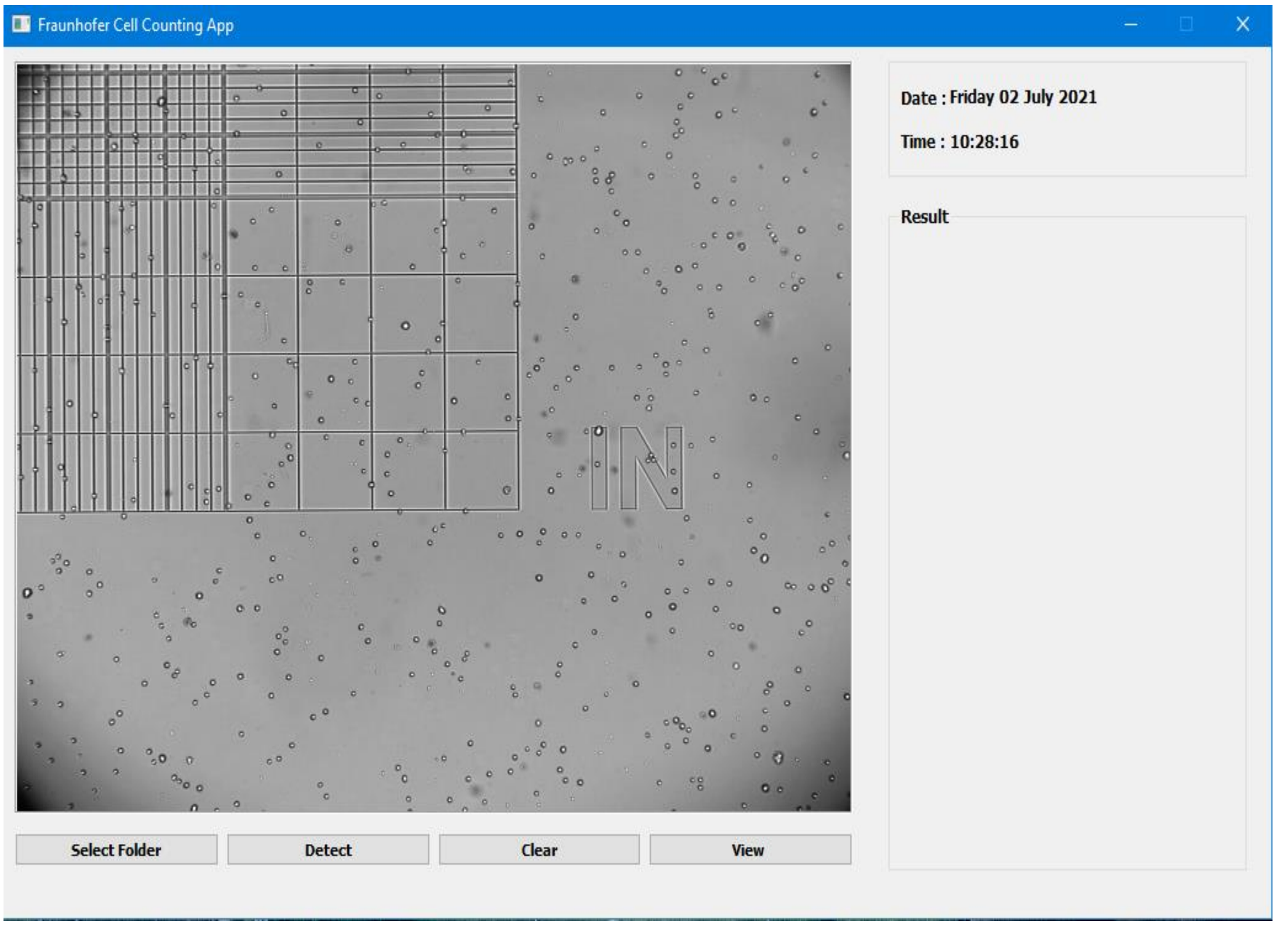
$$Recall = \frac{True\ Positives}{(True\ Positives\ +\ False\ Negatives)} \tag{3.2}$$

# 4. Results

## 4.1 Experimental Results

Following the conclusion of the training process of the algorithms, I successfully created a cell detector application, as can be seen in (Fig. 16). The result of the individual test conducted on Faster R-CNN are as follows:

- From the figure presented, it is worth noting that both algorithms performed the basic function of detecting the Living and Dead Cells with a high confidence level.
- Figure 17 shows the detection made by the Faster R-CNN model.
- The collection of false positives, true positives, false negatives, and true negatives helps with calculating the Precision, Recall, and Accuracy.



Date : Friday 02 July 2021

Time : 10:28:16

*Figure 16:Implemented GUI system for cell detection.*

*Table 2: Logic for running the GUI application for cell detection.*

```
1:  Open Application
2:  Import All Necessary Libraries
3:  Import Object detection algorithm
4:  Import Trained weights
5:  Import OPC-UA
6:  Import Cropping Script
7:  living cells = 0
8:  dead cells = 0
9:  Select Folder with Test Images
10: Click Detect button
11: if Folder Contains Images and Detect button clicked
12:     Crop the Region of interest on Image using cropping script and save
13:     Apply the predictions using Trained weights
14:     Remove fake predictions using appropriate threshold value
15:     for i in range (length of predictions) do
16:         label = label of cell
17:         # Cell counting
18:         if label = = ‘L_Cell’ then living cell = living cell + 1
19:         else if label = = ‘D_Cell’ then dead cell = dead cell + 1
20:         end if
21:         i = i + 1
22:         Save image
23:     end for
24: end if
25: Display results on result panel
26: Click view button to view saved results – Click close to close view GUI
27: Click the Clear button for file handling
28: Close Application
```

## 4.2 Detection Performance

As stated in the previous chapters, I chose Faster RCNN as the object detection algorithm for the detection of cells. However, it is quite unclear what hyperparameter configuration will lead to a good detection performance as the behavior of a convolutional neural network can be difficult to predict. To mitigate this problem, different experiments were conducted with varying hyperparameters, with the dataset being the top hyperparameter in focus. The dataset comprises 2360 images of size 220 X 200, and initially, each training session ran at 10,000 steps. After the first training trial, the CNN found it difficult to learn from the image, so they were augmented to a 416 x 416 size, and this proved to be a better image quality for the learning process, and I also adjusted the steps to 40,000. Various experiments and their outcome are presented in Table 3. Table 4 went further to show the effectiveness of the object detection process when compared to the Neubauer chamber.

*Table 3: Experimental results with changing parameters*

| **Experiment no.** | **Batch Size** | **Image number** | **Epochs** | **Learning Rate** | **Accuracy** |
|---|---|---|---|---|---|
| 1 | 12 | 2,360 | 10,000 | 0.00002 | 89.6% |
| 2 | 12 | 5,100 | 10,000 | 0.00002 | 91.5% |
| 3 | 32 | 5,100 | 10,000 | 0.00002 | 87.3% |
| 4 | 12 | 2,360 | 40,000 | 0.00002 | 93.2% |
| 5 | 12 | 5,100 | 40,000 | 0.00002 | 95.7% |

*Table 4: Results of the comparison between the counting processes*

| | **(Object Detection Counting)** | **(Manual Counting)** |
|---|---|---|
| Number of Living Cells | 298 | 289 |
| Number of Dead Cells | 13 | 11 |
| Average of Living Cells | 74.5 | 72.25 |
| Average of Dead Cells | 3.25 | 2.75 |
| Percentage of Living Cells | 95.82% | 96.33% |
| Percentage of Dead Cells | 4.18% | 3.67% |

