UNIVERSITY OF IOANNINA
DEPARTMENT OF INFORMATICS &
TELECOMMUNICATIONS

UNDERGRADUATE THESIS JOURNAL

Object Detection for Low Light Images

Dimitrios Mpouziotas

Supervisor Petros Karvelis

March 25, 2024

OBJECT DETECTION FOR LOW LIGHT IMAGES

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.Επίσης δηλώνω πως η εφαρμογή chatGPT, δεν χρησιμοποιήθηκε ποτέ με σκοπό την παραγωγή κειμένου ή κώδικα σε αυτήν την πτυχιακή, εκτός από την αναζήτηση αναφορών, άρθρων, βάσεων δεδομένων, επιστημονικών άρθρων.

Acknowledgements

The development of this thesis has been a challenging task, and I am grateful for the guidance and assistance of the ever growing Darknet community, which improved the technical quality of this thesis drastically. I would like to express my sincere gratitude to my professor and thesis supervisor, Petros Karvelis, whose mentorship elevated my understanding and skills in Computer Vision and Data analysis and enabled me to reach new heights and potential.

Finally, I am deeply grateful to my parents for their unwavering support and presence in my life, especially as I approached the ending point of my academic journey as a bachelors student, on my graduation day.

ABSTRACT

Object Detection is a computer vision method that was developed for detecting objects in images and videos. Although it has surpassed human performance and it is considered practically solved, most object detectors struggle to detect objects in sub-optimal lighting conditions. Various techniques have been developed in order to enhance low light images to a much better state. This thesis is about advancing further into the topic of object detection and computer vision, as well as determining a conclusion based on the well known object detector YOLO (You Only Look Once). A statistical analysis of YOLOv4's [1] performance on ExDark [2], the largest low light image dataset. An exploration on various low light image enhancement techniques such as GC [3], HE [4], and Kindling the Darkness aka KinD [5], in order to increase YOLOv4's performance. A deeper understanding of the problem caused within low light conditioned images by using various data visualization techniques, such as contour maps, sobel derivatives, 3D image plotting and intensity value histograms.

Figure 1: Image with 2 target detections and 1 prediction **(ExDark Image 03215)**

This thesis operates as an academic research resource and aims to find a conclusion on the performance of the YOLOv4 object detector using several low light image enhancement techniques.

ΠΕΡΙΛΗΨΗ (Abstract in Greek)

Η αναγνώριση αντικειμένων είναι μια μέθοδος μηχανικής όρασης η οποία αναπτύχθηκε με σκοπό την ανίχνευση ορισμένων αντικειμένων πάνω σε εικόνες ή/και βίντεο. Ακόμα και αν έχει ξεπεράσει την ικανότητα του ανθρώπου και θεωρείτε σχεδόν λυμένο ως πρόβλημα, τα περισσότερα μοντέλα αναγνώρισης αντικειμένων δυσκολεύονται στην ανίχνευση αντικειμένων σε εικόνες υποβαθμισμένης φωτεινότητας (Χαμηλής φωτεινότητας). Διάφορες τεχνικές υλοποιήθηκαν με σκοπό την ενίσχυση των εικόνων, σε καλύτερη κατάσταση και φωτεινότητα. Αυτή η πτυχιακή έχει ως στόχο την περαιτέρω πρόοδο στο θέμα της αναγνώρισης αντικειμένων και μηχανικής όρασης, καθώς και τον καθορισμό ενός συμπεράσματος χρησιμοποιώντας το περίφημο μοντέλο αναγνώρισης αντικειμένων **YOLO (You Only Look Once)**. Μια στατιστική ανάλυση της απόδοσης του **YOLOv4** στο **ExDark**, το μεγαλύτερο **dataset** με εικόνες χαμηλής φωτεινότητας. Μια εξερεύνηση σε διάφορους αλγόριθμους ενίσχυσης εικόνων χαμηλής φωτεινότητας, όπως το **GC**, **HE** και **Kindling the Darkness** επίσης γνωστό ως **KinD**, με σκοπό την αύξηση της απόδοσης του μοντέλου **YOLOv4**. Μια βαθύτερη κατανόηση στο πρόβλημα των εικόνων χαμηλής φωτεινότητας χρησιμοποιώντας διάφορες τεχνικές οπτικοποίησης δεδομένων, όπως **sobel** παραγώγους, χάρτες περιγραμμάτων, οπτικοποίηση φωτεινότητας εικόνων σε **3D** διαγράμματα και ιστογράμματα τιμών εντάσεων.

Η πτυχιακή αυτή λειτουργεί ως ακαδημαϊκός επιστημονικός πόρος, που έχει ως στόχο να βρει ένα συμπέρασμα στην αύξηση της απόδοση του μοντέλου **YOLOv4**, χρησιμοποιώντας διάφορες τεχνικές ενίσχυσης φωτεινότητας εικόνων.

Contents

List of Figures

1 Introduction

Computer Vision has been a topic of intense study and growing interest. With computer vision, we can build autonomous systems that can complete tasks without any human intervention. One of those tasks is object detection, which was developed for computers to simulate and mimic the human visual system [10].

Over the years, numerous systems for object detection have been proposed [11] and one of the most well-known systems is YOLO models (You Only Look Once) [12]. YOLO is an Object Detection algorithm, that uses the architecture of Convolutional Neural Networks [13] to detect objects in colored images. YOLO is a state-of-the-art object detector, which was developed and gradually improved to classify objects for images. It can detect a number of objects at high performance and precision.

There are several YOLO models maintained within the Darknet framework [12]. Darknet is fast and easy to install and supports CPU and GPU computing. In this work, YOLOv4 (version 4) [1, 14] will be an important tool for the analysis of this thesis in order to determine a conclusion.

Although object detectors report state-of-the-art accuracy in various datasets [11], often images are captured under suboptimal lighting conditions [15].

By enhancing images in real life problems such as self-driving cars with object detection systems, we can boost the models' performance to detect objects even in bad weather or at night [16], [17]. This is to minimize any possible errors that may occur during those events and reduce accidents that could be caused by the lack of luminance in the image [18]. Systems like these are required to be fast, accurate, and efficient to minimize the margin of error. In order to face this problem and decrease the margin of error caused by the sub-optimal lighting conditions for the object detector, this thesis will include various enhancement methods. This way, the object detector, will be able to achieve higher detection rate and precision, while at the same time maintaining high performance.

Another major problem with low light and especially on extremely low light images, is that when applying several enhancements to it, the result will be a rather noisy image, or even worse it may distort the image, making it nearly impossible to detect any object within it. However, a wide variety of methods concerning standard image processing methods or Deep Learning based methods exist in the literature of computer vision and image processing. In Figure 2, a low light image is displayed and the result of an object detection method is also depicted. It is clear that there are two cats. The cat on the left cannot be detected by the object detection system (namely the YOLOv4), even though in the human eye, it is obvious. The reason the model is unable to detect the other object is due to the outline of the cat's head not being visible enough, only the lower part of its body can be seen.

A solution to this problem could be by continuing the training of the model with darker than usual images, but this solution is very time-consuming and hard to achieve. Instead, we could employ image enhancement methods to the images and solve this issue.

This thesis involves analyzing YOLOv4 and how it operates under low light imaging conditions. It also involves algorithms that enhance images to a point where the model can reach

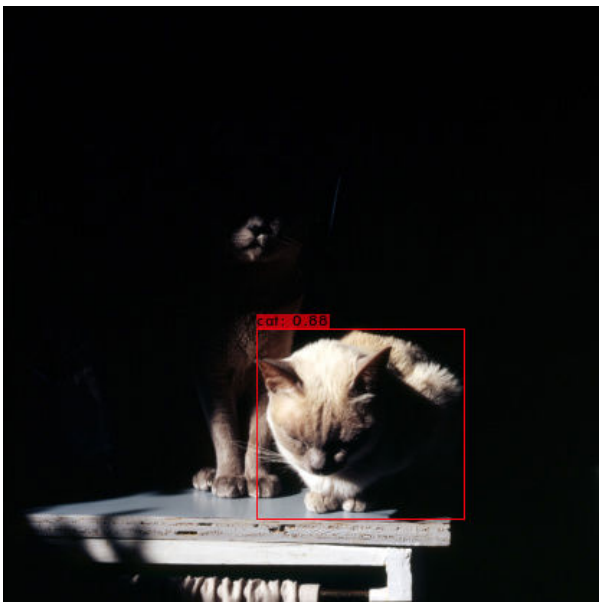


Figure 2: Faulty detection using an object detector.

its full potential and decrease its margin of error. These methods are namely Gamma Correction [19], Histogram Equalization [20] and Kindling the Darkness (KinD) [5] which they are an important tool, in order to enhance low light images and measure the effect in YOLOv4's results.

This thesis also includes analyzing the evaluation results of YOLOv4 and how it operates on images with sub-optimal lighting conditions. YOLOv4 was evaluated on various low light enhancement algorithms and the evaluation results were extracted, in order to calculate YOLOv4's performance. Various data visualization techniques were used to visually identify the differences during the enhancement of an image, as well as any distortion generated during the enhancement.

During the development of this thesis a strong introductory on the Perceptron and Convolutional Neural Networks was included, as well as a brief study on the topic of training and evaluating the YOLOv4 model. Various datasets were created on an unusually difficult task, in order to solve an object detection problem. During the expansion of each dataset, various methods were researched and applied into the datasets to minimize any possible errors between each class and negative classes [21]. This is to acknowledge how YOLOv4 visually learns from images and how YOLOv4 struggles to detect certain objects in low light conditioned images.

1.1 The Goals and Contribution

This topic is being researched due to its real world uses and several other inspirations of object detection systems (e.g. Tesla's Autopilot object detection system, face recognition systems,

object detection in games, satellite and CCTV camera systems). The study of computer vision is on-par with data science and general machine learning tasks, that can contribute to the scientific community. This study is in development and is still being researched actively today. In this topic of this research, YOLOv4's behavior will be analyzed using several groundtruth information given by the ExDark dataset [2] with several low light enhancement methods and algorithms. The analysis behind this research is to increase the detection rate of the model and the quality and luminance of the images in order to allow YOLOv4 to reach its maximum potential.

The approach of this research in this thesis will first include various tests of low light images on the YOLOv4 model. After a variety of information has been extracted from those low light images, they will be enhanced and tested again with multiple enhancement methods and algorithms. This way, a conclusion will be extracted and determine which enhancement method and algorithm performs better or worse.

The research topic of this thesis will contribute to the scientific community by finding and analyzing methods related to object detection for low light images. Several examples of the importance of this research topic, can be the following.

1.1.1 Self-driving Autopilot Cars

Tesla self-driving cars, have been the center of attention for electric cars. Although they have proven to be state-of-the-art systems, often times accidents were caused due to errors given by the autopilot model. Those issues were caused due to objects not being detected as intended. This isn't necessarily because the model wasn't trained well, but in several conditions where the lighting is not enough for the object detector, in short during the night an autopilot system will struggle to actively and accurately detect objects in those conditions.

This research will assist in minimizing the margin of error and number of accidents caused by the self-driving car models not detecting objects as intended, as well as allow the model to reach its maximum potential.

1.1.2 Satellite & aircraft systems

As technology improved, computer vision has advanced even to the military and space industries. Various systems are used in order to detect projectiles and predict incoming impacts. This research will contribute to these systems in suboptimal lighting conditions, to evade expensive damages dealt by the impact.

1.1.3 Face Recognition Systems

Public forces like the military or the police regularly use systems to identify suspects in question. Public forces use systems on cameras such as CCTV in order to identify a person, movement, objects, or crime [22].

Regularly, those forces use enhancement algorithms in order to boost the quality or the luminance of an image. Once again, this research will be able to contribute a conclusive analysis for those systems to increase their precision and detection rate.

1.2 The Darknet Framework

Darknet[12] is a framework that was originally developed by Joseph Redmon. Redmon graduated in Middlebury, Vermont, and majors in Computer Science and Mathematics. Darknet today is still regularly used daily, by many users, since it contains a variety of state-of-the-art models for any user to install in their computer, in order to train or evaluate a neural network. As of time of writing Stephane Charette is the current owner of the Darknet GitHub repository [23], as well as various other tools that provide support to Darknet. The expansion tools that provide extra support to Darknet, are DarkHelp [6] and DarkMark [24] [25], aka the newly founded name DarkSuite (Darknet + DarkHelp + DarkMark). Darknet can be installed in Linux and Windows. Stephane Charette provides a wide variety of information on how to operate Darknet including DarkHelp and DarkMark [26].

As of the time of writing, YOLO version 7 has been developed and according to the graphs given by Alexey B. Figure 3 it surpasses in terms of performance a lot of state-of-the-art object detectors.

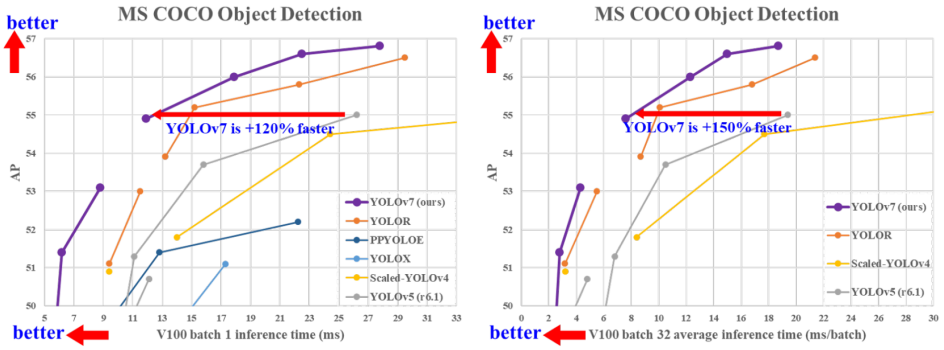


Figure 3: A recent performance graph comparison of various YOLO object detector versions up to YOLOv7

With the appropriate hardware and software installed into the system, training the yolov4-tiny model, can take up to 2 hours. Figure 4 previews the training time YOLOv4-tiny needs in certain scenarios.

	original 4608x3456 JPG images	4608x3456 JPG images, quality=75	800x600 JPG images, quality=75	416x416 JPG images, quality=75
Darknet compiled to use GPU + OpenCV	10 iterations: 42.26 seconds 10K iterations: 11h 44m	10 iterations: 35.27 seconds 10K iterations: 9h 47m	10 iterations: 6.90 seconds 10K iterations: 1h 55m	10 iterations: 6.76 seconds 10K iterations: 1h 53m
Darknet compiled to use GPU + OpenCV, **but using PNG images** instead of JPG	n/a	10 iterations: 80.70 seconds 10K iterations: 22h 25m	10 iterations: 6.93 seconds 10K iterations: 1h 56m	10 iterations: 6.71 seconds 10K iterations: 1h 52m
Darknet compiled to use GPU, but *without* OpenCV	10 iterations: 113.31 seconds 10K iterations: 31h 29m	10 iterations: 106.56 seconds 10K iterations: 29h 36m	10 iterations: 9.19 seconds 10K iterations: 2h 33m	10 iterations: 7.70 seconds 10K iterations: 2h 8m
Darknet compiled for **CPU** + OpenCV *(no GPU)*	10 iterations: 532.86 seconds 10K iterations: > 6 days	10 iterations: 527.41 seconds 10K iterations: > 6 days	10 iterations: 496.47 seconds 10K iterations: > 5 days	10 iterations: 496.03 seconds 10K iterations: > 5 days

Figure 4: Yolov4-tiny training time using a 2080 Ti graphics card.

In this thesis, Darknet is the primary tool in order to operate the YOLOv4 model and apply various tasks such as model training or evaluation on images and videos.