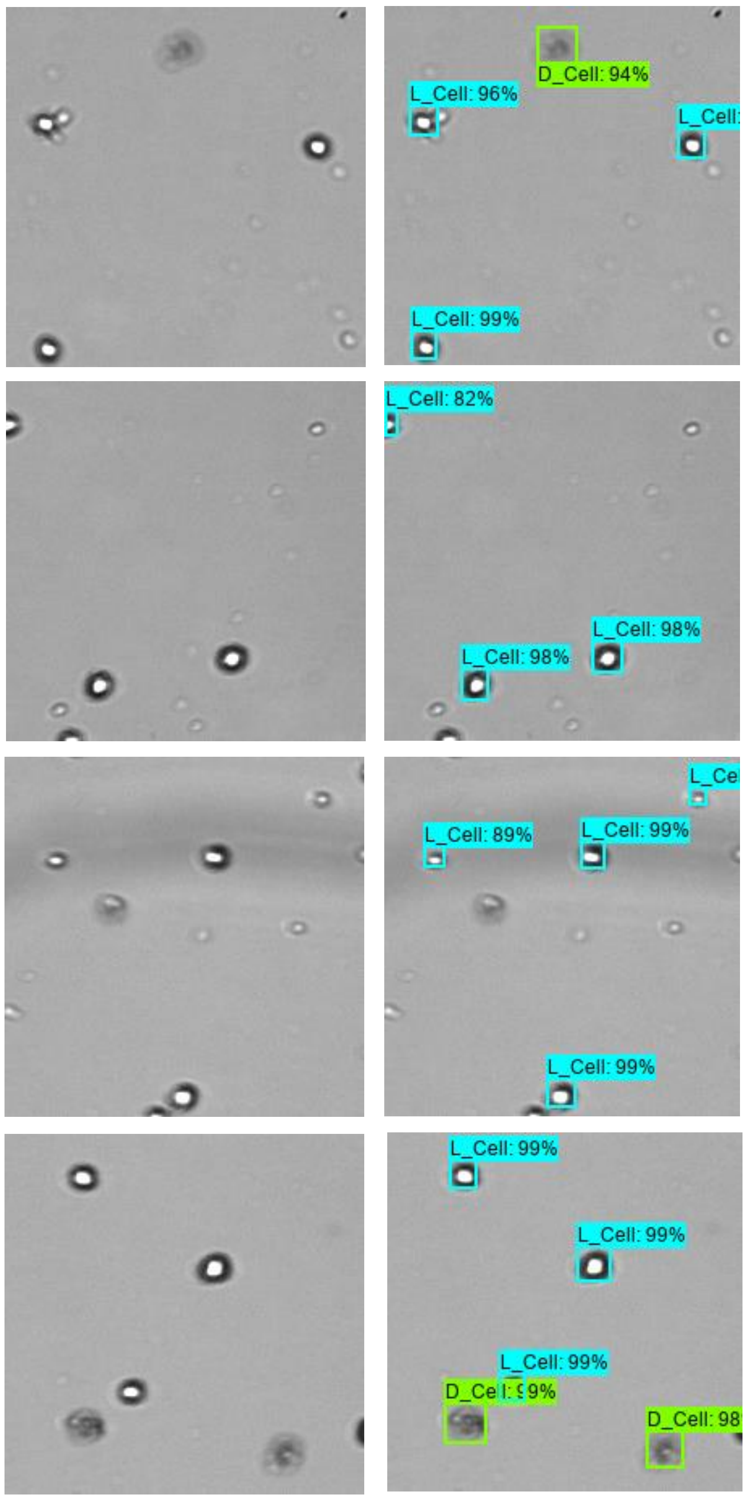


*Figure 17:Predictions made by the Cell detection model.*

### 4.2.1 Evaluation Metrics

The evaluation metrics is a crucial part in object detection as it provides insight into the accuracy obtained by the framework. To compute this, the matching matrix, also known as the confusion matrix, is used to calculate the statistical classification. Table 6 further illustrates the use of the matching matrix with each row representing the instances in an actual class and the columns corresponding to the instances in a predicted class. Other performance metrics like accuracy, precision, and recall can also be calculated using the matching matrix.

*Table 5:Confusion matrix*

| **Actual Class** | **Predicted Class** | |
|---|---|---|
| | **Living Cells** | **Dead Cells** |
| **Living Cells** | *True Positive* | *False Negative* |
| **Dead Cells** | *False Positive* | *True Negative* |

*Table 6: Model result using the confusion matrix*

| **Actual Class** | **Predicted Class** | |
|---|---|---|
| | **Living Cells** | **Dead Cells** |
| **Living Cells** | *298* | *0* |
| **Dead Cells** | *1* | *12* |

## Accuracy

Accuracy is a helpful metric when calculating the errors in predicting, is given by the number of correctly predicted examples divided by the Total prediction. And using the equations below and the confusion matrix values, the accuracy of the model is as follows:

$$Correct\ Predictions\ (CPs) = True\ Positives + True\ Negatives = 298 + 12 = 310$$

$$Total\ Predictions\ (TPs) = TruePositives + TrueNegatives + FalsePositives + FalseNegatives$$

$$(TPs) = 298 + 12 + 1 + 0 = 311$$

$$Accuracy = \frac{CPs}{TPs} = \frac{310}{311} = 0.9967 * 100 = 99.7\%$$

## Precision

The precision measures the percentage of the correct and accurate predictions and can be calculated using the confusion matrix formula stated in the method section of the thesis.

$$P = \frac{TP}{(TP + FP)} = \frac{298}{(298 + 1)} = 0.9966 = 99.66\%$$

Where FP is the total number of false positives, and the TP is the total number of true positives.

## Recall

Like the Precision, the Recall is an interesting evaluation metric as it computes the percentage of correctly classified predictions. The additional value FN is the number of false negatives.

$$Recall = \frac{TP}{(TP + FN)} = \frac{298}{(298 + 0)} = 1 = 100\%$$

## F1 Score

$$F1\,Score = \frac{2 * Precision * Recall}{(Precision + Recall)} = \frac{2 * 0.9966 * 1}{(0.9966 + 1)} = \frac{1.9932}{1.9966} = 0.9982 = 99.82\%$$

I chose the trained model with the best result for the integration of the system. Based on Figure 18, the total loss on the validation set can be observed at 0.48. It is also visualized using TensorFlow Board as shown in (Fig. 19) here you can find the graphical representation of the loss, learning rate, precision, and recall.

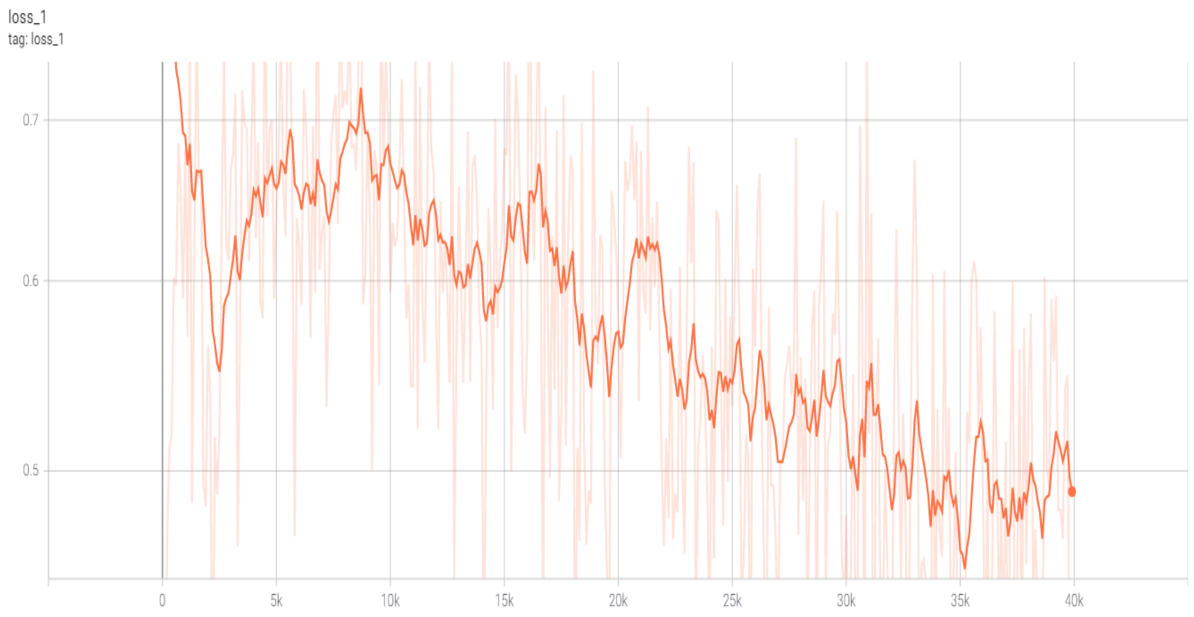
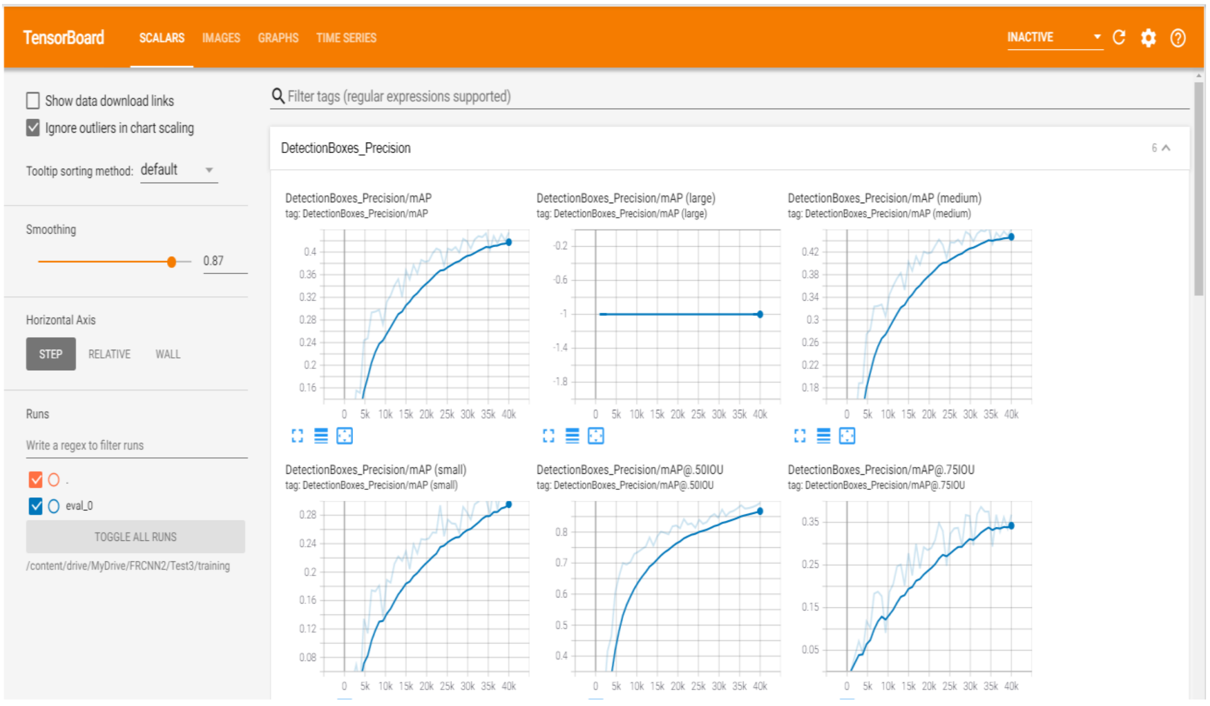