1.3 An Introductory to OpenCV

OpenCV is an image processing library that provides several tools for image & video processing tasks. It was originally built for Image Processing developers and is a great tool for performing Computer Vision tasks. Today it is considered to be the most famous image processing tool for developers primarily in the topic of programming. OpenCV is open-source and was developed by Intel[7]. The library is available in the programming languages C/C++, Python and Java.

OpenCV is an important tool for the analysis of this thesis, in order to complete certain tasks such as image enhancement, image processing and data visualization. OpenCV has a variety of installation guides for each platform (Windows, Linux) and programming language. Most tools in this thesis were developed using this library in the Python programming language.

1.4 Image Processing Using OpenCV

Image Processing is the application of tools and algorithms on images in order to apply various transformations. Image processing allows for automating various tasks on images, as well as the first layer of a Convolutional Neural Network, which requires an image to be ”convoluted”. OpenCV, as mentioned in Section 1.3, is regularly used in image processing by developers to automate image processing applications.

Various image processing applications, are noise reduction, data augmentation [27], image enhancement [28], image resizing and sampling [29], as well as image analysis or data visualization.

Some of the more crucial methods mentioned during the development of this thesis is low light image enhancement, data visualization and noise reduction methods. The thesis author developed various algorithms using OpenCV and the Python programming language and were included within the appendix of this thesis.

2 The Low Light Condition Problem

Low light conditions in an image is the lack of color and brightness, which can negatively affect the visibility and detail of certain objects within the image. Low light conditions can also result in bad detections for the object detector, as well as reduce the number of expected objects to be predicted. In most low light conditioned images, degradation and noise is especially evident, making it difficult for any model to detect objects within that image. In images with such lighting conditions, noise and lack of brightness could cause the differences between objects and shapes to be less discrete, resulting in less detections for the object detector.

In real-world problems for object detection, we need to assume that an object detector needs to be able to detect objects in images with sub-optimal lighting conditions. In order to

assist object detectors to accurately detect objects within low light images, we can adjust the brightness of the image with several methods by up-lifting the brightness of the image.

In order to reach this state, understanding and visualizing an image's low light conditions is an important step to, better understand the low light conditions of an image.

2.1 Problems & Degradation of a Low Light Image

Low light conditions in an image may degrade an image's quality as well as detail.

2.2 Data Visualization for Low Light Images

By visualizing the data of an image, numerous information can be concluded that can assist in understanding the low light condition problem. There are several methods to visualize or measure an image's low light conditions. In this section, several methods will be demonstrated and implemented. With those methods, information that cannot easily be observed with the human eye will be highlighted and clarified.

In order to visualize the data of a low light image, understanding the structure of an image is an important step to determining that factor.

2.2.1 Image Structure

An image consists of a 2-Dimensional array that contains intensity values ranging from 0 to 255. Each image contains a certain color format with a set number of channels. There are various color formats that determine the number of channels and information of an image, the most common color format is RGB. The RGB color format consists of 3 different color channels, Red, green and blue channels. This indicates that an image can be a 2-Dimensional array consisting of multiple color channels. In other words, a 500x500 RGB image contains a total of 500x500x3 intensity values and is equal to 250,000 total pixels but 750,000 total information to process, Figure 5. In the other hand, the total color variations of a pixel in an RGB image is 16,777,216 different colors, which is the total intensity value, powered by the total color channels of the image. The human eye, is estimated to perceive up to 7 million color variations [30].

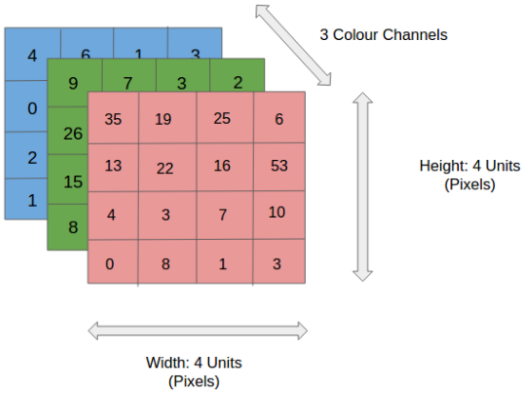
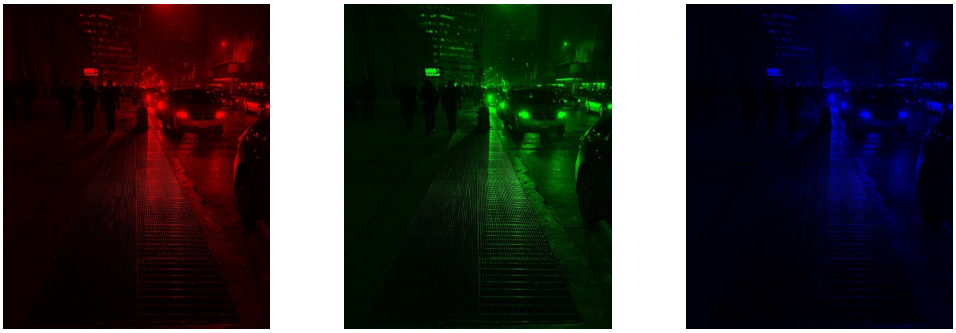


Figure 5: The structure of an RGB Image.

By using OpenCV, we can split the image into its 3 color channel variants and view each color channel, Figure 6 and apply various changes to them.



(a) Red Channel (b) Green Channel (c) Blue Channel

Figure 6: The color channels of an Image. **(ExDark Image 06365)**

In image processing, we can also convert the image from the RGB color format, into the CIELAB color space [31]. The CIELAB color space simulates the human vision

2.2.2 Introduction to Image Histograms

A method to visualize an image's low light conditions is by using Histograms [32]. The information that can be displayed in a histogram is the total count of intensity values, including the contrast if all the color channels are displayed within the histogram, Figure 7.

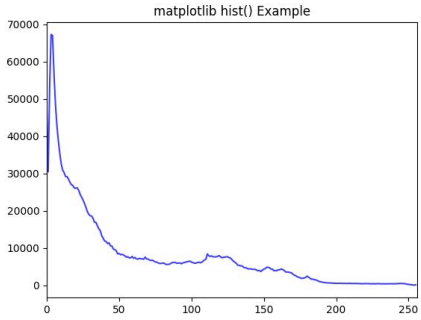


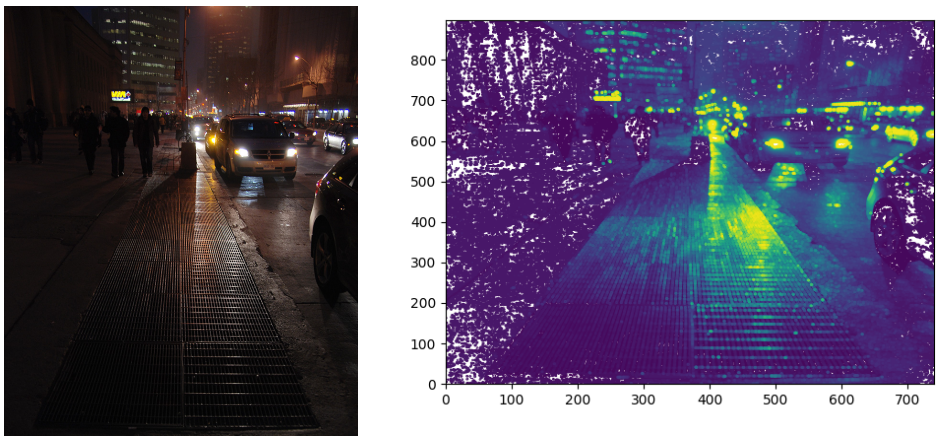
Figure 7: Histogram of an Image.

The x-axis represents the intensity values that ranges from 0 to 255. The Y-axis represents the total count of the same intensity values within the image. This will allow us to understand the overall brightness of the image, including the most common intensity values.

Histograms are an important tool in various implemented algorithms and are of use in the data analysis of this thesis.

2.2.3 Image Contour maps

By using contours, we can visualize the limit of what an object detector can use as information [33, 34]. Contours represent the outline of shapes of objects. Contours are used to find differences of objects in images. Contours are basically curved lines drawn around the shapes of objects. A display of an image can be seen in Figure 8



(a) ExDark Image

(b) Contour Map Result

Figure 8: The contour map of a low light image **(ExDark Image 06365)**

By generating the contour map of a low light image, we can observe that in certain areas of the contour map, due to the lack of brightness and color variations from the original image, the people on the left side of the contour map are not visible enough, Figure 8.

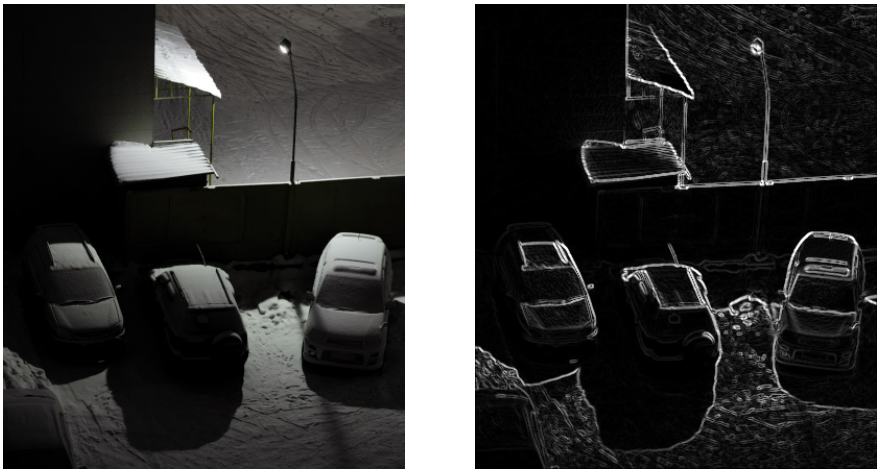
In the contour map, we can observe the brightest spots in the image. A spot where there is very limited light and color variation will hardly contain any contours around those areas. Contour maps are one of the methods used in simulating the limitations of the visual system of an object detector.

2.2.4 Sobel Derivatives

Sobel derivatives is another method to determine the edge of shapes within an image [35]. OpenCV provides a function, called Sobel, that uses derivatives in order to detect and draw the outlines of shapes within an image. This algorithm works by processing a kernel (Most commonly a 3x3 sized kernel) and detecting the edges.

Mathematically, this algorithm approximates the differences between the pixels of an image, in other words, it calculates the rate of change of the pixels within the kernels by using derivatives. In order to draw the image appropriately, different gradient kernels are used for both the x and y axis.

The result from this algorithm is displayed in Figure 9



(a) ExDark Image (b) Sobel Derivative Image

Figure 9: The Sobel Derivative result of a low light image **(ExDark Image 02453)**

Sobel derivatives simulate the visual limit of the object detector for low light images and conducts an insight to the low light condition problem.

2.2.5 3D Image Plotting

By plotting an image into a 3D space, we can visualize a variety of information such as the highest point of intensity values the brightest and darkest spots of the image and where it is located. In order to graph a 3D image, the X and Y axis are used to fit the image, the Z axis is used to display the intensity value of the image colored in a heatmap like color-spectrum. The result of a 3D plotted image is displayed in Figure 10

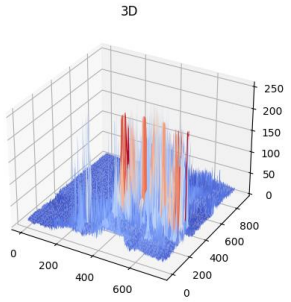
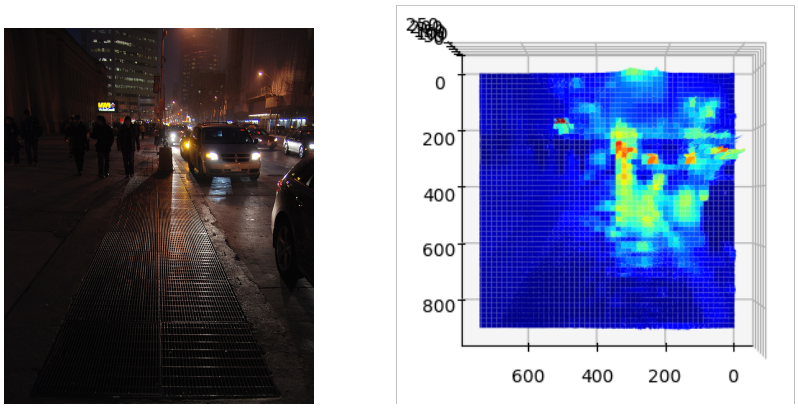


Figure 10: 3D Image Plot

In Figure 11 the 3D plot of an image is displayed on a top-down view. The Z axis was colored based on the intensity value, the higher the intensity value, the more right in the JET color map (A color map regularly used in Heat maps). In short, the brighter the pixel, the more red the location of the graph will be.



(a) ExDark Image (b) 3D Plot

(c) JET Color spectrum: The higher the intensity value, the more right in the color spectrum

Figure 11: Top-down view of a 3D plot of an image. **(ExDark Image 06365)**

The 3D plot of an extremely low light image would contain mostly blue colored pixels. This way, we can see the difference between the highest and lowest intensity values of an image. Fundamentally, this algorithm produces a heatmap of an image.

2.3 Determining The Light Levels of an Image

Determining the light level of an image is important in order to determine whether a model should enhance the image with a higher or lower value. This can contribute to conditions where a model can dynamically determine whether the image should or shouldn't enhance an image, or even dynamically change values that adjust the lighting of an image.

There are several methods to identify an image's light levels. The most generic method is to calculate the means of the total intensity values of an image. Another effective method to determine the light level of an image is by using histograms. We can use various histograms to see various details about which areas are the least or the most bright in an image.

In this thesis, the images will be identified into four categories:

Normal Lighting
Low Lighting
Very Low Lighting
Extremely Low Lighting

All the algorithms that determine the light level of an image were implemented using Python and OpenCV.

2.3.1 The Means Algorithm

A simple and effective method to calculate the light level of an image, is by calculating the means of intensity values within the image, Equation 1.

$$Means = \frac{\sum_{i=1}^{W \cdot H} Vi}{W \cdot H} \tag{1}$$

1: $W \cdot H$ is the total pixels being processed, `Vi` is the Intensity value `i`. The result is the average intensity value of a grayscale image

By converting the image into its gray counterpart, we can then calculate the total average of intensity values. Each category contains a set bandwidth, or threshold, that will determine the light level of the image based on the Means result.

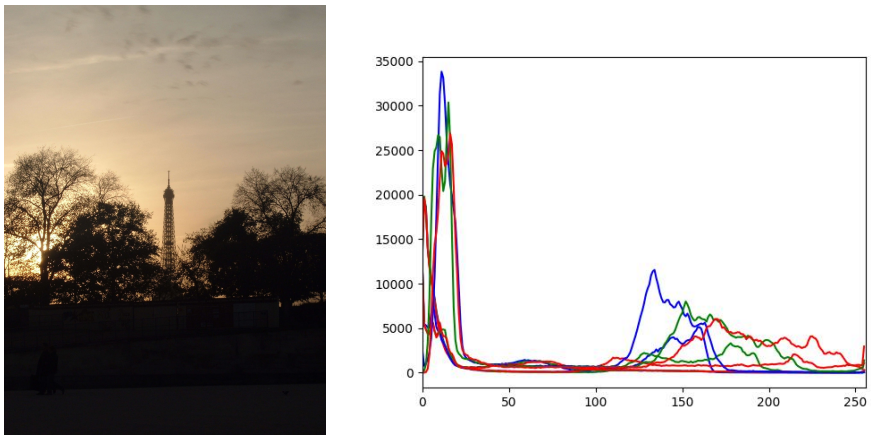
This algorithm may be an efficient way to calculate the light level of an image. A lot of the times it may identify images where most of the objects are located in a dark area of the image and in another area the image is bright, this can affect the classification result of the algorithm. For e.g. Figure 12.