*Figure 18:The Total loss of the model.*



TensorBoard
SCALARS
IMAGES
GRAPHS
TIME SERIES
INACTIVE
Filter tags (regular expressions supported)
DetectionBoxes_Precision
Write a regex to filter runs
TOGGLE ALL RUNS
DetectionBoxes_Precision/mAP
tag: DetectionBoxes_Precision/mAP
DetectionBoxes_Precision/mAP (large)
tag: DetectionBoxes_Precision/mAP (large)
DetectionBoxes_Precision/mAP (medium)
tag: DetectionBoxes_Precision/mAP (medium)
DetectionBoxes_Precision/mAP (small)
tag: DetectionBoxes_Precision/mAP (small)
DetectionBoxes_Precision/mAP@.50IOU
tag: DetectionBoxes_Precision/mAP@.50IOU
DetectionBoxes_Precision/mAP@.75IOU
tag: DetectionBoxes_Precision/mAP@.75IOU

☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.87
Horizontal Axis: STEP RELATIVE WALL
Runs
☑ .
☑ eval_0
/content/drive/MyDrive/FRCNN2/Test3/training

*Figure 19:Tensor board View of Training in Progress.*

# 5. Discussion

## 5.0 Discussion

Faster RCNN is on record as the most accurate model from the literature review, running at a one image per second speed. Although YOLOv4 can detect at a faster rate, ten (10) times more than most state-of-the-art models and ideally the best for real-time detection, speed is not really of much concern for the cell detection model because the image from the microscope is static. This made Faster RCNN the algorithm of choice in the experiment.

The finding of the Research pinpoints three different practices that can affect the performance of the Faster RCNN object detector:

- Dataset structure
- Transfer Learning
- Hyperparameter optimization

Furthermore, the study also showed that overfitting occurs when training approaches 40,000 steps of training.

## 5.2 Dataset structure

As previously stated, the object detection model relies heavily on the dataset composition and making changes to it can significantly increase the accuracy of the object detector model. Initially, 2500 cropped images were annotated using the LabelImg. This part of the Research proved to be the hardest because it was labor-intensive, requiring approximately 48+ hours of careful work to ensure the data was not mislabeled. By augmenting the dataset using Roboflow, more additional datasets were generated. The augmented incorporated adding white noise, changing the brightness and contrast to the original dataset, and the difference in the image’s pixel values was sufficient for the computer to classify it as entirely new data. The improved dataset proved to increase the accuracy of the model.

## 5.3 Transfer Learning

The literature review mentioned the need for transfer learning during object detection training as models trained on a broader dataset such as MS COCO, and ImageNet produces weights which made it easy to detect objects in the hidden layers and decreases the overall time spent on training. Although the Research didn’t go deeper into confirming these state facts with results. The training process for 10,000 steps without transfer learning took more than 7 hours with a low loss result, as opposed to training with the transfer learning, which normally took 3 – 4hours to complete, and it confirmed the increase in time spent training. Further investigation into the reason this difference exists is required.

## 5.4 Hyperparameter optimization

The study showed that increasing the batch size decreases the model's accuracy, so I used a batch size of 12 for the remaining experiments to save time. I also observed that input of 220 x 220 images wasn't possible for the learning process, and since the literature didn't hint at that occurring, it most like was a bug in my code. The reason for this was not investigated, as scaling up the input size using augmentation mitigated the problem. Additional hyperparameters in Faster RCNN, such as the decay and learning rate, could be further investigated for a possible increased inaccuracy.

## 5.5 Problems Encountered

The argumentation of the dataset, though, increased the model’s performance, as shown in Table 4; it also introduced a problem false inclusion of tiny with spots on the dataset as a living cell and overshot the value of the living cells. This problem was due to the change in contrast between the cell boundary and background. And I spotted it with the last batch of the testing dataset, which made it difficult to exclude from the software.

# 6. Conclusions and Future work

## 6.1 Conclusions

This thesis aim was to automate the detection and counting of living and dead cells. I chose the most suitable model from the experiment was chosen based on the performance evaluation. The results of the Research were discussed in the early chapters, along with the answer to the Research questions formulated in the introduction. While the object detection solution needs additional work to provide a perfect result, the result achieved is viable and safe to use, and some of the concluding remarks are as follows:

**Research Question One: Is deep learning models suitable for the detection/recognition of cells?** With the help of a literature review, I obtained knowledge about the various deep learning models capable of performing object detection. Upon studying the performance metrics of the algorithms on the standard dataset (MS COCO), Faster RCNN was identified as the most suitable and efficient deep learning model. A deeper dive into the architecture of the selected algorithms helped increase the understanding of how it works.

**Research Question Two: What is the performance of the selected deep learning models when compared with the conventional method of cell counting using Neubauer Chamber?** After preparing the dataset, I trained the model and subsequently tested the trained model on test images collected from the cell culturing system. Using the confusion matrix, the number of true positives, true negatives, false positives, and false negatives was identified for each cropped image and using these results, the Precision, Recall, Accuracy, and F1 Score of the model was calculated. The deep learning model performed relatively well and gave a consistent result when compared with the Manual counting process. The result section of the thesis holds a detailed analysis.

**Research Question Three: What is the possibility of creating a Graphical User Interface with the deep learning method for optimal use in a laboratory?** Research into creating a Graphical User Interface showed five different methods as PyQT5, Kivy, Tkinter, PySide, and wxPython. The Research also showed the possibility of integrating a machine learning code into the system, and I chose PyQt5 as the framework for the graphical user interface of the thesis.

**Research Question Four: Can the Software created be integrated with the Robotics cell culturing system?** Since Open Platform Communication Unified Architecture (OPC-UA) is the platform that integrates the functionality of the robotics cell culturing system, a python version of the communication implementation was chosen to be used to integrate the cell counting software functionality into the system. Although the start bit for the beginning of the system was not successfully incorporated, the done bit was, which answers the Research question. However, more work will be needed to make it a completely automated system.

## 6.2 Future Work

Computer vision is a topic that has been attracting most researchers recently because of its ability to be applied to a different area of life, easing the stress of repetitive vision problems in the Research area. There are different state-of-the-art object detectors in computer vision, and Research is currently ongoing to make these models better at object detection. Therefore, future work can be done on the object detection models created from this thesis to obtain better results and improve accuracy.

It is essential to mention the obstacles encountered during the experiment due to the lack of an adequate experimentation setup. I carried out the research work using Google Collaboratory, a cloud-based environment with a high Graphical Processing Unit. Although this platform offers researchers quick access to a powerful computing device, stable internet access is required to use it, and a little break in transmission will cancel the process. Therefore, a physical system is required for an uninterrupted training process, which can eventually lead to an improved model.

Secondly, the cropping technique employed is static, and it assumes that the image to be processed stays the same. To have a more advanced method, template matching or edge detection using OpenCV is suggested.

The object recognition method used though currently one the state-of-the-art methods, newer models exist that outperforms it in the speed of processing and generalization.