Figure 12: An image that contains both low and normal light level areas. **(ExDark Image 02594)**

The image, displayed in Figure 12 was classified as **Normal Lighting** and the total means in that image is **125.0**. This algorithm may very well not be accurate when it comes to the most important information that could be located in the darkest spots of the image, as seen in Figure 12.

2.3.2 Determining the Light Level using Histograms

Another method to determine an image's low light condition levels is by using histograms. The histogram of an image will be able to give information such as the brightness and contrast of an image, including where the darkest and brightest spots of the image are located. An example of an image's Histogram is shown in Figure 13.

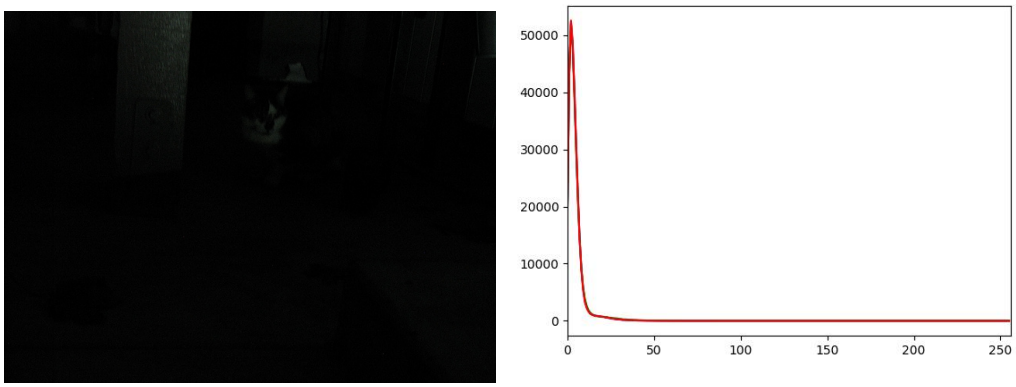


(a) ExDark Image (b) Histogram of Image

Figure 13: The histogram of a low light image **(ExDark Image 06756)**

As seen in Figure 13 the image contains a decent amount of light, the histogram will have a fair distribution between the range of intensity values.

A darker than usual image would contain a really high count of intensity values on the most left side of the histogram. An example of this can be displayed in Figure 14



(a) ExDark Image (b) Histogram of Image

Figure 14: The histogram of a low light image **(ExDark Image 06756)**

In order to calculate the light level of the image using histograms is by attempting to calculate the biggest cluster of intensity values and compare it to the rest of the image or the rest of the clusters within that image. The algorithm that can be used in this case is K-Means Clustering or several other machine learning algorithms.

3 Convolutional Neural Networks, the Tool of Object Detection

Most object detection models use CNN (Convolutional Neural Networks) in order to classify objects within an image. CNNs work by feeding images into the model, it will then process the image and extract information from it, Figure 15. The most visible and most important results are the bounding box of a detected object and the classified type. The bounding box is the square or rectangle that is generated to represent the location of a detected object in an image. Next to the bounding boxes, the model will also include various other information, such as the confidence of the model and the class type. The Confidence value represents how determined the model is based on what it has classified. In many cases, a model can have 90 to 100% confidence on an **x** object, but in reality it could be something completely different from what the model is predicting. The confidence value will also determine which object is the class type that is displayed in the image.

In order to understand how Convolutional Neural Networks work, we will need to understand several other interpretations, such as what linear and non-linear problems are. The perceptron

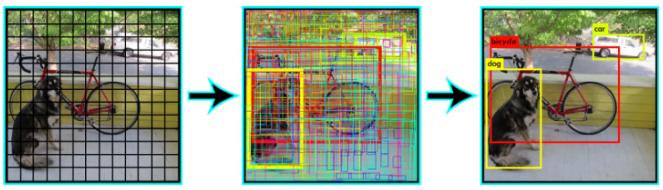


Figure 15: A visualization of an image being processed by a Convolutional Neural Network

3.1 Linear & Non-linear Problems

It is important to understand the difference between linear and non-linear problems in order to summarize object detection and Convolutional Neural Networks. Linear and non-linear problems are terms used to explain the complexity of a problem which is created by the features of a dataset for a neural network to solve. A linear problem, graphically, is when two clusters of data can be linearly separated by drawing a straight line between the two clusters, Figure 16. Otherwise, if the two clusters cannot be separated by drawing a straight line, then the problem is considered non-linear, Figure 17.

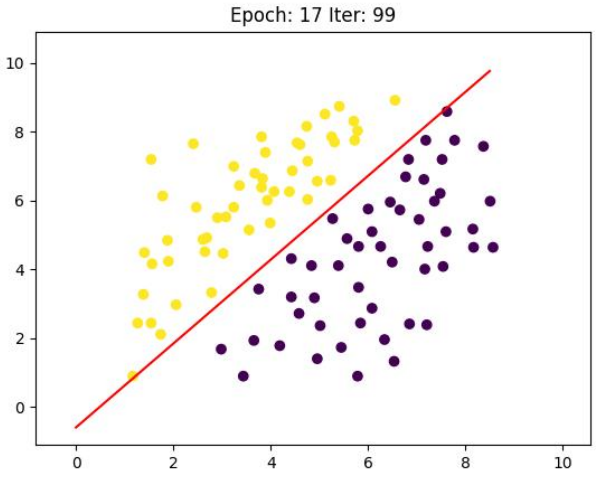


Figure 16: A visualization using graphs of a linearly separable problem.
The example image is produced by the Perceptron algorithm [8]

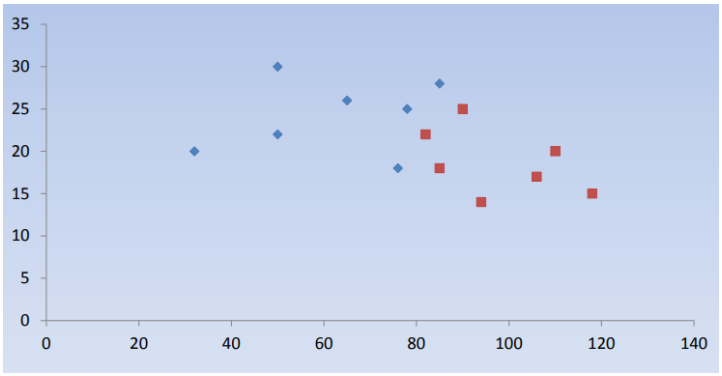


Figure 17: A visualization using graphs of a non-linearly separable problem.
An example of two clusters that cannot be linearly separated by drawing a straight line.

There is a method to convert a non-linearly separable dataset into a linearly separable dataset. This can be done by increasing the number of dimensions of a problem. In other words, e.g. if a dataset with two features X1, X2 and K as the output is a non-linearly separable dataset, we can add a new feature X3 in the dataset, this way we will increase the dimensions up to 3 based on the number of features, Figure 18.

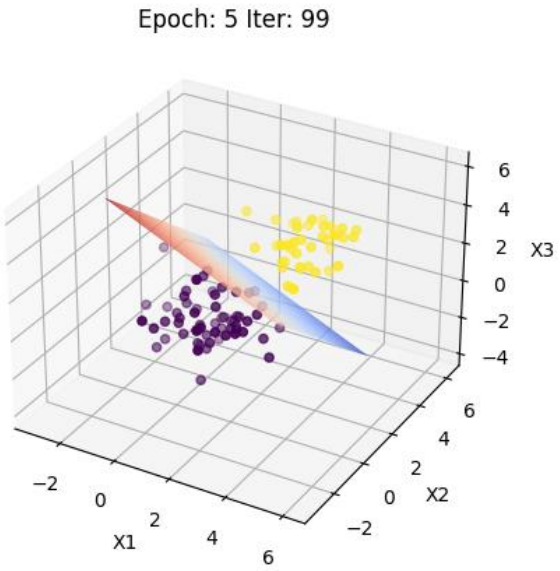


Figure 18: A visualization using graphs of a linearly separable problem using hyperplanes.
A 3D (3 features) example of class types that can be linearly separated by drawing a hyperplane.

3.2 The Perceptron

The Perceptron is an Artificial Neural Network that can be trained to predict an output, based on the inputs. It will allow us to acknowledge and understand how the classification layer of

the Convolutional Neural Network works. The perceptron and generally all neural networks were inspired by the ideology of the human brain and its structure. The perceptron is an artificial implementation of a biological human neuron.

The structure of a perceptron is separated in three phases, Figure 19. The features (in data science the inputs of a dataset are called features) as the first phase, the dot product of the features, the weights and the bias as the second phase and finally, the last phase is the unit step function.

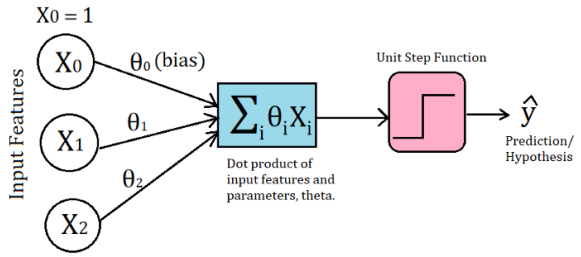


Figure 19: A visualization of a Perceptron with 2 features and 1 bias (X0) as the inputs, the dot product and the step function [9]

The limitations of the perceptron, is that it can only return one output and can solve only linear problems. In order to solve non-linear problems with a perceptron, we can build a multi-layered perceptron network. This way we will be able to solve non-linear problems such as object detection.

3.3 Training a Perceptron

The simplest Artificial Neural Network example is the Perceptron [9], which simulates one neuron of the human brain. We can train a Perceptron by constantly feeding into the network data from a dataset, that contain features and target class types. An implementation of a Perceptron is provided in this GitHub page [8]

Features are the inputs of the network, for example, a pony's height/weight vs horse's height/weight. The dimensionality of a perceptron will increase according to the number of features that are included in the dataset. Class types is the value the network will classify based on the features. The example with a horse's and pony's height/weight is a linear problem, non-linear problems are harder to solve and can regularly be unpredictable. Object Detection problems are non-linear.

The linear formula that is used to calculate the difference of each class type is the dot product of the weights and the features plus the bias. The bias is used to scale the result of the linear separability formula in order to fit the data.

$$LinearOut = Feature[i] * weights[i] + bias \tag{2}$$

2: i = 1, 2, ... n, where n is all the different type of features (Height, Width, Weight etc)

The output of the linear formula is then processed in the activation function or unit step function. The output of the activation function will be either 1 or 0 based on the Linear Output. The values 1 and 0 correspond to the class types, which can be a string or a number (e.g. cat as the value 1 and dog as the value 0).

$$ActivationFunction = \begin{cases} 1, & \text{if } LinearOut >= 0 \\ 0, & \text{Otherwise} \end{cases} \tag{3}$$

3: **Activation function.** Determines which class will be the output based on the Linear Output

The values that are adjusted when training a Perceptron are the weights. The weights can be trained using the following formula

$$Weights[i] = Weights[i] + LR * (Target - predicted) * Feature[i] \tag{4}$$

4: LR is the Learning Rate of the Neural Network, i = 1, 2, ... n, where n is all the different type of features

3.4 The Structure of a Convolutional Neural Network

The main difference between a Perceptron and a CNN is that the CNN contains various other layers that prepare the image for the classification layer. Another important difference between the two is that as mentioned in section 3.1 the Perceptron will only solve linear problems. In the contrary, a CNN's classification layer that is meant for object detection in images, will need to to solve non-linear problems in order to satisfy the problem. This indicates that the classification layer of the CNN will be a multi-layered [36] Perceptron.

The structure of a Convolutional Neural Network [36] generally consists of two major parts, the feature extraction layers and the classification layer, Figure 20. The feature extraction layer works by decomposing and compressing the image into a processable state for the classification layer. The classification layer's design is a more complex version of the perceptron, since we are attempting to solve a non-linear problem. In the classification layer, we can use a multi-layered perceptron to identify objects from the decomposed image. There are also several other architectures used to build an efficient classification layer, such as a fully-connected layer and support vector machines.

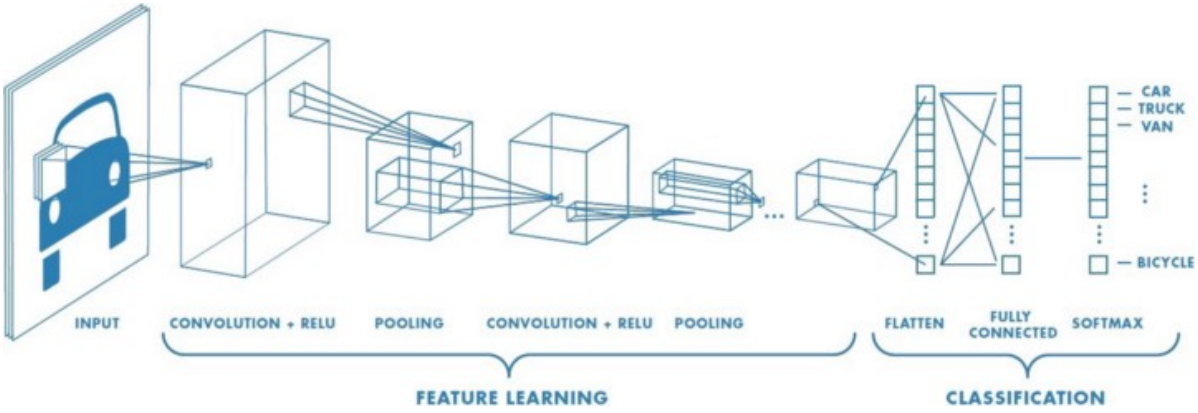


Figure 20: **Convolutional Neural Network Structure**

3.4.1 The Input Layer

The input layer of a CNN processes each color channel of the image (e.g. Images with the color format RGB, are split into their counterparts, Red, Green, Blue color channels). The network now has multiple perspectives of the image to process. After the image has been filtered into multiple different layers, it is now easier for the network to process the information the image contains.

3.4.2 The Convolutional Layer

After the image has been decomposed into its different color-channel variants, the image then proceeds into the Convolutional layer of the network. The network will then process a subset of pixels instead of every pixel at a time. That subset is called the Kernel, and it is used to apply operations and extract features.

Most Kernels use a 3 (Height) x 3 (Width) space, to process the subset of the image. The final result given by the convolutional layer will be the feature map, a compact map of important information extracted from the image, Figure 21.

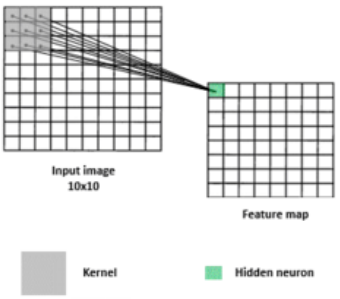


Figure 21: **Convolutional Layer**. Feature extraction layer.

3.4.3 The Pooling Layer

As a final phase for the feature extraction layers, the decomposed feature map proceeds into the pooling layer. The goal of the pooling layer is to reduce the feature map's size in order to increase computational performance, while at the same time, highlight the most important information in the feature map.

There are two pooling algorithms, max pooling and average pooling. The pooling layer uses the same feature extraction method, the Kernel, that was used in the convolutional layer, to process a subset of features and extract the most valuable information. In that subset, either the average or the max value is extracted into a new feature map, Figure 22. This helps reduce noise and lower the process time by using the most important values instead. The max pooling technique is usually more efficient for most CNN.

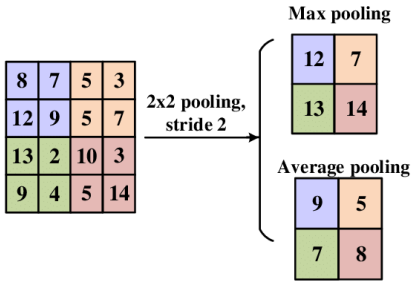


Figure 22: **Pooling Layer**. Feature map compression

To summarize, the pooling layer compresses the feature map and extracts the most important values, decreases noise for higher precision helps with the performance of the network and assists the classification layer to produce better results.

3.4.4 The Classification Layer

Finally, after the image has been decomposed and compressed by the feature extraction layers, the feature map proceeds into the classification layer. Convolutional Neural Networks usually use multi-layered classification perceptrons. By feeding multiple groundtruth images into the network in order to train the network on a certain class type, the weights are gradually adjusted to match the objects given by the groundtruth information.

After a certain amount of iterations and epochs, the network will be able to detect and classify objects with very high average precision. It is important to note that the classification layer will be classifying with non-linear functions in the case of object detection. Non-linear classification is when a class type cannot be classified by linear space.

3.5 Object Detection using YOLOv4

The YOLOv4 model [1] originates from the Darknet framework [23]. It's a deep learning based method that uses Convolutional Neural Networks. Like most models in the Darknet

framework, YOLOv4 was trained on the MS COCO dataset.

The MS COCO dataset [37, 38], is a large-scale object detection, segmentation, and captioning dataset. It was created by a team of contributors sponsored by Microsoft, Facebook, CVDF (Common Visual Data Foundation), Mighty Ai.

YOLOv4's structure is based on the CSPDarkNet53 [39] an advanced Convolutional Neural Network which uses DarkNet-53 [40] as the backbone aka YOLOv3. YOLOv4's performance outperforms most state-of-the-art object detectors nowadays, with high AP (Average Precision and Performance (FPS) Figure 23.

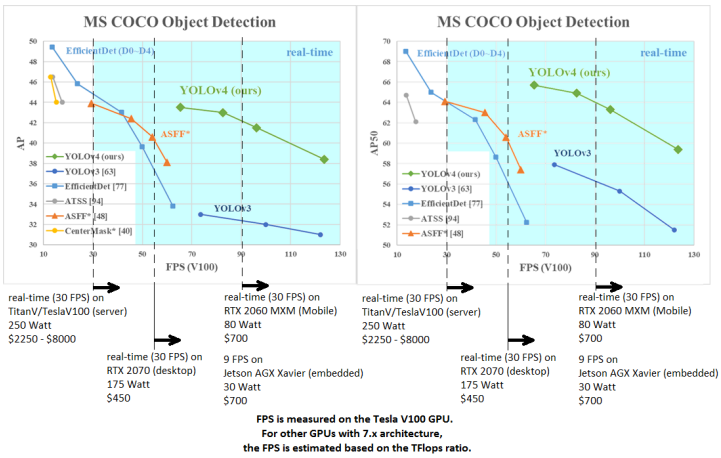


Figure 23: An old performance graph comparison of various YOLO object detector versions up to YOLOv4

For advanced users, the Darknet developers have also created a different library which includes different annotations and allows for more flexibility in managing Darknet's models.

Darkhelp is a C++ API wrapper which provides a lot of functionalities, it allows the user to annotate the model and include more results after the objects have been classified, such as compile time, the classification threshold and more.

4 Training Convolutional Neural Networks

In this section a variety of methods were researched and applied, in order to efficiently train a Convolutional Neural Network such as YOLOv4. The dataset that was created to generate the example for this thesis, involves various bird species, such as corvids, pelicans, flamingos, sparrows, gulls and various other common bird species in Greece. In terms of the topic of object detection in low light images, this section provides a better in-depth understanding on how various models like YOLOv4, detect objects and learn from images.

Training a CNN can be a very challenging task based on the problem required to solve. Solving a CNN problem, is a very dynamic process that takes time, planning and a lot of researching. Various factors that make training a CNN challenging, is the dataset that it will be trained on. Each dataset creates a problem meant to be solved by the Convolutional Neural

Network, several problems might be too similar with each other, so in order to identify each and every one of them precisely, a careful analysis needs to be done.

The training of a Convolutional Neural Network is similar to the training of a Perceptron, although the problems in question are non-linear. Similarly to the perceptron, a CNN's training process is by feeding groundtruth information into the network. This task is repeated many times until an acceptable state has been reached. The problem in question can be considered solved when certain information such as **mean Average Precision** & **Average Loss**, have reached a satisfying rate.

In order to train a Convolutional Neural Network, several steps need to be followed appropriately, such as creating a well structured dataset, using the right model like YOLOv4 [1] (or higher version), or building a convolutional neural network and finally evaluating the model.

During the training of a convolutional neural network, each iteration is a batch size of number of images, each epoch is a re-iteration of the same images again. The more times this process is repeated, the more accurate the model will be based on the given dataset. We can assume that a CNN is trained well, when the mAP (means Average Precision) is high enough for the model to accurately predict objects and when the Average Loss is low enough.

4.1 Groundtruth Information in a CNN

The term `Groundtruth Information` [41] is regularly referenced, to state that a certain pattern of features is an expected result, or `Class Type` and is usually applied to training Perceptrons or Neural Networks. Groundtruth information in Convolutional Neural Networks, are images with information that assist the model to understand, that in a specific location in an image, there is a `X` class type. In short Groundtruth Information is the dataset that the Neural Network is trained on.

There are various forms of groundtruth information and in simple data science, it is a pattern of inputs from a dataset that is available in the form of text. In the topic of object detection, the groundtruth information is provided with the images themselves, with each image containing a '.txt' file labeled as the image's filename. That text file contains various information based on the location of the `expected` class type and the label itself.

The CNN YOLOv4, by default, uses bounding boxes in order to classify objects within images. Although, the detector can be modified to use circles or ellipses instead. A presentation of this exact example is done using Darkhelp [6], an extension for Darknet [23].

When training a Neural Network in your own dataset, it is important that the dataset is well structured and various methods are applied onto the dataset in order to improve the model's precision.

4.2 Creating a Dataset to Train a CNN

In order to create a dataset to train a CNN, several steps need to be followed appropriately when building a well structured dataset. The first process is to research for a large image library or database. For example the class type, ”bird” or any bird species, has a large number of public libraries with images available all over the world, in the following examples, eBird [42] was used to train YOLOv4.

During the development of this thesis, various methods were researched on creating a well structured datasets, in order to train the YOLOv4 model and improve its precision on past training sessions.

4.2.1 Gathering & Annotating Images

A python tool was created to pull a number of images of certain bird species from the public image library eBird [42]. After the image gathering and storing, the images need to be labeled in order so that the model will be able to recognize the position of each object including the class type.

A tool that can be used to train YOLOv4, using the Darknet framework, is Label-studio [43]. This tool works by importing several images and drawing bounding boxes on top of class types we want the model to detect. During image labeling, each object, must not contain any space in between, Figure 24.

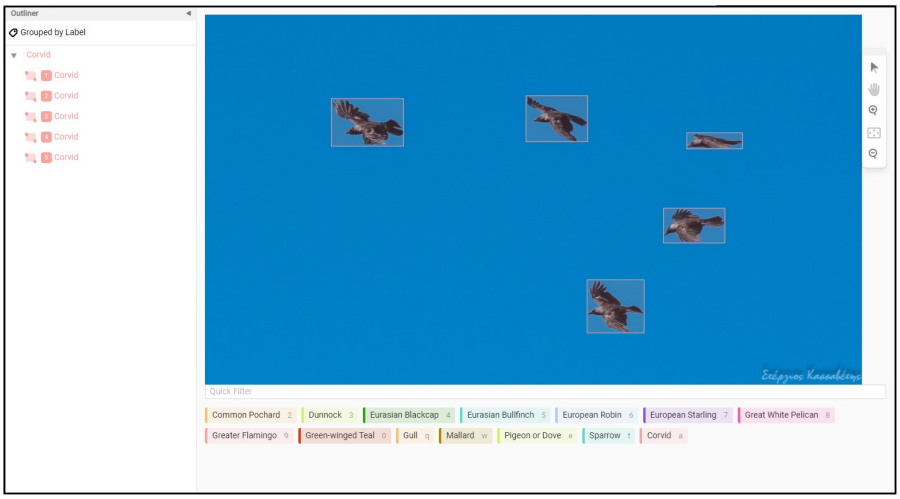


Figure 24: **Label-Studio Image:** Preview of an Image being Labeled using Label-studio

After every image has been labeled appropriately, the images can then be exported as the YOLO format, that evidently is supported by the YOLOv4 model.

4.2.2 Structuring & Improving the dataset

Before training, the dataset is split into two different subsets, the testing and training dataset. A typical training session action would be splitting the dataset at 90% training and 10% testing. This allows the model to determine the mean Average Precision appropriately and not based on the information it has already seen and is being trained on. Several methods such as data augmentation [27], image tiling [44], and negative sampling [21], can increase the model's precision during the training as well as better evaluation results.

Data Augmentation or Synthetic Data [45] in CNN training, is used to improve the precision of the model, in order to minimize the worst case scenarios during the evaluation of the network. Data Augmentation in practice, is when several transformations are applied to the already existing images, such as blur or median blur, random brightness, channel switching, image compression, image flip or rotation. A python library that is able to achieve something like that is Albumentation [46]. Data augmentation is the opposite of enhancing a low light image, degrading it to a state where objects are less visible. In this case, it is used to train the model to identify objects in various real world conditions such as foggy weather, during sub optimal lighting conditions and raining or snowfall.

Negative Samples [21] are images that contain no detections, based on the problem in question. In practice, negative samples can be the background or several details we want the model to not confuse with our training classes. For example, if the primary goal of the training session was to train a model to detect several bird species, in order to include negative samples, a large number of images in the dataset will contain no birds at all in them. This method helps reduce the False Negative errors during the evaluation of the network.

Image Tiling [44] is a technique that is regularly used by many convolutional neural networks. It works by tiling each image into multiple smaller pieces that match the network's size. This technique can dramatically increase the complexity of the problem, whilst the model instead of processing one image, processes multiple images that are subsets of one. The implication of getting this much accuracy using image tiling is a dramatic loss in the computational performance of the model during evaluation, Figure 25.

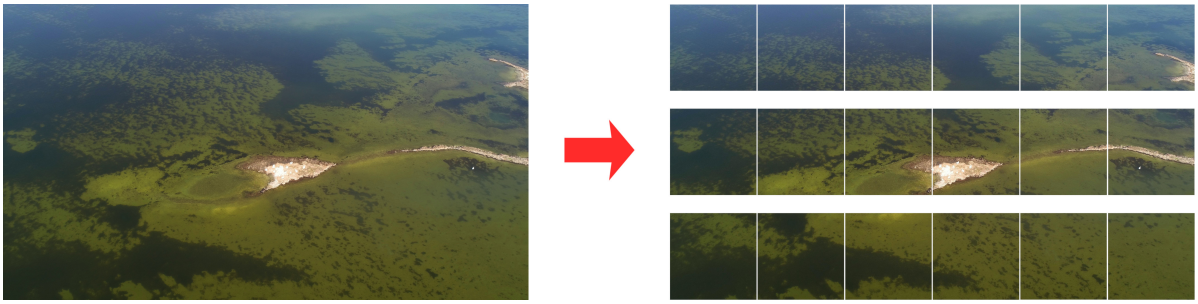


Figure 25: A comparison of an image tiled onto 17 different images to match the network size.
`Original Image size:  3840x2160`
`Image Tile Size:  640x640`

Each convolutional neural network will process an image based on a set network size. Any image that is parsed into the network that is above the network size will be scaled down to fit the network's size itself [47]. This results in a great loss of information resulting in much less accuracy and visibility.

Image tiling can become especially effective, based on the size of the objects we expect the model to detect. An example of this occasion can be displayed in the following Figure 26.

(a) YOLOv4 Prediction Results (512ms)

(b) YOLOv4 Prediction Results (1043ms)

Figure 26: **YOLOv4 prediction results**. This image contains 23 visible birds at an estimated altitude of 40 meters. The image on the left represents the prediction results without the use of `Image Tiling`, and the image on the right is tiled resulting in more accurate predictions.

The left image which does not use this technique has a total of 34.78% mean Average Precision (mAP), 8 True Positives (TP), and 15 False Positives (FP).
The image on the right which uses this technique has a total of 100% mean Average Precision (mAP), 23 True Positives (TP), and 0 False Positives (FP).

4.3 Training Performance Evaluation Methods

During the evaluation of a neural network, various statistical information are extracted, by evaluating the model on various data that it has not seen or been trained on. A model's performance is determined using the **mAP (mean Average Precision)** value. Data scientists use this value to compare the performance with other models, as well as various other training sessions that were deprecated due to poor results [48, 49], Figure 28.

mean Average Precision is the sum of all the average precisions for each class type, after the model has been evaluated. The precision of a class is calculated based on the **True Positives** and **False Positives**.

Recall is also another method to determine the performance of the model and is calculated using **True Positives** and **False Negatives**

True Positive is when the model returns a prediction and is correct, based on the groundtruth information.

True Negative is when the model returns no prediction (Negative sample) and is correct, based on the groundtruth information.

False Positives is when the model returns a prediction and is incorrect, based on the groundtruth information.

False Negative is when the model returns no prediction and is incorrect, based on the groundtruth information.

Average IoU (Intersection over Union) is a statistical comparison between the bounding boxes of the model's prediction and the groundtruth bounding box. The average IoU value represents the difference between the prediction and the groundtruth's bounding box of multiple objects. The higher the value, the better the prediction.

As outlined above, the average IoU is calculated using the groundtruth and the prediction bounding box coordinates. In the case of Darknet and the YOLO models, we used DarkHelp to extract the coordinates of each prediction of our model and compare those predictions to the groundtruth boxes that we labeled. Darknet's groundtruth labels consist of the following format and are stored in '.txt' or '.json' files.

```
Class_ID X_Center_Point Y_Center_Point Box_Width Box_Height
```

(a) IoU comparison results using the **YOLOv4-tiny** model. `Average IoU: 86.93%`

(b) IoU comparison results using the **YOLOv4** model. `Average IoU: 90.22%`

Figure 27: Intersection over Union examples using YOLOv4-tiny & YOLOv4. Comparison of a model's prediction bounding box with the groundtruth bounding box. The percentage represents the IoU calculated using DarkHelp, OpenCV.

In order to calculate the IoU probability, the intersection's box coordinates need to be defined. Using the Intersection coordinates we can calculate the Interception bounding box area. Using the Intersection area we can calculate the Union's bounding box area. The ratio

of Intersection divided by Union is equal to the IoU value.

$$IntersectionTopLeft = [max(grt_x, pred_x), max(grt_y, pred_y)]$$
$$IntersectionBotRight = [min(grt_x + grt_w, pred_x + pred_w), min(grt_y + grt_h, pred_y + pred_h)]$$

$$IntersectionWidth = IntersectionBotRight[0] - IntersectionTopLeft[0]$$
$$IntersectionHeight = IntersectionBotRight[1] - IntersectionTopLeft[1]$$

$$Intersection = IntersectionWidth \cdot IntersectionHeight$$
$$Union = grt_w \cdot grt_h + pred_w \cdot pred_h - Intersection$$
$$IoU = Intersection/Union$$

4.4 Training YOLOv4 using Darknet

YOLOv4 [1], can be trained using Darknet [23]. In order to train any YOLO model efficiently, Darknet will need to be built with Cuda support in order to use the GPU of the computer, thereby dramatically increasing the performance of the model.

The first step in training our own Neural network, assuming Darknet has been built, the user can create a configuration file similar to the YOLOv4.cfg file but with customized settings based on the problem in question. Within the configuration files, the user needs to change the number of classes the model will operate on. The number of iterations per batch based on how much VRAM the Graphics Card has and more. More information is given in the Darknet FAQ website [26]

The second step is preparing the groundtruth information. In order to train YOLOv4, Darknet requires the image files and '.txt' files, to be located in the same directory. Each image contains a '.txt' file labeled as the image's filename. The information that is inside the '.txt' file is the class label's ID, the x and y location and the width and height of the bounding box of the object. The YOLOv4 model has support for the YOLO format text file which is specific to the YOLO models. There are also other formats similar to YOLO but may not be supported by the model.