## Reference

1. Lanza R, Langer R, Vacanti J. Principles of Tissue Engineering. Principles of Tissue Engineering. London: Academic Press; 2013.
2. Schmitz S. Der Experimentator: Zellkultur. Heidelberg: Spektrum Akademischer Verlag; 2011.
3. Girshick R, Donahue J, Darrell T, Malik J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition [Internet]. IEEE; 2014. p. 580–7. Available from: http://ieeexplore.ieee.org/document/6909475/
4. Shelhamer E, Long J, Darrell T. Fully Convolutional Networks for Semantic Segmentation. IEEE Trans Pattern Anal Mach Intell [Internet]. 2017 Apr 1;39(4):640–51. Available from: http://ieeexplore.ieee.org/document/7478072/
5. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow - Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd Edition (2019). Vol. 44, Journal of Physics A: Mathematical and Theoretical. O'Reilly Media; 2011. 1689–1699 p.
6. Murray M-Y. Comprehensive Biotechnology. Comprehensive Biotechnology. Elsevier; 2019.
7. Lab C x I. Basic Process in Cell Culture in General | Basic knowledge | Cell x Image Lab - Nikon [Internet]. Nikon. 2021. Available from: https://www.healthcare.nikon.com/en/ss/cell-image-lab/knowledge/process.html
8. Cadena-Herrera D, Esparza-De Lara JE, Ramírez-Ibañez ND, López-Morales CA, Pérez NO, Flores-Ortiz LF, et al. Validation of three viable-cell counting methods: Manual, semi-automated, and automated. Biotechnol Reports [Internet]. 2015 Sep;7:9–16. Available from: https://linkinghub.elsevier.com/retrieve/pii/S2215017X15000235
9. Burdon RH, Knippenberg PH van, Sharpe PTBT-LT in B and MB, editors. Methods of cell counting and assaying cell viability. In: Laboratory Techniques in Biochemistry and Molecular Biology [Internet]. Elsevier; 1988. p. 7–17. Available from: https://www.sciencedirect.com/science/article/pii/S0075753508706272
10. Kainz P, Urschler M, Schulter S, Wohlhart P, Lepetit V. You should use regression to detect Cells. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer; 2015. p. 276–83.
11. Kost H, Homeyer A, Molin J, Lundström C, Hahn H. Training nuclei detection algorithms with simple annotations. J Pathol Inform. 2017;8(1).
12. Arteta C, Lempitsky V, Noble JA, Zisserman A. Learning to detect cells using non-overlapping extremal regions. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer; 2012. p. 348–56.
13. Moor J. The Dartmouth College Artificial Intelligence Conference: The next fifty years. AI Mag. 2006;27(4):87–91.
14. Samuel AL. Some studies in machine learning using the game of checkers. IBM J Res Dev. 2000;44(1–2):207–19.
15. Bishop MC. Pattern Recognition. Springer New York. 2016.

16. Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature. 1986;323(6088):533–6.

17. Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: 32nd International Conference on Machine Learning, ICML 2015. PMLR; 2015. p. 448–56.

18. Vemuri VK. The Hundred-Page Machine Learning Book. Vol. 22, Journal of Information Technology Case and Application Research. Andriy Burkov Canada; 2020. 136–138 p.

19. Hijazi S, Kumar R, Rowen C. Using convolutional neural networks for image recognition. Cadence (2015). 2019; Available from: https://ip.cadence.com/uploads/901/cnn_wp-pdf

20. Domingos P. A few useful things to know about machine learning. Commun ACM. 2012;55(10):78–87.

21. Chicco D. Ten quick tips for machine learning in computational biology. BioData Min. 2017;10(1):1–17.

22. Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. Commun ACM. 2017;60(6):84–90.

23. Voulodimos A, Doulamis N, Doulamis A, Protopapadakis E. Deep Learning for Computer Vision: A Brief Review. Comput Intell Neurosci. 2018;2018.

24. Panigrahi S, Nanda A, Swarnkar T. A Survey on Transfer Learning. Smart Innov Syst Technol. 2021;194(10):781–9.

25. He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE International Conference on Computer Vision. 2015. p. 1026–34.

26. He K, Gkioxari G, Dollár P, Girshick R. Mask R-CNN. Proc IEEE Int Conf Comput Vis [Internet]. 2017 Mar 20;2961–9. Available from: http://arxiv.org/abs/1703.06870

27. Shelhamer E, Long J, Darrell T. Fully Convolutional Networks for Semantic Segmentation. IEEE Trans Pattern Anal Mach Intell. 2017;39(4):640–51.

28. Lu X, Li Q, Li B, Yan J. MimicDet: Bridging the Gap Between One-Stage and Two-Stage Object Detection. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer; 2020. p. 541–57.

29. Jiao L, Zhang F, Liu F, Yang S, Li L, Feng Z, et al. A Survey of Deep Learning-based Object Detection. 2019 Jul 11; Available from: http://arxiv.org/abs/1907.09408

30. Girshick R. Fast R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision [Internet]. IEEE; 2015. p. 1440–8. Available from: http://ieeexplore.ieee.org/document/7410526/

31. Ren S, He K, Girshick R, Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Trans Pattern Anal Mach Intell. 2017;39(6):1137–49.

32. Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: Unified, real-time object detection. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2016. p. 779–88.

33. Redmon J, Farhadi A. YOLOv3: An Incremental Improvement. arXiv Prepr arXiv180402767 [Internet]. 2018; Available from: http://arxiv.org/abs/1804.02767

34. Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, et al. SSD: Single shot multibox detector. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer; 2016. p. 21–37.

35. Adagale SS, Pawar SS. Image segmentation using PCNN and template matching for blood cell counting. In: 2013 IEEE International Conference on Computational Intelligence and Computing Research, IEEE ICCIC 2013. IEEE; 2013. p. 1–5.

36. Grishagin I V. Automatic cell counting with ImageJ. Anal Biochem [Internet]. 2015 Mar;473:63–5. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0003269714005570

37. Hernández CX, Sultan MM, Pande VS. Using Deep Learning for Segmentation and Counting within Microscopy Data. 2018 Feb 28; Available from: http://arxiv.org/abs/1802.10548

38. Liu, Junker, Murakami, Hu. Automated Counting of Cancer Cells by Ensembling Deep Features. Cells [Internet]. 2019 Sep 2;8(9):1019. Available from: https://www.mdpi.com/2073-4409/8/9/1019

39. Pride L, Agehara S. Useful Image-Based Techniques for Manual and Automatic Counting Using ImageJ for Horticultural Research. Edis [Internet]. 2021;2021(1). Available from: https://journals.flvc.org/edis/article/download/125670/128247

40. Nunamaker JF, Chen M, Purdin TDM. Systems development in information systems research. J Manag Inf Syst [Internet]. 1990 Dec 21;7(3):89–106. Available from: https://www.tandfonline.com/doi/full/10.1080/07421222.1990.11517898

41. Zhao ZQ, Zheng P, Xu ST, Wu X. Object Detection with Deep Learning: A Review. IEEE Trans Neural Networks Learn Syst [Internet]. 2019 Nov;30(11):3212–32. Available from: https://ieeexplore.ieee.org/document/8627998/

42. Wei Y, Song N, Ke L, Chang MC, Lyu S. Street object detection / tracking for AI city traffic analysis. In: 2017 IEEE SmartWorld Ubiquitous Intelligence and Computing, Advanced and Trusted Computed, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart City Innovation, SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI 2017 - Conference Proceedings [Internet]. IEEE; 2018. p. 1–5. Available from: https://ieeexplore.ieee.org/document/8397669/

43. Elhoseiny M, Bakry A, Elgammal A. Multiclass object classification in video surveillance systems-Experimental study. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops [Internet]. IEEE; 2013. p. 788–93. Available from: http://ieeexplore.ieee.org/document/6595962/

44. Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, et al. Microsoft COCO: Common objects in context. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) [Internet]. 2014. p. 740–55. Available from: http://link.springer.com/10.1007/978-3-319-10602-1_48

# Appendix

## Appendix A: Cropping function for Dataset Preparation

```
#.......................................................................
# Importing All Necessary libraries
import tensorflow as tf
from tensorflow import keras
import cv2
import numpy as np
from matplotlib import pyplot as plt
import os
import glob as glob
#.......................................................................

# Mounting the Google drive
from google.colab import drive
drive.mount('/drive')

# Data Cropping
import cv2
from PIL import Image
from matplotlib import pyplot as plt
from pylab import *
import glob as glob

files = glob.glob ("/drive/My Drive/Dataset/Train/*.png")
savedir = "/save_folder path/"
height = 219
width = 225

def crop_image(height, width, k):
    for myFile in files:
        image = cv2.imread(myFile)
        # plt.imshow(image)
        imgheight = image.shape[0]
        imgwidth = image.shape[1]
        for y in range(0,imgheight, height):
            for x in range(0,imgwidth, width):
                cropped = image[y:y+height, x:x+width]
                if cropped.shape != (height, width, 3):
                  pass
                else:
                    print(cropped.shape)
                    save_to = savedir + "a" + str(k) + ".png"
                    print(save_to)
                    # plt.imshow(cropped)
                    cv2.imwrite(save_to, cropped)
                    myk += 1
```

## Appendix B: Training process on Google Collaboratory

```
# **Cell Counting by using Faster RCNN**

Trained data:
1. Image: 2450images
2. Test image:set B
3. Step 10000

Remark:
Colab disconnects the notebook if we leave it idle for more than 30 minutes

**To stop Google Colab From Disconnecting**
Ctrl+Shift+i run the following JavaScript code in your console:

function ClickConnect(){
    console.log("Clicked on connect button");
    document.querySelector("colab-connect-button").click()
}
setInterval(ClickConnect,60000)

#Step 1: Installation#

###1.1 install tensorflow 1.15
"""

# Commented out IPython magic to ensure Python compatibility.
# %tensorflow_version 1.x
!pip install tensorflow_gpu==2.0 --user
!pip install -q tf_slim

"""### 1.2 Selecting and Downloading a Pre-trained Model
Configs and Hyperparameters
"""

# If you forked the repo, you can replace the link.
repo_url = 'https://github.com/roboflow-ai/tensorflow-object-detection-faster-rcnn'

# Number of training steps - 1000 will train very quickly, but more steps will increase
accuracy.
num_steps = 40000  # increase to improve (200000)

# Number of evaluation steps.
num_eval_steps = 50
```

```
MODELS_CONFIG = {
    'ssd_mobilenet_v2': {
        'model_name': 'ssd_mobilenet_v2_coco_2018_03_29',
        'pipeline_file': 'ssd_mobilenet_v2_coco.config',
        'batch_size': 12
    },
    'faster_rcnn_inception_v2': {
        'model_name': 'faster_rcnn_inception_v2_coco_2018_01_28',
        'pipeline_file': 'faster_rcnn_inception_v2_pets.config',
        'batch_size': 12
    },
    'rfcn_resnet101': {
        'model_name': 'rfcn_resnet101_coco_2018_01_28',
        'pipeline_file': 'rfcn_resnet101_pets.config',
        'batch_size': 8
    }
}

# Pick the model you want to use (about the speed)
# Select a model in `MODELS_CONFIG`.
selected_model = 'faster_rcnn_inception_v2'

# Name of the object detection model to use.
MODEL = MODELS_CONFIG[selected_model]['model_name']

# Name of the pipline file in tensorflow object detection API.
pipeline_file = MODELS_CONFIG[selected_model]['pipeline_file']

# Training batch size fits in Colab's Tesla K80 GPU memory for selected model.
batch_size = MODELS_CONFIG[selected_model]['batch_size']

"""###1.3 Clone the `tensorflow-object-detection` repository or your fork."""

# Commented out IPython magic to ensure Python compatibility.
import os

# %cd /content

repo_dir_path = os.path.abspath(os.path.join('.', os.path.basename(repo_url)))

!git clone {repo_url}
# %cd {repo_dir_path}
!git pull
```

```
"""###1.4Mount your google drive
**need to access with link**
Click the link below and enter your authorization code
"""

from google.colab import drive
drive.mount('/content/drive')

"""#Step2:import training data

###2.1 Prepare `tfrecord` files
"""

"""###**2.2 [optional 1] google drive****
####Import google Drive and copy `train` and `valid` folder to Colab Folder
After ran this cell,you can see the train and valid folders are created under `content` > `
tensorflow-object-detection-faster-rcnn` > `data`
"""
## copy train folder from google drive to colab
!cp -r /content/drive/MyDrive/FRCNN2/Test4/train /content/tensorflow-object-detection-
faster-rcnn/data
## copy test folder
#!cp -r /content/drive/MyDrive/FRCNN2/test /content/tensorflow-object-detection-faster-
rcnn/data
## copy valid folder
!cp -r /content/drive/MyDrive/FRCNN2/Test4/valid /content/tensorflow-object-detection-
faster-rcnn/data

import tensorflow as tf

for example in tf.python_io.tf_record_iterator("/content/tensorflow-object-detection-
faster-rcnn/data/valid/Cells.tfrecord"):
    print(tf.train.Example.FromString(example))
```

```
"""###**2.2 [optional 2] using ROboflow**
Upload the training images to Roboflow
Roboflow automatically creates your TFRecord and label_map files that you need!
You will generate TFRecords and .CSV there and copy the link and paste it below.
To create a dataset in Roboflow and generate TFRecords, follow [this step-by-
step guide](https://blog.roboflow.ai/getting-started-with-roboflow/).
"""
# UPDATE THIS LINK - get our data from Roboflow
####################################################
### Update Links, or manually upload
####################################################
# Choose here download: CSV - Tensorflow object detection
!curl -
L "https://app.roboflow.com/ds/Er0fjKaMDZ?key=Pf7k6YFZjl" > roboflow.zip; unzip roboflow.zi
p; rm roboflow.zip
# Choose here download: Other - Tensorflow TFRecord
!curl -
L "https://app.roboflow.com/ds/bkfI9YriDt?key=mVqKVTwQQp" > roboflow.zip; unzip roboflow.zi
p; rm roboflow.zip

"""###2.3[Optional] List files"""

#%ls

"""###2.4[Optional] list images in the `train` and `valid`folder """

#%ls train

# show what we have in valid
#%ls valid/

"""###2.5 Update these TFRecord names"""

# NOTE: Update these TFRecord names from "white" and "white_label_map" to your files!
####################################################
### Rename Files According to the name you choose!
####################################################
test_record_fname = '/content/tensorflow-object-detection-faster-
rcnn/data/valid/Cells.tfrecord'
train_record_fname = '/content/tensorflow-object-detection-faster-
rcnn/data/train/Cells.tfrecord'
label_map_pbtxt_fname = '/content/tensorflow-object-detection-faster-
rcnn/data/train/Cells_label_map.pbtxt'
```

```
"""#Step 3: Install `models` package

###3.1 Download `models` folder
"""

# Commented out IPython magic to ensure Python compatibility.
# %cd /content
!git clone --quiet https://github.com/tensorflow/models.git


"""
###3.2  Replace visualization_utils.py

This .py file will counting object and save each counting result in result.txt

"""

## copy visualization_utils.py from google drive to colab
# important: the file contains the path where to store the result.txt file.
#            therefore it´s important to adjust the path if need be
!cp -
r /content/drive/MyDrive/FRCNN2/visualization_utils.py /content/models/research/object_dete
ction/utils

"""###3.3 Install protobuf, pycocotools & some other packages"""

# Commented out IPython magic to ensure Python compatibility.
!apt-get install -qq protobuf-compiler python-pil python-lxml python-tk

!pip install -q Cython contextlib2 pillow lxml matplotlib

!pip install -q pycocotools

# %cd /content/models/research
!protoc object_detection/protos/*.proto --python_out=.

import os
os.environ['PYTHONPATH'] += ':/content/models/research/:/content/models/research/slim/'

!python object_detection/builders/model_builder_test.py
```

```
"""###3.4 Download base model"""

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/models/research

import os
import shutil
import glob
import urllib.request
import tarfile
MODEL_FILE = MODEL + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'
DEST_DIR = '/content/models/research/pretrained_model'

if not (os.path.exists(MODEL_FILE)):
    urllib.request.urlretrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)

tar = tarfile.open(MODEL_FILE)
tar.extractall()
tar.close()

os.remove(MODEL_FILE)
if (os.path.exists(DEST_DIR)):
    shutil.rmtree(DEST_DIR)
os.rename(MODEL, DEST_DIR)

!echo {DEST_DIR}
!ls -alh {DEST_DIR}

fine_tune_checkpoint = os.path.join(DEST_DIR, "model.ckpt")
fine_tune_checkpoint
```

```
"""###3.5 Configuring a Training Pipeline

This is the last step before starting to train the model
"""

import os
pipeline_fname = os.path.join('/content/models/research/object_detection/samples/configs/',
 pipeline_file)

assert os.path.isfile(pipeline_fname), '`{}` not exist'.format(pipeline_fname)

def get_num_classes(pbtxt_fname):
    from object_detection.utils import label_map_util
    label_map = label_map_util.load_labelmap(pbtxt_fname)
    categories = label_map_util.convert_label_map_to_categories(
        label_map, max_num_classes=90, use_display_name=True)
    category_index = label_map_util.create_category_index(categories)
    return len(category_index.keys())

import re

num_classes = get_num_classes(label_map_pbtxt_fname)
with open(pipeline_fname) as f:
    s = f.read()
with open(pipeline_fname, 'w') as f:

    # fine_tune_checkpoint
    s = re.sub('fine_tune_checkpoint: ".*?"',
               'fine_tune_checkpoint: "{}"'.format(fine_tune_checkpoint), s)

    # tfrecord files train and test.
    s = re.sub(
        '(input_path: ".*?)(train.record)(.*?")', 'input_path: "{}"'.format(train_record_fn
ame), s)
    s = re.sub(
        '(input_path: ".*?)(val.record)(.*?")', 'input_path: "{}"'.format(test_record_fname
), s)

    # label_map_path
    s = re.sub(
        'label_map_path: ".*?"', 'label_map_path: "{}"'.format(label_map_pbtxt_fname), s)

    # Set training batch_size.
    s = re.sub('batch_size: [0-9]+',
               'batch_size: {}'.format(batch_size), s)
```

```
# Set training steps, num_steps
    s = re.sub('num_steps: [0-9]+',
               'num_steps: {}'.format(num_steps), s)

    # Set number of classes num_classes.
    s = re.sub('num_classes: [0-9]+',
               'num_classes: {}'.format(num_classes), s)
    f.write(s)

!cat {pipeline_fname}

model_dir = 'training/'
# Optionally remove content in output model directory to fresh start.
!rm -rf {model_dir}
os.makedirs(model_dir, exist_ok=True)

"""#Step 4: Run Tensorboard"""

# Launch after you have started training
# logs save in the folder "training"
%load_ext tensorboard
%tensorboard --logdir /content/drive/MyDrive/FRCNN2/Test3/training

"""# Step5: Train the model"""

!pip install lvis

!python -W ignore /content/models/research/object_detection/model_main.py \
    --pipeline_config_path={pipeline_fname} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --num_eval_steps={num_eval_steps}

!ls {model_dir}
#total steps of training ckpt-xxx.index
```

```
"""#Step6: Exporting a Trained Data

Once your training job is complete, you need to extract the newly trained inference graph,
which will be later used to perform the object detection. This can be done as follows:

###6.1 Prepare the Trained Inference Graph
Download the `frozen_inference_graph.pb` to colab lib

The path: /`content`/`models`/`research`/`fine_tuned_model`/`frozen_inference_graph.pb`
"""

import re
import numpy as np

output_directory = './fine_tuned_model'

lst = os.listdir(model_dir)
lst = [l for l in lst if 'model.ckpt-' in l and '.meta' in l]
steps=np.array([int(re.findall('\d+', l)[0]) for l in lst])
last_model = lst[steps.argmax()].replace('.meta', '')

last_model_path = os.path.join(model_dir, last_model)
print(last_model_path)
!python /content/models/research/object_detection/export_inference_graph.py \
    --input_type=image_tensor \
    --pipeline_config_path={pipeline_fname} \
    --output_directory={output_directory} \
    --trained_checkpoint_prefix={last_model_path}

!ls {output_directory}

import os

pb_fname = os.path.join(os.path.abspath(output_directory), "frozen_inference_graph.pb")
assert os.path.isfile(pb_fname), '`{}` not exist'.format(pb_fname)

!ls -alh {pb_fname}

"""
###6.2 **Export the trained result to your google drive**
Export the `training` and `fine_tune_model`folder includes `check point`, `label map` ,and
`forzen_interence_graph`
"""

# statement for copying a folder: !cp -
r <Folder path to copy> <Folder path where to paste the folder>
#  !cp -r <Folder wanna copy> <destination>
!cp -r /content/models/research/training /content/drive/MyDrive/FRCNN2/Test3/
!cp -r /content/models/research/fine_tuned_model  /content/drive/MyDrive/FRCNN2/Test3/
```