Image1.txt

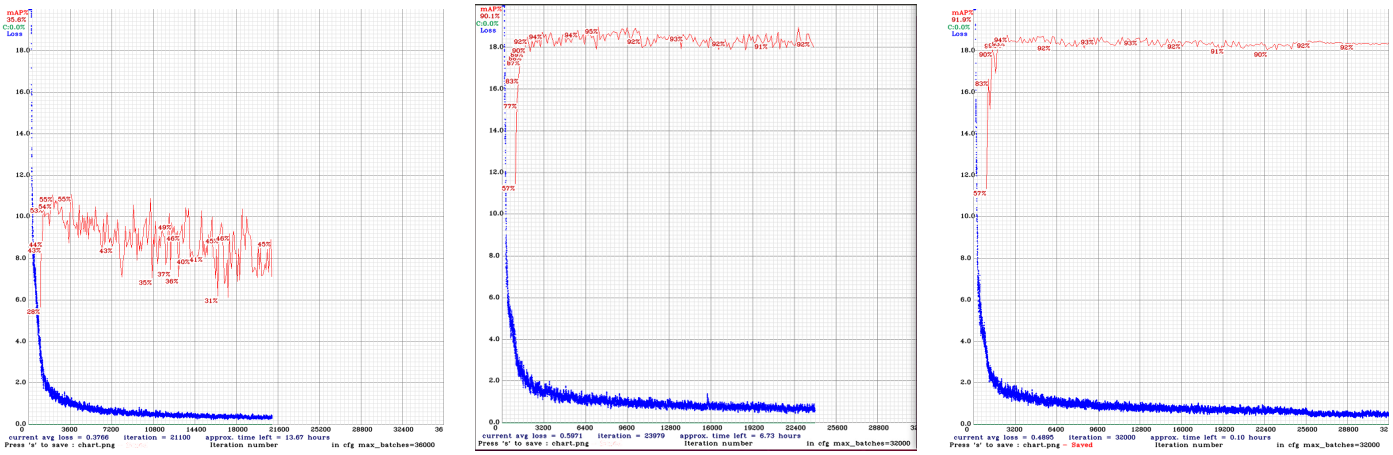
- Class_ID, x_Center, y_Center, Width, Height

During the training of YOLOv4 using Darknet, the user can observe a chart that represents the mAP (mean Average Precision) and Average Loss. The mAP is determined by the testing

dataset, while the average loss is the ratio for each miss-classifications per iteration for the entire batch of the training dataset. After the training is complete, the user can then evaluate the trained weights of the model and detect the objects it was trained on.

During the development of this thesis the YOLOv4 model was trained on 3 different datasets with various bird species. The first dataset was a small scale dataset, with a total of 250 images and 18 class types. The second dataset is a large scale image dataset with random images and a total of 16 class types and 2300 images. The last dataset was a well structured dataset with a total of 2586 images and 16 class types, various methods were applied, such as data augmentation, negative sampling, including various bounding box optimizations during the labeling of the second dataset.

The resulting training chart for each training session was different from each other. The first training session, was the unstructured dataset with a number of class types, but a very small amount of images per class type, the chart of that training session is displayed in Figure 28a. The second training session since it was a larger scale dataset with random images, the resulting training chart showed better results compared to the first training session, Figure 28b. The third training session was a large scale and well structured dataset with data augmentation and negative sampling applied to the dataset. The resulting training chart seemed to maintain more balance for the total mAP, as displayed in Figure 28c.



(a) Training Chart 1 (b) Training Chart 2 (c) Training Chart 3

Figure 28: YOLOv4 Training charts of three different training sessions from worse to better. Higher red line values is better. Lower Blue line values is better

4.5 Evaluating YOLOv4

The evaluation of YOLOv4 model, involves testing the trained weights, in either videos or images. Within the first training session of the model, it was able to accurately detect in the training dataset images, but it under-performed in images it has not seen before, this problem is called overfitting [50] and is regularly observed during the training of a neural network. The second training session, since the model was trained on a large scale image dataset, the

results were much better overall and overfitting was not as evident. The mAP was a lot better than the first session and overall produced a lot better results. During the evaluation of the second training session 28b, there were no negative samples within the dataset, False Negative predictions were regularly observed such as the one in displayed in Figure 29a.

In the other hand, on the 3rd training session 28c, the model was trained again with a certain amount of negative samples contained inside the dataset. The result given by the model is displayed in Figure 29b

(a) False Negative

(b) True Positive

Figure 29: Evaluated image result with a False Negative and a True Negative result, where the negative class ”person” was identified and that is incorrect.

YOLOv4 contains a flag that allows the user to evaluate and extract the mAP and average IoU values to quickly compare the performance of each model. One of the training sessions that achieved the best results is displayed in the bellow table 1

	Test Dataset Best Weights			
Class Type	**True Positive**	**False Positive**	**Groundtruth Prediction**	**Average Precision**
Common Pochard	30	4	33	94,28%
Corvid	56	9	61	96,75%
Dunnock	42	2	42	99,94%
Eurasian Blackcap	28	3	30	92,07%
European Robin	25	0	25	100%
European Starling	18	0	19	94,74%
Great White Pelican	35	7	59	60,16%
Greater Flamingo	36	3	38	97,16%
Green-Winged Teal	38	2	40	99,46%
Gull	47	5	54	92,29%
Mallard	61	17	66	95,30%
Pigeon or Dove	42	13	49	86,25%
Sparrow	35	2	38	94,33%
	Total: 493	Total: 67	Total: 554	mAP: 93,03%
				Average IoU: 71,64%

Table 1: Training Evaluation Results Using The **`map`** Flag

It can be concluded that YOLOv4 and even most Convolutional Neural Networks in real world problems need a large number of negative sample images and a decent amount of data augmented images, to increase the mean average precision. The final result will be the reduction of the False Positives/Negatives results of the model.

5 Object Detection in Low Light Conditions

After the brief introduction on Convolutional Neural Networks and how they detect objects in an image, we begin to realize that they heavily rely on images that have enough luminance. When the shape of an object is not visible enough, the detector fails to classify that same object. If the image does not have enough luminance, we begin to face a problem where the image will not make it possible for the network to detect certain objects within it, Figure 30. In most cases, images with low light conditions will cause this problem.

Figure 30: **ExDark Low Light Image.** An extremely low light conditioned image

Human vision heavily relies on bright environments, we are able to see color when light falls into our eyes. The light we perceive are electromagnetic wavelengths that our brain

processes into colors. Since we are a biological species, we are limited to a certain amount of wavelengths that we are able to perceive. The human eyesight can perceive a wavelength of 380 to 750 nanometers (400 - 790 Terahertz) of the color spectrum, Figure 31. In short, the estimated human visual system can detect up to 7 million color variations[30] in the other hand, the average image has a depth color of 24 bits which translates to 16,777,216 color variations. This means that computer vision has an advantage in terms of the detail it can perceive.

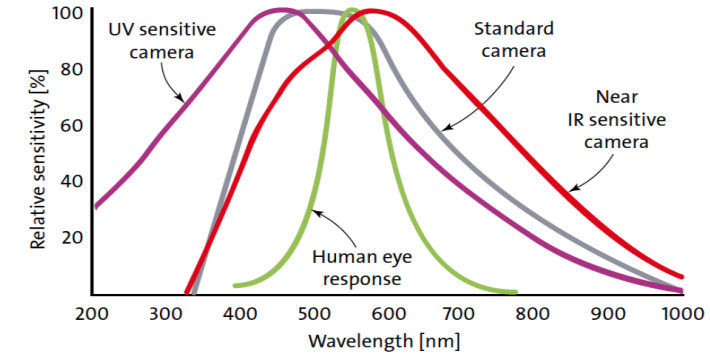


Figure 31: **Visual System graph.** A graph comparison of human eyesight with various camera sensitivities

Most Object detectors struggle to classify objects accurately within low light conditioned images, this is due to the lack of brightness and color within the image.

We now begin to realize that most object detectors fail to detect objects in low light conditions, due to the lack of color and brightness within the image. In the next section, this thesis will include various methods that will be useful for computer vision models to detect objects more effectively and accurately in low light conditions.

5.1 Enhancing Degraded & Low Light Images

Images with low light conditions often face problems with degradation. Degradation in an image is when noise is evident, which can negatively affect any object detector.

By enhancing images in real life problems such as self-driving cars with object detection systems, we can assist the model to consistently detect objects even on a bad weather or during the night. This is to minimize any possible errors that may occur during those conditions and reduce the risk of an accident that could be caused by those systems. Systems like these are required to be fast, precise and efficient to minimize the margin of error.

In this thesis, I will use and reference multiple techniques to enhance images with low light conditions, to achieve higher detection rates and average precision. Enhancing a low light image can be a double-edged sword. After enhancing an image, noise gets up-lifted and can negatively affect the model to be less accurate.

5.2 Enhancement methods

Image enhancement works by applying filters to an image, each filter that is applied to the image is done with the purpose of making the objects in that image more visible than before. There are several methods to enhance an image, as well as removing noise.

Saturation & Contrast

Two very important components about enhancing an image's quality is by adjusting the saturation and contrast. By changing these two values, the image will be a lot brighter and colorful.

Intensity Value Inverting

Inverting an image is regularly done to highlight the darker areas of the image. By inverting an image, the light and dark areas are reversed, this method is effective for increasing the luminance of an image if combined with other filters, making the shapes and details much more visible. This method is used in advanced low light enhancing algorithms.

Intensity value scaling

All algorithms use different methods to scale the intensity values of the image. Intensity value scaling works by using a multiplier like the gamma correction algorithm to increase it. The intensity value is the number of a color channel of an image. This value usually ranges from 0 to 255.

Convolutional Neural Networks

Convolutional Neural Networks (CNN) are used for object detection as well as low light image enhancement. As of today, these networks are one of the most powerful methods developers use to enhance low light images and are able to compete with state-of-the-art algorithms.

Degradation Removal or Denoising

Denoising or degradation removal methods are regularly used in state-of-the-art enhancement algorithms in order to increase the level of detail in an image. Degradation removal is an essential tool that removes noise that is evident in the image. Noise is up-lifted when adjusting the luminosity of the image, decreasing the level of detail.

5.3 Low Light Image Enhancement Algorithms

The goal of enhancing a low light image is to increase the luminance and visibility of objects that originally were not visible.

The tools used for this thesis to implement most algorithms are with Python, OpenCV and NumPy.

There are many algorithms specifically for low light enhancement, the most widely used algorithms are Histogram Equalization, gamma correction and various Convolutional Neural Networks methods.

5.4 Histogram Equalization

Histogram Equalization [4, 51] is a commonly used algorithm that uses histograms in order to increase the luminosity of an image. The histogram that is created from the image represents the count of intensity values of the equalized color space. The algorithm stretches out the most frequent intensity values and creates a newer distribution that is wider and more regulated. By doing that, we change the result will be brighter and contain more luminance.

Histogram Equalization works by equalizing the intensity values of the image, in other words the most frequent intensity values are distributed equally to make a brighter image. Histogram Equalization suffers in very low to extremely low light images due to the concentration of intensity values, the result from those images will be an extremely degraded image, Figure 32. An important note about Histogram Equalization is that there are several methods to equalize an image. By equalizing either all the RGB color channels of the image or by converting the color format of the image into HSV (Hue, Saturation, Value) and equalizing the value color channel.

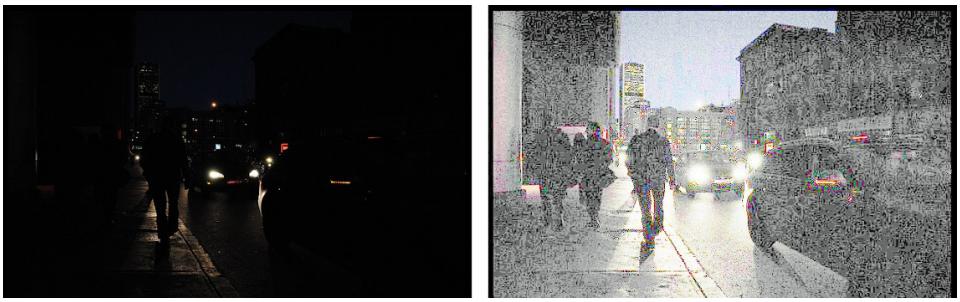


Figure 32: **HE Algorithm (ExDark Image 06542).** Preview of an extremely degraded image

An acceptable result from Histogram Equalization would be Figure 33.

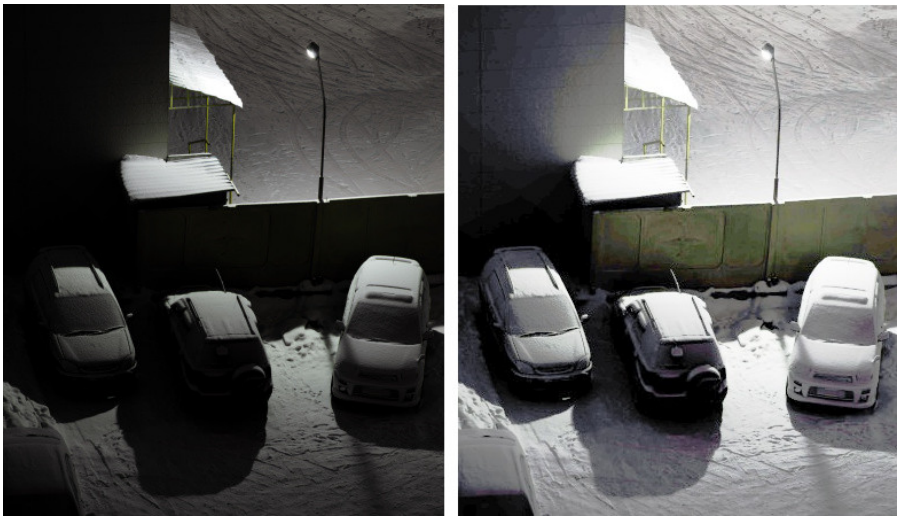


Figure 33: **HE Algorithm (ExDark Image 02453).** Preview of an original and equalized image result.

The Histogram distributions of the images in Figure 33 are previewed in Figure 34.

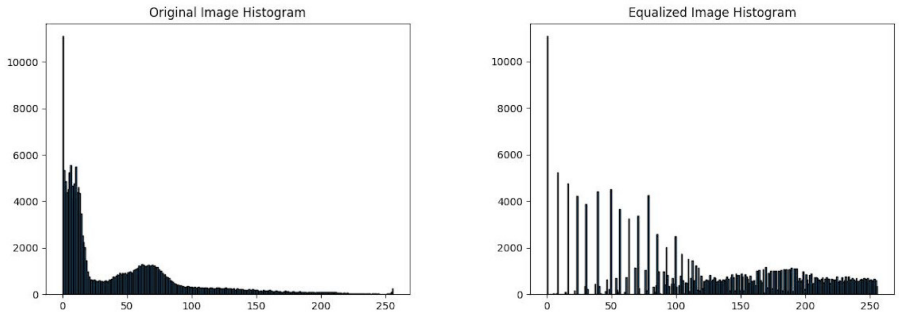


Figure 34: **Histograms from Figure 33.** Histograms of an Original and Equalized image.

Notice the newer distribution, that we acquired from the algorithm Figure 34, the intensity values are further spread apart from each other and equalized in the entire intensity value range. While the histogram of the original image is more compact and dense. Overall, the algorithm can be effective in certain scenarios, but it will begin to fail in very low to extremely low light conditions.