## Appendix C: Graphical User Interface Code with Object detection imported

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'main.ui'
#
# Created by: PyQt5 UI code generator 5.15.4
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


import sys
import datetime
import os
import pathlib
import glob
from PIL import Image
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import QTimer, QTime
from details import Ui_Display
from opcua import Client


try:
    url = "opc.tcp://<ip here>"
    client = Client(url)
    client.connect()
    mydonebit = client.get_node("ns=2; i=3")
    mystartbit = client.get_node("ns=2; i=4")
    mystartbit = bool(mystartbit.get_data_value())
except IOError:
    print ("Error: Not Connected")


class Ui_CellAI(object):
    def setupUi(self, CellAI):
        CellAI.setObjectName("CellAI")
        CellAI.resize(1031, 603)
        self.centralwidget = QtWidgets.QWidget(CellAI)
        self.centralwidget.setObjectName("centralwidget")
        self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox.setGeometry(QtCore.QRect(720, 110, 291, 461))
        font = QtGui.QFont()
        font.setPointSize(9)
        font.setBold(True)
        font.setWeight(75)
        self.groupBox.setFont(font)
        self.groupBox.setObjectName("groupBox")
        self.ResultLabel = QtWidgets.QLabel(self.groupBox)
        self.ResultLabel.setGeometry(QtCore.QRect(10, 20, 271, 391))
        self.ResultLabel.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.ResultLabel.setFrameShadow(QtWidgets.QFrame.Plain)
        self.ResultLabel.setText("")
        self.ResultLabel.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.
AlignTop)
        self.ResultLabel.setIndent(-1)
        self.ResultLabel.setObjectName("ResultLabel")
        self.photo_img = QtWidgets.QLabel(self.centralwidget)
        self.photo_img.setGeometry(QtCore.QRect(10, 10, 681, 521))
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSize
Policy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.photo_img.sizePolicy().hasHeightForWidth())
        self.photo_img.setSizePolicy(sizePolicy)
        self.photo_img.setFrameShape(QtWidgets.QFrame.Box)
        self.photo_img.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.photo_img.setText("")
        self.photo_img.setScaledContents(True)
        self.photo_img.setObjectName("photo_img")
        self.horizontalLayoutWidget = QtWidgets.QWidget(self.centralwidget)
        self.horizontalLayoutWidget.setGeometry(QtCore.QRect(10, 540, 681, 31))
        self.horizontalLayoutWidget.setObjectName("horizontalLayoutWidget")
        self.horizontalLayout = QtWidgets.QHBoxLayout(self.horizontalLayoutWidget)
        self.horizontalLayout.setSizeConstraint(QtWidgets.QLayout.SetDefaultConstraint)
        self.horizontalLayout.setContentsMargins(0, 0, 0, 0)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.open_file = QtWidgets.QPushButton(self.horizontalLayoutWidget)
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.open_file.setFont(font)
        self.open_file.setObjectName("open_file")
        self.horizontalLayout.addWidget(self.open_file)
        self.detect_btn = QtWidgets.QPushButton(self.horizontalLayoutWidget)
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.detect_btn.setFont(font)
        self.detect_btn.setObjectName("detect_btn")
        self.horizontalLayout.addWidget(self.detect_btn)
        self.clear_work = QtWidgets.QPushButton(self.horizontalLayoutWidget)
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.clear_work.setFont(font)
        self.clear_work.setObjectName("clear_work")
        self.horizontalLayout.addWidget(self.clear_work)
        self.view_work = QtWidgets.QPushButton(self.horizontalLayoutWidget)
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.view_work.setFont(font)
        self.view_work.setObjectName("view_work")
        self.horizontalLayout.addWidget(self.view_work)
        self.groupBox_3 = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox_3.setGeometry(QtCore.QRect(720, 10, 291, 81))
        self.groupBox_3.setTitle("")
        self.groupBox_3.setObjectName("groupBox_3")
        self.label_2 = QtWidgets.QLabel(self.groupBox_3)
        self.label_2.setGeometry(QtCore.QRect(10, 10, 41, 29))
        font = QtGui.QFont()
        font.setPointSize(9)
        font.setBold(True)
        font.setWeight(75)
        self.label_2.setFont(font)
        self.label_2.setWordWrap(False)
        self.label_2.setObjectName("label_2")
        self.label_3 = QtWidgets.QLabel(self.groupBox_3)
        self.label_3.setGeometry(QtCore.QRect(10, 40, 41, 29))
        font = QtGui.QFont()
        font.setPointSize(9)
        font.setBold(True)
        font.setWeight(75)
        self.label_3.setFont(font)
        self.label_3.setObjectName("label_3")
        self.DateLabel = QtWidgets.QLabel(self.groupBox_3)
        self.DateLabel.setGeometry(QtCore.QRect(50, 8, 279, 31))
        font = QtGui.QFont()
        font.setPointSize(9)
        font.setBold(True)
        font.setWeight(75)
        self.DateLabel.setFont(font)
        self.DateLabel.setObjectName("DateLabel")
        self.TimeLabel = QtWidgets.QLabel(self.groupBox_3)
        self.TimeLabel.setGeometry(QtCore.QRect(50, 40, 279, 29))
        font = QtGui.QFont()
        font.setPointSize(9)
        font.setBold(True)
        font.setWeight(75)
        self.TimeLabel.setFont(font)
        self.TimeLabel.setObjectName("TimeLabel")
        CellAI.setCentralWidget(self.centralwidget)
```

```
        self.retranslateUi(CellAI)
        QtCore.QMetaObject.connectSlotsByName(CellAI)

    def retranslateUi(self, CellAI):
        _translate = QtCore.QCoreApplication.translate
        CellAI.setWindowFlag(QtCore.Qt.WindowMaximizeButtonHint, False)
        CellAI.setWindowTitle(_translate("CellAI", "Fraunhofer Cell Counting App"))
        self.groupBox.setTitle(_translate("CellAI", "Result"))
        self.open_file.setText(_translate("CellAI", "Select Folder"))
        self.detect_btn.setText(_translate("CellAI", "Detect"))
        self.clear_work.setText(_translate("CellAI", "Clear"))
        self.view_work.setText(_translate("CellAI", "View "))
        self.label_2.setText(_translate("CellAI", "Date :"))
        self.label_3.setText(_translate("CellAI", "Time :"))
        self.DateLabel.setText(_translate("CellAI", "0"))
        self.TimeLabel.setText(_translate("CellAI", "0"))

        self.open_file.clicked.connect(self.OpenButton_handler)
        self.detect_btn.clicked.connect(self.detect)
        self.clear_work.clicked.connect(self.MyClear)
        self.view_work.clicked.connect(self.view_handler)

        #Date and Time
        now = QtCore.QDate.currentDate()
        current_date = now.toString('dddd dd MMMM yyyy')
        self.DateLabel.setText(current_date)

        self.timer = QTimer()
        self.timer.timeout.connect(self.displayTime)
        self.timer.start(1000)

        #Files are saved here for display
        self.image_files = []




    def displayTime(self):
        currentTime = QTime.currentTime()
        displayText = currentTime.toString('hh:mm:ss')
        self.TimeLabel.setText(displayText)

    def OpenButton_handler(self):
        self.ImageBrowse()


    def detect(self):
        # Object Detection
        import object_detection_counting
        os.system('python object_detection_counting.py')