5.5 Gamma Correction

Gamma Correction[3] is one of the most famous low light enhancement algorithms regularly used by many companies (E.g. Nvidia, game development, graphic driver companies, etc.). It works by scaling each pixel in the image.

Gamma Correction uses the γ value that can be adjusted by the user. If the γ value is greater than 1, the brightness of the image will increase. Otherwise, the brightness of the image will decrease. The algorithm processes the intensity value by converting it from 0 - 255

to 0 - 1. The Gamma Value is inverted, as seen in the Formula 5, in order to scale the intensity values up.

$$ScaledIntensityValue = i^{\frac{1}{\gamma}} \tag{5}$$

5: **GC Formula.** Intensity Value Scaling Formula

This algorithm was implemented using OpenCV and NumPy in python. The result from this algorithm can be previewed in Figure 35

Figure 35: **GC Algorithm (ExDark Image 03152).**
Preview of the original image, compared to two different gamma value results.
Left Image: Original (Or even $\gamma = 1$)
Middle Image: Increased Brightness $\gamma = 2$
Right Image: Decreased Brightness $\gamma = 0.5$

5.6 Kindling the Darkness (KinD)

Kindling the Darkness (KinD)[5] is a state-of-the-art deep learning based method that uses convolutional neural networks to enhance low light images. KinD was inspired by the Retinex theory. Its structure is divided into three layers, image decomposition, reflectance restoration and illumination adjustment layer. These layers play an important role in enhancing the image to boost the accuracy rate and detection rate of an object detector.

The decomposition layer works by decomposing the image into two different components, the reflectance and illuminance maps. The illumination adjustment layer works by adjusting the light levels of the map. The strength of the light adjustments depends on the ratio value α, similarly to Gamma Correction's γ value. Users can adjust the α value willingly from 0 to 5.0 during the testing phase of the network.

Finally, the reflectance restoration layer works by removing any degradation on the map. Degradation removal (Noise Removal) is the act of removing any haze/noise that exists in an

image and restore it into a clearer image. The act of removing degradation in an image can assist the classifier to detect objects. Degradation removal and image Degradation has been widely researched for the topic of computer vision. The reflectance restoration layer uses the Block Matching (3D BM3D)[52] algorithm in order to restore the image. The result from KinD will be the following Figure 36

Figure 36: **KinD Algorithm (ExDark Image 02446).** Preview of an original and KinD image result.

6 The ExDark Dataset

The ExDark Dataset is a collection of low light images of objects, all listed in certain categories. [2] ExDark contains a total of 7363 images (Total Size 1.40 GB uncompressed). This research begun on May 29, 2018, and as of time of writing still being expanded. The categories that are included in this dataset are displayed in Figure 37 and Table 2

Class	Bicycle	Boat	Bottle	Bus	Car	Cat	Chair	Cup	Dog	Motorbike	People	Table
N. Images	652	679	547	527	638	735	648	519	801	503	609	505

Table 2: **ExDark dataset**. The class labels and Number of images contained in each class.

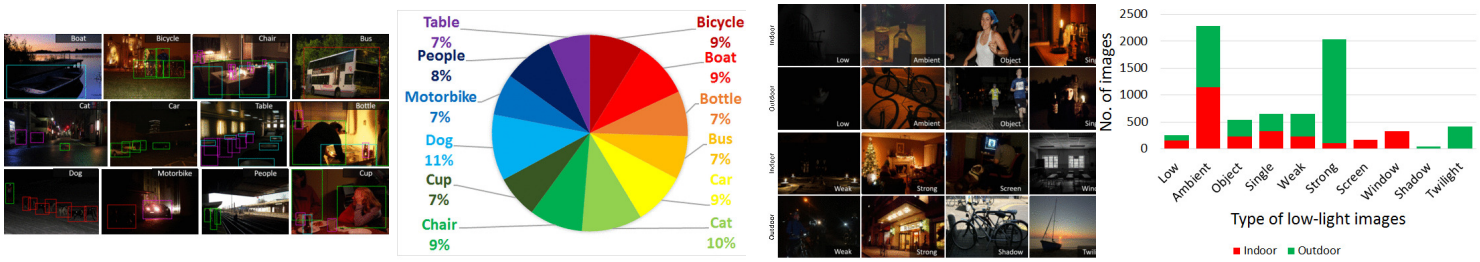


Figure 37: Information given by the7 ExDark dataset

The ExDark dataset was created to provide in research for low light enhancement algorithms for object detectors and image classification 37.

ExDark will be a valuable asset in this thesis, in order to feed ExDark's images into the YOLOv4 model and extract information from various low light conditions.

6.1 ExDark's Groundtruth Information

The ExDark dataset contains groundtruth information [53]. Within ExDark, there are 12 different folders labeled as the object classes that contain '.txt' files labeled as the corresponding images. The information given, is the coordinates of the bounding boxes that represent the location of a detected object, as well as a string name, that represents the class type of the detected object.

6.2 Annotation Embedding Algorithm

The annotation embedding algorithm is a support algorithm that generates the groundtruth information given by ExDark as '.txt' files into the images. The algorithm works by reading the information within each '.txt' file labeled as the corresponding image. The information given in the text files are multiple bounding box locations and class type information. The algorithm reads this information and re-generates the images, but this time with the bounding boxes and class type information visualized in the image.
The generated images represent a visualization, not necessarily to train an object detector, but in order to understand and visualize the groundtruth information.

ExDark has also referenced a Matlab toolbox that generates the bounding boxes from the text files to the images, similarly to the algorithm mentioned before. Due to legal reasons, the algorithm was re-implemented using Python and OpenCV.

7 Implemented Algorithms & Frameworks

This section includes a description of various algorithms implemented in this thesis. This section's importance is to maintain a clear and informative picture of the experiments and research that was compiled in this thesis.

The algorithms that were implemented and mentioned before, are Gamma Correction, Histogram Equalization. All the enhancement algorithms were implemented in such a way to be able to read, filter and write in multiple images at the same time. A person with minimal experience with pip and python can run these programs.

Several other algorithms were also implemented to assist in the experiments done in this research. Including various other data visualization algorithms.

Those tools are, the annotation embedding algorithm for the ExDark dataset, an algorithm that generates the Histograms of images, a simple progress bar to simulate the completed processes of each algorithm. The algorithms mentioned above can be found in the following GitHub repository [54].

External tools and Versions used in the experiments:

1. OpenCV Ver.: `4.5.5.62`
2. Python Ver.: `3.10 64x bit`
3. NumPy Ver.: `(Any version will do)`

4. TensorFlow Ver.: `2.10.0`
5. PIL Ver.: `9.0.0`
6. Matplotlib Ver.: `3.5.1`

7.1 Implementing & Running the GC Algorithm

Gamma Correction is a simple algorithm implemented using OpenCV, NumPy and several other third party libraries in Python[3]. This algorithm was built in order to process and return multiple images at once. It contains an input folder and an output. The algorithm transforms the pixels using a lookup table and an inverse gamma value as explained in 5.5. OpenCV contains a function, called LUT, that uses the lookup table to transform the pixels of the image.

The following code can be cloned using git in this GitHub repository [54] Running the Code in a Command Line

```
C:\Users\UserName\Desktop\Project> python GammaCorrection.py
```

```
def GammaCorrectionMulti():
    Gamma = 2
    inverseGamma = 1 / Gamma
    table = [((i / 255) ** inverseGamma) * 255 for i in range(256)]

    table = np.array(table, np.uint8)

    gammaCorrected = cv.LUT(image, table)
    backend.writeImages(index, gammaCorrected, imageFilename)
    print("\n\nImage Filtering Complete. View outputGC folder...")
```

The above code uses the gamma value and inverses it, in order to create a lookup table. The OpenCV LUT function (Lookup Table) is used to as a colormap reference for the image, to increase the intensity values. The result will be a gamma corrected image.

7.2 Implementing & Running the HE Algorithm

The histogram equalization algorithm is fairly easily implemented using the equalizeHist function from OpenCV. Similarly to the Gamma Correction, it accepts and outputs multiple images to increase the ease of use of the algorithm. The algorithm first extracts the color channels of the image into blue, green, red and equalizes them, afterwards the equalized color channels are merged back together into a new image which results into the equalized image.

The following code can be cloned using git in this GitHub repository [54] Running the Code in a Command Line

```
C:\Users\UserName\Desktop\Project> python HistogramEqualization.py
```

```
import cv2 as cv
from src.Backend import backend

def histogramEqualizationMulti():
    b, g, r = cv.split(img)

    equ_b = cv.equalizeHist(b)
    equ_g = cv.equalizeHist(g)
    equ_r = cv.equalizeHist(r)
    equ = cv.merge((equ_b, equ_g, equ_r))

    print("\nImage Filtering Complete.")
    cv.imshow("Equalized Image", equ)
```

The above code splits the color channels of the image using the OpenCV function split() and applies histogram equalization for each color channel separately. The color channels are then merged back together into the image. The result will be an equalized image.

7.3 Building & Running Kindling The Darkness (KinD)

Building KinD is an easy process using git to clone the GitHub page [5]. Due to backwards-compatibility issues with the current available TensorFlow versions, the user will need to manually change the model.py import files due to versions 1.x TensorFlow no longer available in the pip repositories. The changes will be provided with the thesis for ease of use.
Similarly to the algorithms GC and HE, the algorithm was modified to allow multiple inputs and returns multiple outputs.

The algorithm will effectively run with the following requirements
1. Python version: `3.7 (64x bit) >=`
2. TensorFlow Version: `>= 1.10.0 (Will not work for any version 2.x and above)`
3. Other Libraries: `NumPy, PIL`
Running the Code in a Command Line

```
C:\Users\UserName\Desktop\Project> python evaluate.py
```

7.4 Building & Running The Darknet Framework

Darknet as explained before is a framework originally created by Joseph Redmon, as of time of writing Joseph Redmon's GitHub repository of Darknet, has been abandoned and is currently maintained in another repository by Alexey Bochkovskiy and Stephane Charette. Stephane Charette provides a variety of information in a FAQ website, including a well described guide on building Darknet. [26]

There are three methods to build Darknet. A version where it supports CPU computing, another version where it supports GPU computing and is primarily for training or high performance object detection. Finally, there is a version that is more advanced in order to be

built, that requires OpenCV's DNN module and CUDA. This version allows OpenCV to be built with CUDA support and increases the performance of the model. Building Darknet using OpenCV with CUDA support, will require a lot of time [55].

In order to get started with Darknet, this part of the thesis will assume that Darknet has already been built and installed into the computer [23]. We can use the pre-trained weights given by Alexey Bochkovskiy in the GitHub page [23]. We can detect objects in an image using Darknet with the following commands.

```
> darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights MyImages/myImage.jpg
```

When running the above command, a window will open containing the image with various detected objects in it, Figure 38.

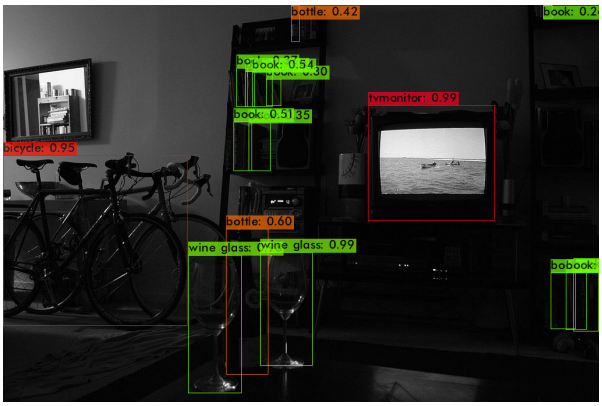


Figure 38: YOLOv4 Result by using the Darknet framework **(ExDark Image 01611)**

In the command line, various information will be printed depicting information from YOLOv4's CNN. The information that will be of use in this analysis is located at the end. This information as explained in section 3 is the confidence of the model as well as the class type of the detected object.

The argument `cfg/coco.data`, tells the model which classes to detect on and specifically on the classes trained using the COCO dataset.
The argument `cfg/yolov4.cfg`, tells the framework to use the YOLOv4 model.
The argument `yolov4.weights`, is the path to the weights file of the model. Alexey Bochkovskiy provides pre-trained weights of the YOLOv4 model.

There are also various flags depending on what the user wants the output to be.
The flag `-dont_show`, prevents the pop-up window containing the image with the detected objects to appear.
The flag `-thresh 0.2`, changes the minimum confidence threshold to 20%, for any object to be treated as detected for the final output.

7.5 Advanced IoU & Average Precision calculator on the groundtruth information for the YOLOv4 model using DarkHelp [6] & OpenCV [7]

An application that was developed using OpenCV and DarkHelp in the C++ programming language. It compares the resulting predictions of the model and the groundtruth information from the '.txt' files and finally calculates the IoU and average precision of each prediction. If the IoU value exceeds 50% then the groundtruth bounding box will be drawn and the IoU value will be displayed. An example of this algorithm's result was displayed in Figure 27

8 The Analysis

The Analysis of this research includes testing the performance of YOLOv4 using a variety of low light condition images from the ExDark dataset [2], as well as testing the same images after applying several enhancement algorithms on them. The conclusion of this thesis will be the determining factor of which enhancement algorithm performs best. The information that will be extracted from YOLOv4 is the class type of the detected object, the confidence of the model for each detection and the total average confidence.

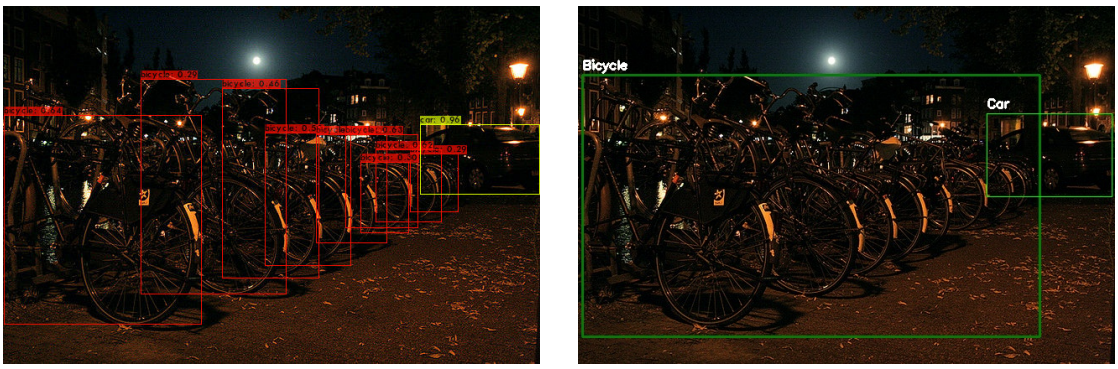
The extracted information will be organized into tables and used to calculate statistical values. The statistical information that will be extracted is the Recall and Precision of the model for each image, including the total average recall and precision.

8.1 Analyzing the Results with the Groundtruth Information

Groundtruth information is a valuable asset in determining the performance of each enhancement algorithm using real data and without any assumptions. Each result from YOLOv4 will carefully be analyzed and compared to the groundtruth information. In short, the analysis is limited to what the groundtruth information offers us.

An object will be evaluated if the location of the bounding box from YOLO and the groundtruth information represent the same object. In other words, if two objects with the same class are detected from YOLOv4 and one of the two is contained within the groundtruth image, only the one that the groundtruth information contains will be evaluated.

Within the groundtruth images, multiple clusters of detections in a concentrated area are compacted into one detection, YOLOv4 does not apply this. This could hinder the results, so hand-selecting the images will be important. If that occurs and all the objects within that cluster are correctly detected, it will be evaluated as one correct detection and not many. An example of this is shown in Figure 39



YOLOv4 Groundtruth Image

Figure 39: A preview comparison of detected clustered objects of YOLOv4 and the Groundtruth image (**ExDark Image 00200**)

Lastly, some of the results from YOLOv4 will be counted as correct detections even if they were not given by the groundtruth information as a class type, e.g. if the detection is a wine glass and groundtruth states that, that exact object is a cup, it will be assumed as a correct detection.