        # Display Final Result
        text = open('final_result.txt').read()
        self.ResultLabel.setText(text)
        try:
            led = mydonebit.set_value(1)
        except NameError:
            print ("Error: Not Connected")

    def MyClear(self):
        try:
            import shutil

            self.ResultLabel.setText(" ")
            self.photo_img.setText(" ")
            try:
                led = mydonebit.set_value(0)
            except NameError:
                print ("Error: Not Connected")

            final_result_filePath = 'final_result.txt'
            result_filePath = 'result.txt'
            dir1 = "test_images/cropped_images"
            dir2 = "test_images/output_images"

            if os.path.exists(result_filePath) or os.path.exists(final_result_filePath):
                if os.path.exists(result_filePath):
                    os.remove(result_filePath)

                if os.path.exists(final_result_filePath):
                    os.remove(final_result_filePath)

                shutil.rmtree(dir1)
                shutil.rmtree(dir2)


                dir = 'test_images'
                for f in os.listdir(dir):
                    if f.endswith('.png'):
                        os.remove(os.path.join(dir, f))

        except IOError:
            print ("Error: Follow the neccessary steps and stop testing my Code")


    def view_handler(self):
        self.window = QtWidgets.QMainWindow()
        self.ui = Ui_Display()
        self.ui.setupUi(self.window)
        # CellAI.hide()
        self.window.show()



    def ImageBrowse(self):
        try:
            import fnmatch
            self._base_dir = os.getcwd()
            self._images_dir = os.path.join(self._base_dir, 'test_images')

            self._folderPath =  QtWidgets.QFileDialog.getExistingDirectory(None, "Select Fo
lder")

            for dirpath, dirs, files in os.walk(self._folderPath):
                for filename in fnmatch.filter(files, '*.png'):
                    Imagefpath = os.path.join(dirpath, filename)
                    self.image_files.append(Imagefpath)
                    self.photo_img.setPixmap(QtGui.QPixmap(self.image_files[0]))
                    self.photo_img.show()
        except IOError:
            print ("Error: Follow the neccessary steps and stop testing my Code")




if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    CellAI = QtWidgets.QMainWindow()
    ui = Ui_CellAI()
    ui.setupUi(CellAI)
    CellAI.show()
    sys.exit(app.exec_())
```

## Appendix D: Imported Object detection Code

```
######## Cell Detection & Counter Using Tensorflow-trained Classifier #########
#
# Author: Chukwudi Okerulu
# Date: 10/07/21
# Description:
# This program uses a TensorFlowtrained neural network to perform cell detection and
further code to implement cell counting.
# It loads the classifier and uses it to perform cell detection on an image.
# It draws boxes, scores, and labels around the objects of interest in the image.
## Notable websites:
## https://github.com/tensorflow/models/blob/master/research/object_detection/object_detect
ion_tutorial.ipynb
## https://github.com/datitran/object_detector_app/blob/master/object_detection_app.py


# Import packages
import os
import cv2
import numpy as np
import tensorflow.compat.v1 as tf
import sys
import pathlib
import glob
import re
import math
import fnmatch
from PIL import Image

# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")

# Import utilites
from utils import label_map_util
from utils import visualization_utils as vis_util

# Grab path to current working directory
CWD_PATH = os.getcwd()
```

```
###########################################################################################
# Sorting images into a new folder and change filename
ORIGINAL_DIR = os.path.join(CWD_PATH, 'test_images' , 'original_images')

for alpbt in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':

    if(pathlib.Path(os.path.join(CWD_PATH,'test_images', 'original_images',alpbt)))!= None
:
        for i in range(1,5):
            PATH_TO_TEST_IMAGES_DIR = pathlib.Path(os.path.join(CWD_PATH,'test_images','ori
ginal_images',alpbt + str(i)))
            TEST_IMAGE_PATHS = sorted(list(PATH_TO_TEST_IMAGES_DIR.glob("*.png")))
            #IMAGE_NAME = os.path.join(PATH_TO_TEST_IMAGES_DIR, "*.png")
            for IMAGE_NAME in TEST_IMAGE_PATHS:
                img = Image.open(IMAGE_NAME)
                complete_name = os.path.join(CWD_PATH,'test_images', alpbt + str(i) + ".png
")
                img.save(complete_name)

    else:
        break


###########################################################################################
# Cropping images into smaller images for better detection
# Crops the image into 16 grid and saves it as "new_filename" in 220*220 pixel
def crop_image(img, crop_area, new_filename):
    cropped_image = img.crop(crop_area)
    cropped_image.save(new_filename)
# The x, y coordinates of the areas to be cropped. (x1, y1, x2, y2)(left,upper,right,low)

lower_area_1_1=1954
lower_area_1_2=1734
upper_area_1_1= 1734
upper_area_1_2 = 1514
crop_areas_1 = [(100, upper_area_1_1, 320, lower_area_1_1), (520, upper_area_1_1, 740, lowe
r_area_1_1), (740, upper_area_1_1, 960, lower_area_1_1), (960, upper_area_1_1, 1180, lower_
area_1_1), (1180, upper_area_1_1, 1400, lower_area_1_1), (1400, upper_area_1_1, 1620, lower
_area_1_1), (1620, upper_area_1_1, 1840, lower_area_1_1), (1840, upper_area_1_1, 2060, lowe
r_area_1_1),
                (100, upper_area_1_2, 320, lower_area_1_2), (520, upper_area_1_2, 740, lowe
r_area_1_2), (740, upper_area_1_2, 960, lower_area_1_2), (960, upper_area_1_2, 1180, lower_
area_1_2), (1180, upper_area_1_2, 1400, lower_area_1_2), (1400, upper_area_1_2, 1620, lower
_area_1_2), (1620, upper_area_1_2, 1840, lower_area_1_2), (1840, upper_area_1_2, 2060, lowe
r_area_1_2)]

crop_areas_2 = [(300, upper_area_1_1, 520, lower_area_1_1), (520, upper_area_1_1, 740, lowe
r_area_1_1), (740, upper_area_1_1, 960, lower_area_1_1), (960, upper_area_1_1, 1180, lower_
area_1_1), (1180, upper_area_1_1, 1400, lower_area_1_1), (1400, upper_area_1_1, 1620, lower
_area_1_1), (1620, upper_area_1_1, 1840, lower_area_1_1), (1840, upper_area_1_1, 2060, lowe
r_area_1_1),
                (300, upper_area_1_2, 520, lower_area_1_2), (520, upper_area_1_2, 740, lowe
r_area_1_2), (740, upper_area_1_2, 960, lower_area_1_2), (960, upper_area_1_2, 1180, lower_
area_1_2), (1180, upper_area_1_2, 1400, lower_area_1_2), (1400, upper_area_1_2, 1620, lower
_area_1_2), (1620, upper_area_1_2, 1840, lower_area_1_2), (1840, upper_area_1_2, 2060, lowe
r_area_1_2)]

left_area_1_1=1900
left_area_1_2=2120
right_area_1_1= 2120
right_area_1_2 = 2340
crop_areas_4 = [(left_area_1_1, 100, right_area_1_1, 320), (left_area_1_1, 320, right_area_
1_1, 540), (left_area_1_1, 540, right_area_1_1, 760), (left_area_1_1, 760, right_area_1_1,
980), (left_area_1_1, 980, right_area_1_1, 1200), (left_area_1_1, 1200, right_area_1_1, 142
0), (left_area_1_1, 1420, right_area_1_1, 1640), (left_area_1_1, 1640, right_area_1_1, 1860
),
                (left_area_1_2, 100, right_area_1_2, 320), (left_area_1_2, 320, right_area_
1_2, 540), (left_area_1_2, 540, right_area_1_2, 760), (left_area_1_2, 760, right_area_1_2,
980), (left_area_1_2, 980, right_area_1_2, 1200), (left_area_1_2, 1200, right_area_1_2, 142
0), (left_area_1_2, 1420, right_area_1_2, 1640), (left_area_1_2, 1640, right_area_1_2, 1860
)]

left_area_3_1=0
left_area_3_2=220
right_area_3_1= 220
right_area_3_2 = 440
crop_areas_3 = [(left_area_3_1, 100, right_area_3_1, 320), (left_area_3_1, 320, right_area_
3_1, 540), (left_area_3_1, 540, right_area_3_1, 760), (left_area_3_1, 760, right_area_3_1,
980), (left_area_3_1, 980, right_area_3_1, 1200), (left_area_3_1, 1200, right_area_3_1, 142
0), (left_area_3_1, 1420, right_area_3_1, 1640), (left_area_3_1, 1640, right_area_3_1, 1860
),
                (left_area_3_2, 100, right_area_3_2, 320), (left_area_3_2, 320, right_area_
3_2, 540), (left_area_3_2, 540, right_area_3_2, 760), (left_area_3_2, 760, right_area_3_2,
980), (left_area_3_2, 980, right_area_3_2, 1200), (left_area_3_2, 1200, right_area_3_2, 142
0), (left_area_3_2, 1420, right_area_3_2, 1640), (left_area_3_2, 1640, right_area_3_2, 1860
)]
```

```
# Find all png images in a directory
imgList=glob.glob(os.path.join(CWD_PATH,'test_images','*1.png'))
alpbt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"    #the alphabet for the image labeling
save_path = os.path.join(CWD_PATH,'test_images','cropped_images')
if not os.path.exists(save_path):
    os.makedirs(save_path)

pic_count=0
for img in imgList:                                 #Loop through all found images
    image = Image.open(img)                         #open the image
    pic_name = alpbt[pic_count]+ "1_crop"       #for the label of the image correspond to e
ach alphabet
    pic_count +=1