8.2 The Interpretation

In order to understand the process of the analysis, we need to understand some of the important terms used in this thesis. An important side-note is to understand the difference between a detection, prediction and classification.

Detection: Detection is either the object detector or the ground truth information states that an object at a certain location has been detected to be there.

Prediction: Prediction is when an object detector has created a probability of a detected object within an image to be a certain class type. (E.g. The model has predicted that this object is a Cat with 95% confidence and 5% confidence that it is a Dog)

Classification: A general term used in machine learning. Classification is When a model determines an output or class type from a pattern of features.

Detection Rate: The total number of detections given by the model, in comparison to the number of groundtruth detections.

Average Confidence: The result of YOLOv4's total accumulated means in confidence in the same image.

Missing Detections: The number of detections that were not registered by YOLOv4 based on the groundtruth information for each image.

Correct Prediction: Predictions given by the model which are correct based on the given groundtruth information.

Recall: The ratio between the correct predictions of the model and the missing detections based solely on the groundtruth information. [49, 56]

$$\textbf{Recall} = \frac{\textbf{Correct Predictions}}{\textbf{Correct Predictions + Missing Detections}} \tag{6}$$

6: Recall's output ranges from 0 to 1, it can be scaled up, into a percentage.

Precision: The ratio between the Correct Predictions of the model and the Incorrect Classifications of the model, based solely on the groundtruth information. [49, 56]

$$\textbf{Precision} = \frac{\textbf{Correct Predictions}}{\textbf{Correct Predictions + Incorrect Prediction}} \tag{7}$$

8: Precision's output ranges from 0 to 1, it can be scaled up, into a percentage.

Performance: The overall performance of the model. The ratio between the correct predictions of the model and the incorrect classifications and missing detections of the model, based solely on the groundtruth information.

$$\textbf{Performance} = \frac{\textbf{TP}}{\textbf{TP + FP + FN}} \tag{8}$$

9 Applied Analytics

In this section, a number of images are evaluated based on the information interpreted in Section 8.2 in order to extract valuable information based on YOLOv4's performance. Three algorithms are analyzed, including the original images. Every image used in this analysis was provided by the ExDark Dataset [2] as well as the groundtruth information that is included. The algorithms, that were explained in prior sections, are Gamma Correction, Histogram Equalization and Kindling the Darkness aka KinD.

This thesis, uses the pre-trained weights from Alexey Bochkovskiy, to determine the performance of each enhancement algorithm using YOLOv4. Each section that involves a certain test will include a large table with various information, in order to conclude from the results.

The images used for the analysis, in the latter sections, are randomly or deliberately selected from the ExDark dataset.

9.1 Inspection of the Enhanced Images

This section provides a visual understanding of the difference between each enhancement algorithm.

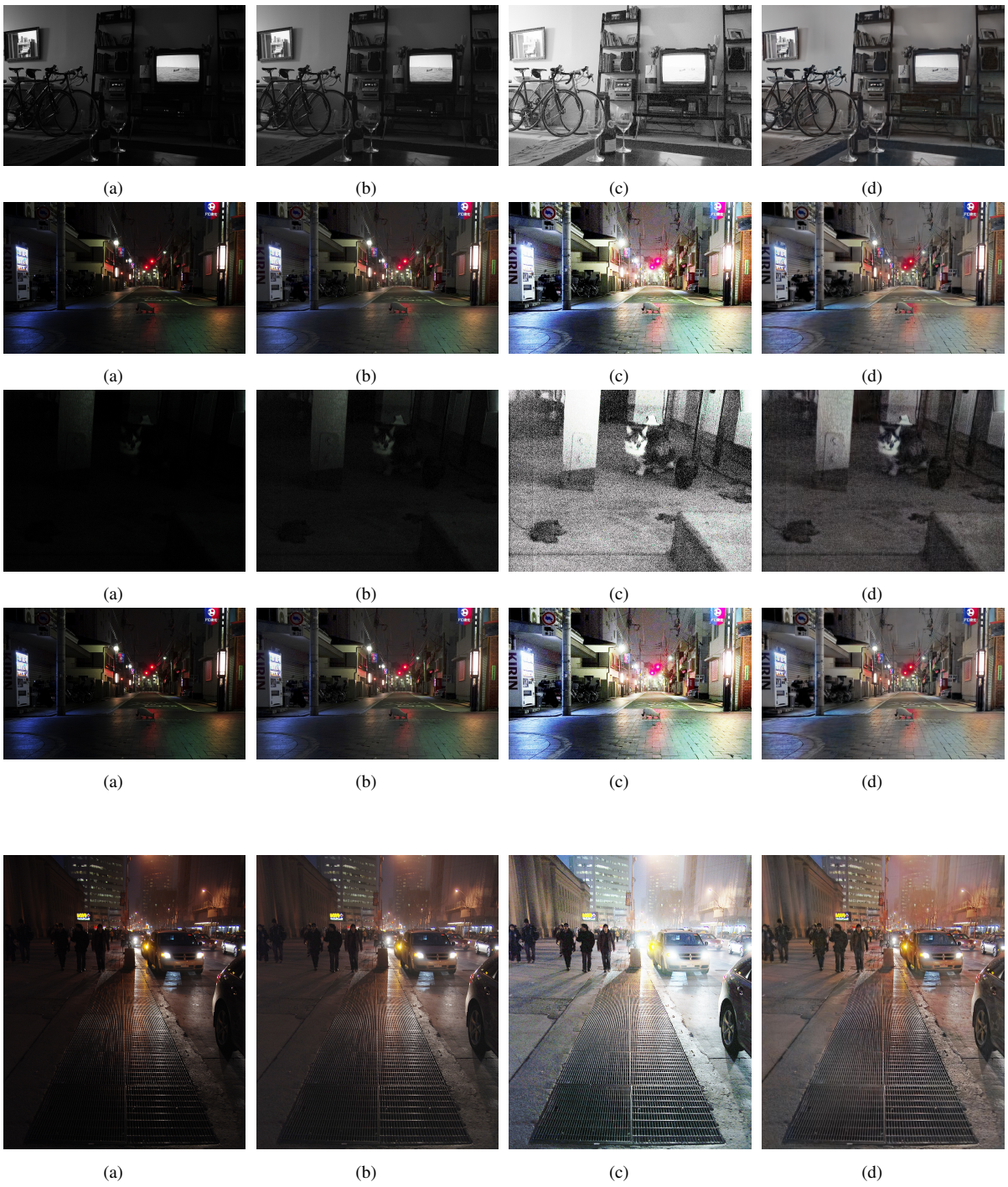


Figure 41: **(a):** Original Image, **(b):** Gamma Correction, **(c):** Histogram Equalization, **(d):** Kindling the Darkness

9.2 Data Analysis & Results: Original Images

YOLOv4 alone is a very powerful model and will hardly struggle to accurately predict objects within the images as long as the weights of the model were trained appropriately. In

this section and the next ones, several tables are displayed with various information that was interpreted in Section 8.2.

The results, extracted from YOLOv4 which are also based on the groundtruth information on the original images, are listed in the Table 3. Each image was referenced in order to depict the exact image the analysis was applied on.

	Original Images[2]					
Image	**N. Groundtruth Detections**	**Correct Predictions**	**Incorrect Predictions**	**Missing Detections**	**Recall**	**Precision**
00253	6	6	0	0	100%	100%
00301	2	1	0	1	50%	100%
00405	9	8	0	1	88.89%	100%
01367	10	10	0	0	100%	100%
01437	10	5	0	5	50%	100%
01468	15	13	0	2	86.67%	100%
01489	13	11	0	2	84.61%	100%
01611	5	5	0	0	100%	100%
01764	15	13	2	0	100%	86.67%
02049	5	4	0	1	80%	100%
02446	9	5	0	4	55.55%	100%
02453	3	3	0	0	100%	100%
02594	8	8	0	0	100%	100%
02756	6	4	0	2	66.67%	100%
03052	3	1	1	1	50%	50%
03215	2	1	0	1	50%	100%
03278	1	1	0	0	100%	100%
05380	2	1	1	0	100%	50%
05951	2	2	0	0	100%	100%
06077	3	3	0	0	100%	100%
06248	3	3	0	0	100%	100%
06255	5	5	0	0	100%	100%
06298	5	5	0	0	100%	100%
06365	9	9	0	0	100%	100%
06542	5	4	0	1	80%	100%
25 imgs	Total: 156	Total: 130	Total: 4	Total: 21	Avg: 85.695%	mAP: 95.466%

Table 3: Original image information based on the groundtruth images.

The results confirm YOLOv4's performance by accurately predicting with very few detections without the use of an enhancement algorithm. **The performance of the model for the original images is 83.87%**

9.3 Data Analysis & Results: Gamma Correction

Gamma Correction provides good results and will hardly hinder the performance of the object detector in certain situations. Many times the algorithm was performing well with YOLOv4 and other times the given results were degrading the image, reducing recall and precision.

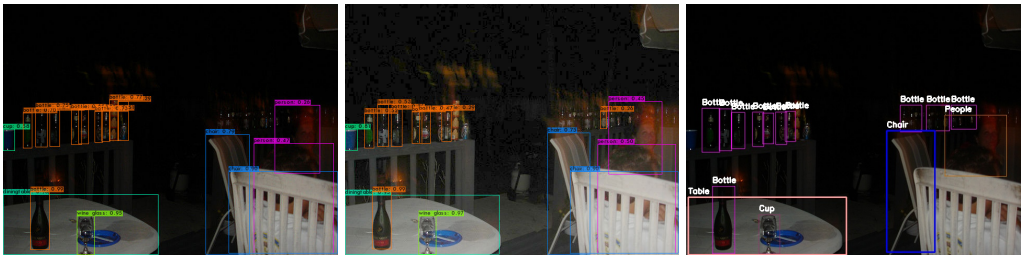
The information extracted from YOLOv4 and the groundtruth images by using the Gamma Correction on the Original Images 3 are listed in the Table 4

	Gamma Correction γ = 2[3]					
Image	**N. Groundtruth Details**	**Correct Predictions**	**Incorrect Predictions**	**Missing Detections**	**Recall**	**Precision**
00253	6	6	0	0	100%	100%
00301	2	2	0	0	100%	100%
00405	9	9	0	0	100%	100%
01367	10	10	0	0	100%	100%
01437	10	8	0	2	80%	100%
01468	15	12	0	3	80%	100%
01489	13	11	0	2	84.61%	100%
01611	5	5	0	0	100%	100%
01764	15	12	0	3	80%	100%
02049	5	4	0	1	80%	100%
02446	9	5	0	3	66.67%	100%
02453	3	1	2	0	100%	33.33%
02594	8	8	0	0	100%	100%
02756	6	2	0	4	33.33%	100%
03052	3	1	1	1	50%	50%
03215	2	2	0	0	100%	100%
03278	1	1	0	0	100%	100%
05380	2	0	2	0	0%	0%
05951	2	2	0	0	100%	100%
06077	3	3	0	0	100%	100%
06248	3	3	0	0	100%	100%
06255	5	5	0	0	100%	100%
06298	5	5	0	0	100%	100%
06365	9	9	0	0	100%	100%
06542	5	3	0	2	60%	100%
25 imgs	Total: 156	Total: 129	Total: 5	Total: 21	Avg: 84,58%	mAP: 91,33%

Table 4: GC, information based on the groundtruth images

It can be observed from the results, that by using gamma correction, the object detector was explicitly performing worse than with the original images. This is due to Gamma Correction lifting the already existing noise of every image by scaling the pixels, resulting in a noisy mess. **The performance of the model for the gamma corrected images is 83.22%**

YOLOv4 was unable to predict several objects within the background of the images, due to loss of information from the noisy image, Figure 42.



(a) Original Image (b) Gamma Correction (c) Expected Result

Figure 42: Brief visual comparison of noise levels, between the Original and enhanced image results. **(ExDark Image 01468)**

9.4 Data Analysis & Results: Histogram Equalization

	Histogram Equalization[4, 51]					
Image	**N. Groundtruth Detections**	**Correct Predictions**	**Incorrect Predictions**	**Missing Detections**	**Recall**	**Precision**
00253	6	5	0	1	83.33%	100%
00301	2	0	0	2	0%	0%
00405	9	9	0	0	100%	100%
01367	10	0	0	10	0%	0%
01437	10	3	0	7	30%	100%
01468	15	5	0	10	33.33%	100%
01489	13	8	0	5	61.53%	100%
01611	5	5	0	0	100%	100%
01764	15	5	1	8	38.46%	83.33%
02049	5	4	0	1	80%	100%
02446	9	4	0	5	44.44%	100%
02453	3	3	0	0	100%	100%
02594	8	8	0	0	100%	100%
02756	6	0	2	4	0%	0%
03052	3	1	0	1	50%	50%
03215	2	1	0	1	50%	100%
03278	1	1	0	0	100%	100%
05380	2	0	2	0	0%	0%
05951	2	2	0	0	100%	100%
06077	3	3	0	0	100%	100%
06248	3	3	0	0	100%	100%
06255	5	4	1	0	100%	80%
06298	5	4	0	1	80%	100%
06365	9	9	0	0	100%	100%
06542	5	0	0	5	0%	0%
25 imgs	Total: 156	Total: 89	Total: 4	Total: 61	Avg: 62.043%	mAP: 76.533%

Table 5: HE, information based on the groundtruth images

The results provided in Table 5 show that Histogram Equalization performs considerably worse than the rest of the enhancement algorithms, including the original images. Due to the extreme degradation of the algorithm, the resulting images rendered it impossible to predict any object using the YOLOv4 model.

The average recall and precision compared to the rest of the enhancement algorithm results are substantially lower. **The performance of the model for the equalized images is 57.79%**. This performance indicates a drastic loss in the quality of the images and the overall performance of the model whilst using HE.

The YOLOv4 result on the equalized image with the ID 01367 is displayed in Figure 43 to visualize and observe the major degradation that resulted from the Histogram Equalization algorithm. The equalized image in Figure 43 had reached a state where it rendered it impossible to detect any objects within it.

(a) Original (b) Histogram Equalization (c) Expected Result

Figure 43: Brief Visual comparison of an extremely degraded Image. **(ExDark Image 01367)**

9.5 Data Analysis & Results: KinD

	KinD - Kindling the Darkness α = 5.0[5]					
Image	**N. Groundtruth Detections**	**Correct Predictions**	**Incorrect Predictions**	**Missing Detections**	**Recall**	**Precision**
00253	6	6	0	0	100%	100%
00301	2	1	0	1	50%	100%
00405	9	8	0	1	88.89%	100%
01367	10	8	0	2	80%	100%
01437	10	5	0	5	50%	100%
01468	15	11	0	4	73.33%	100%
01489	13	7	1	5	58.33%	87.5%
01611	5	5	0	0	100%	100%
01764	15	14	0	1	93.33%	100%
02049	5	4	0	1	80%	100%
02446	9	5	0	4	55.55%	100%
02453	3	3	0	0	100%	100%
02594	8	8	0	0	100%	100%
02756	6	4	0	2	66.7%	100%
03052	3	1	1	1	50%	50%
03215	2	1	0	1	50%	100%
03278	1	1	0	0	100%	100%
05380	1	1	1	0	100%	50%
05951	2	2	0	0	100%	100%
06077	3	3	0	0	100%	100%
06248	3	3	0	0	100%	100%
06298	5	5	0	0	100%	100%
06365	9	9	0	0	100%	100%
06542	5	5	0	0	100%	100%
06255	5	4	1	0	100%	80%
25 imgs	Total: 156	Total: 124	Total: 4	Total: 28	Avg: 83.845%	mAP: 94.7%

Table 6: KinD, information based on the groundtruth images

By comparing the results shown in Table 6 with the original image results, Recall and Precision were decreased by a very small margin. This is due to KinD's noise reduction algorithm, negatively afflicting YOLOv4's performance on images with extremely low light conditions. In the latter sections, that exact issue will be analyzed even further. **The performance of the model with the KinD results is 79,48%**

9.6 Statistical Comparison & Evaluation

This section provides a clearer image of the statistical comparisons between each enhancement algorithm and the original images by averaging both Recall and Precision.