    # Loops through the "crop_areas" list and crops the image based on the coordinates in t
he list
    for i, crop_area in enumerate(crop_areas_1):
        filename = os.path.splitext(img)[0]
        ext = os.path.splitext(img)[1]
        #new_filename = filename + '_' + str(i) + ext
        new_filename = os.path.join(save_path,pic_name + "_" + str(i) + ".png")
        cropped_image = image.crop(crop_area)
        cropped_image.save(new_filename)
        #crop_image(image, crop_area, new_filename)


# Find all png images in a directory
imgList=glob.glob(os.path.join(CWD_PATH,'test_images','*2.png'))
pic_count=0
for img in imgList:                             #Loop through all found images
    image = Image.open(img)                     #open the image
    pic_name = alpbt[pic_count]+ "2_crop"   #for the label of the image correspond to each
alphabet
    pic_count +=1

    # Loops through the "crop_areas" list and crops the image based on the coordinates in t
he list
    for i, crop_area in enumerate(crop_areas_2):
        filename = os.path.splitext(img)[0]
        ext = os.path.splitext(img)[1]
        #new_filename = filename + '_' + str(i) + ext
        new_filename = os.path.join(save_path,pic_name + "_" + str(i) + ".png")
        cropped_image = image.crop(crop_area)
        cropped_image.save(new_filename)
        #crop_image(image, crop_area, new_filename)

# Find all png images in a directory
imgList=glob.glob(os.path.join(CWD_PATH,'test_images','*3.png'))
pic_count=0
```

```
for img in imgList:                         #Loop through all found images
    image = Image.open(img)                 #open the image
    pic_name = alpbt[pic_count]+ "3_crop"   #for the label of the image correspond to each 
alphabet
    pic_count +=1

    # Loops through the "crop_areas" list and crops the image based on the coordinates in t
he list
    for i, crop_area in enumerate(crop_areas_3):
        filename = os.path.splitext(img)[0]
        ext = os.path.splitext(img)[1]
        #new_filename = filename + '_' + str(i) + ext
        new_filename = os.path.join(save_path,pic_name + "_" + str(i) + ".png")
        cropped_image = image.crop(crop_area)
        cropped_image.save(new_filename)
        #crop_image(image, crop_area, new_filename)


#Find all png images in a directory
imgList=glob.glob(os.path.join(CWD_PATH,'test_images','*4.png'))
pic_count=0
for img in imgList:                         #Loop through all found images
    image = Image.open(img)                 #open the image
    pic_name = alpbt[pic_count]+ "4_crop"   #for the label of the image correspond to each 
alphabet
    pic_count +=1

# Loops through the "crop_areas" list and crops the image based on the coordinates in the l
ist
    for i, crop_area in enumerate(crop_areas_4):
        filename = os.path.splitext(img)[0]
        ext = os.path.splitext(img)[1]
        #new_filename = filename + '_' + str(i) + ext
        new_filename = os.path.join(save_path,pic_name + "_" + str(i) + ".png")
        cropped_image = image.crop(crop_area)
        cropped_image.save(new_filename)
        #crop_image(image, crop_area, new_filename)
```

```
###########################################################################################
# Code for Object detection and counting
# Name of the directory containing the object detection module we're using
MODEL_NAME = 'inference_graph'
IMAGE_NAME = 'image1_0.png'

# Grab path to current working directory
CWD_PATH = os.getcwd()

# Path to frozen detection graph .pb file, which contains the model that is used
# for object detection.
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,'training','labelmap.pbtxt')

# Path to image
# PATH_TO_IMAGE = os.path.join(CWD_PATH,'test_images', IMAGE_NAME)

# If you want to test the code with your images, just add path to the images to the TEST_IM
AGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = pathlib.Path(os.path.join(CWD_PATH,'cropped_images'))
#Make an array that stores all images
#TEST_IMAGE_PATHS = sorted(list(PATH_TO_TEST_IMAGES_DIR.glob('*.png')))


# Number of classes the object detector can identify
NUM_CLASSES = 2
k = 0

# Load the label map.
# Label maps map indices to category names, so that when our convolution
# network predicts `1`, we know that this corresponds to `white`.
# Here we use internal utility functions, but anything that returns a
# dictionary mapping integers to appropriate string labels would be fine
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_
CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

# Load the Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')
    sess = tf.Session(graph=detection_graph)



imgList=glob.glob(os.path.join(CWD_PATH ,'test_images','cropped_images','*.png'))
for PATH_TO_IMAGE in imgList:
    # Define input and output tensors (i.e. data) for the object detection classifier
    #print(PATH_TO_IMAGE)
    # Input tensor is the image
    image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

    # Output tensors are the detection boxes, scores, and classes
    # Each box represents a part of the image where a particular object was detected
    detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')

    # Each score represents level of confidence for each of the objects.
    # The score is shown on the result image, together with the class label.
    detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
    detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')

    # Number of objects detected
    num_detections = detection_graph.get_tensor_by_name('num_detections:0')

    # Load image using OpenCV and
    # expand image dimensions to have shape: [1, None, None, 3]
    # i.e. a single-column array, where each item in the column has the pixel RGB value
    image = cv2.imread(PATH_TO_IMAGE)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image_expanded = np.expand_dims(image_rgb, axis=0)

    # Perform the actual detection by running the model with the image as input
    (boxes, scores, classes, num) = sess.run(
        [detection_boxes, detection_scores, detection_classes, num_detections],
        feed_dict={image_tensor: image_expanded})

    # Draw the results of the detection (aka 'visulaize the results')

    vis_util.visualize_boxes_and_labels_on_image_array(
        image,
        np.squeeze(boxes),
        np.squeeze(classes).astype(np.int32),
        np.squeeze(scores),
        category_index,
        use_normalized_coordinates=True,
        line_thickness=2,
        min_score_thresh=0.60)

    #save to same folder as data input
    Output_image_path = os.path.join(CWD_PATH ,'test_images','output_images')
    k += 1
    if not os.path.exists(Output_image_path):
        os.makedirs(Output_image_path)

    Output_image_path = os.path.join(CWD_PATH ,'test_images','output_images', ' ')
    img = Image.fromarray(image)
    img.save(Output_image_path + str(k) +'.png')


    # All the results have been drawn on image. Now display the image.
    #cv2.imshow('Object detector', image)

    # Press any key to close the image
    #cv2.waitKey(0)

    # Clean up
    #cv2.destroyAllWindows()



############################################################################################
# Code to combine the result of white and black cell count and output final result txt.file
# Accessing the result.txt file
PATH_TO_RESULT = os.path.join(CWD_PATH, 'result.txt')
PATH_TO_FINAL_RESULT = os.path.join(CWD_PATH, 'final_result.txt')
result_file = open(PATH_TO_RESULT,"r")

# Initialization of final result file
final_result_file = open(PATH_TO_FINAL_RESULT, "w")
final_result_file.close()

# Reading each line on the result.txt and put it into an array-list
lines = result_file.readlines()
# Initialization of the amount of cells
white_amount = 0
black_amount = 0

# Opening the final result file to add data
final_result_file = open(PATH_TO_FINAL_RESULT, "a")


# Iterating through each line of result.txt
save_path = os.path.join(CWD_PATH,'test_images','cropped_images')
for i in range(0,len([name for name in os.listdir(save_path) if os.path.isfile(os.path.join
(save_path, name))])): # The range need to be change according to desired amount of cropped
 images per complete image
    #print(lines[i])
    single_line = lines[i]
    weiss = re.search('\'L_Cell\': (\d+)', single_line, re.IGNORECASE)
    schwarz = re.search('\'D_Cell\': (\d+)', single_line, re.IGNORECASE)
    #leer = re.search('{}', single_line)
    if single_line.find('L_Cell') >= 0 and single_line.find('D_Cell') >= 0 :
        #print(weiss.group(1))
        white_amount += int(weiss.group(1))
        #print(schwarz.group(1))
        black_amount += int(schwarz.group(1))
    elif single_line.find('L_Cell') >= 0:
        #print(weiss.group(1))
        white_amount += int(weiss.group(1))
    elif single_line.find('D_Cell') >= 0:
        #print(schwarz.group(1))
        black_amount += int(schwarz.group(1))
    else:
        black_amount += 0
        white_amount += 0

# Writing the final result into the .txt data file
total_amount = white_amount + black_amount
white_per = (white_amount/total_amount)*100
black_per = (black_amount/total_amount)*100

final_result_file.write("Amount of Living cells : " + str(white_amount) +
                        "\n" + "Amount of Dead cells : " + str(black_amount) +
                        "\n\n" + "Average of Living cells : " + str(white_amount/4) +
                        "\n" + "Average of Dead cells : " + str(black_amount/4) +
                        "\n\n" + "Percentage of Living cells : " + str(round(white_per)) +
                  "%" + "\n" + "Percentage of Dead cells : " + str(round(black_per)) + "%")
final_result_file.close()
result_file.close()
```