The Figure 44, provides a general understanding of the performance of each enhancement algorithm, including the original images, by calculating the increase or decrease of each result. The percentage is calculated by averaging recall and precision and calculating the percentage change for each result given by the Tables [3, 4, 5, 6]. In order to create the Figure 44, the following equation 9 was used to calculate the percentages for each result.

$$\textbf{Change} = \frac{\textbf{OriginalResult - GCResult}}{\textbf{GCResult}} \tag{9}$$

The percentage change is calculated by using the equation 9. If the "Change" is positive, then the original result has an increase compared to the GC result. If the "Change" is negative, then the original result has a decrease. Meaning that for the original result, an increase is better than the GC Result and a decrease is worse than the GC result. The same is applied to each enhancement result in Figure 44

1) Original Result Performance to

- Gamma Correction: `0.78% Worse`
- Histogram Equalization: `45.12% Worse`
- Kindling the Darkness: `5.52% Worse`

2) Gamma Correction Performance to

- Original Images: `0.77% Better`
- Histogram Equalization: `44.00% Worse`
- Kindling the Darkness: `4.7% Worse`

3) Histogram Equalization Performance to

- Original Images: `31.09% Better`
- Gamma Correction: `30.55% Better`
- Kindling the Darkness: `27.28% Better`

4) Kindling the Darkness (KinD) Performance to

- Original Images: `5.23% Better`
- Gamma Correction: `4.49% Better`
- Histogram Equalization: `37.53% Worse`

Figure 44: A comprehensive statistical comparison of the performance of each enhancement algorithm compared to each other, using the YOLOv4 performance results. Retrieved in Sections [9.2, 9.4, 9.3, 9.5]

Considering the results provided in Figure 44, it can be observed that the result from the original images, **on average,** performed better than any other enhancement algorithm. KinD and Gamma Correction, fall short by a small margin, while Histogram Equalization is outperformed by the original images by a very large margin, of about 30%. Gamma Correction was unable to outperform both the result from the original images and KinD. KinD outperformed the rest of the enhancement algorithms, excluding the original result.

10 Analysis Using Data Visualization

The previous subsections provide a clear understanding of the overall performance, on average, for each enhancement algorithm, including the original image results. This section shows the details to why the enhancement algorithms failed to produce a better result.

Starting with Histogram Equalization, which performed poorly, it will be important to visualize the contours as well as the color variation differences of the produced images (Sobel Derivatives). By generating the sobel derivative images, the difference between the intensity values will be drawn into the resulting image and showing the manifested noise as well as imbalances.

10.1 Data visualization of Equalized Images

This section contains information of images that had no correct detections or all of the detections were missing listed in table 5 data visualizations

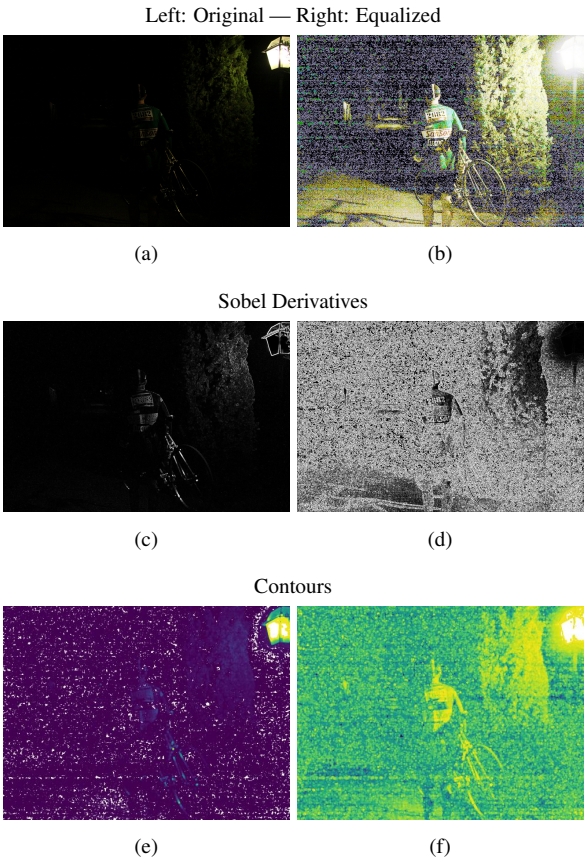


Figure 45: Comparison between the Original and Equalized Image. **(ExDark Image 00301)**

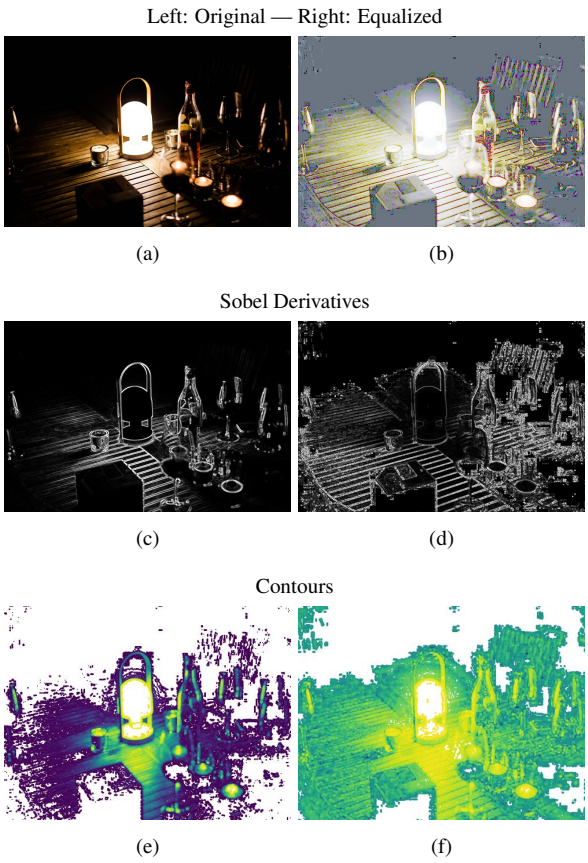


Figure 46: Comparison between the Original and Equalized Image. **(ExDark Image 01437)**

By equalizing the images using Histogram Equalization, it made it nearly impossible to detect anything in the images, considering the amount of noise generated. The Sobel Derivative algorithm visualized all the noise that was manifested, and the contour map, visualized the unbalanced contrast enhancement from Histogram Equalization.

10.2 Data Visualization of Gamma Correction

Considering the results given in table 4 the visualized images will generate a better result than the ones provided in Section 10.1.

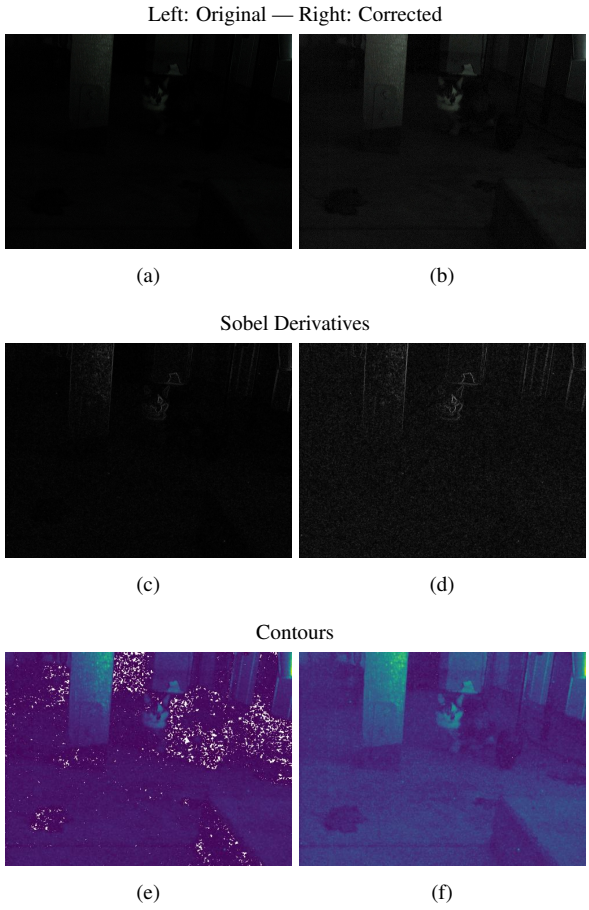


Figure 47: Comparison between the Original and Corrected Image. **(ExDark Image 03278)**

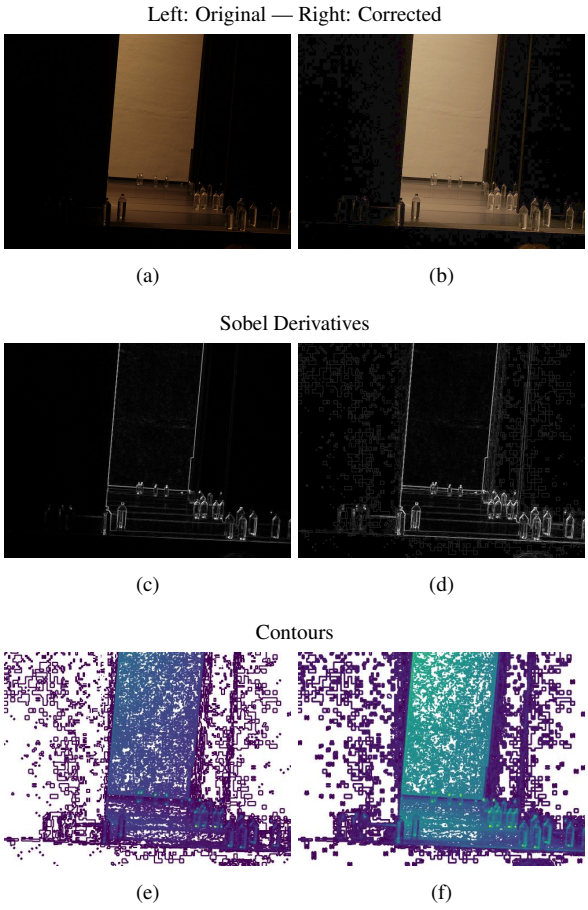


Figure 48: Comparison between the Original and Corrected Image. **(ExDark Image 01489)**

That's it! The results are very interesting and show an overall improvement in detail for both figures. The contours and Sobel derivative results shown in Figure 47 have improved as well. The contour map of the corrected image has revealed information that originally did not exist in the original contour map. The same applies to the Sobel derivative result, with the visibility of objects getting highlighted even better. Although, as mentioned before, the noise was uplifted by the algorithm after the intensity values were scaled up, reducing the quality of the image.

10.3 Data Visualization of KinD

It is expected that KinD will improve the color variation of several low-light-conditioned images and improve the overall result. Although it is also expected, considering the results given

in Table 6, that it will negatively affect the original image and YOLOv4's performance due to KinD's noise reduction algorithm. This section will provide further information on how KinD affected YOLOv4's performance negatively by a small margin and did not improve it.

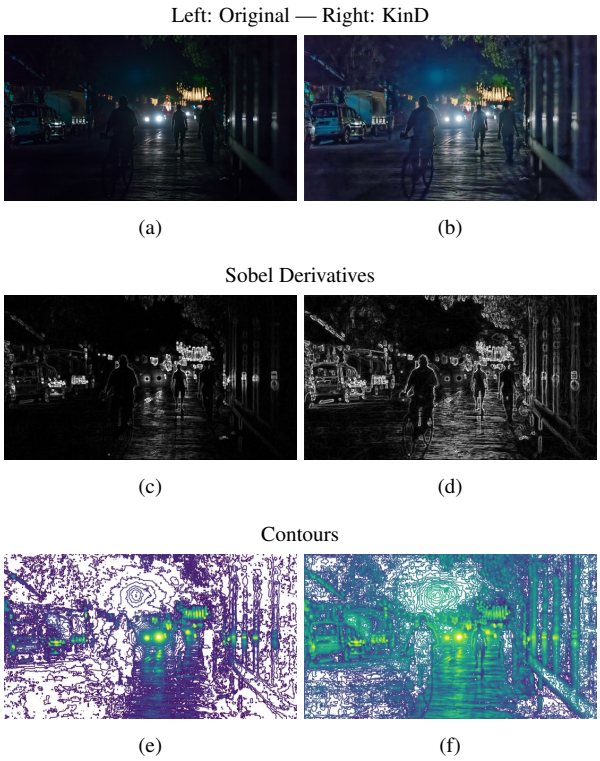


Figure 49: Comparison between the Original and KinD Image. **(Ex-Dark Image 00253)**

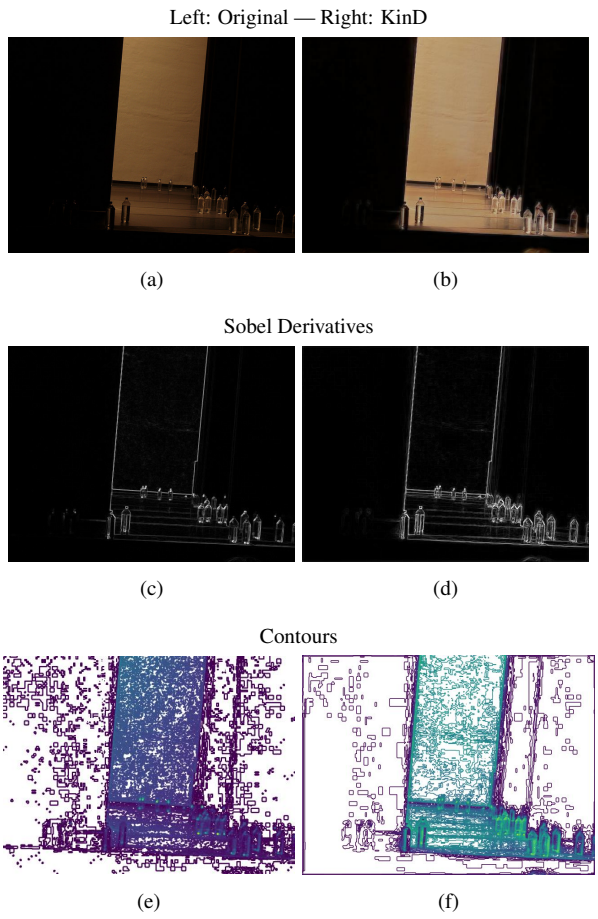
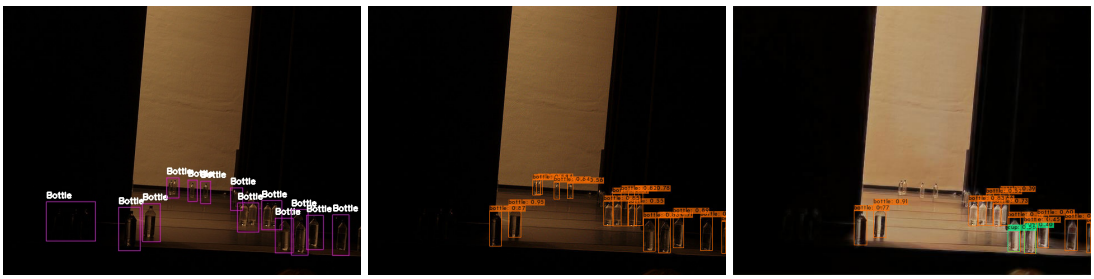


Figure 50: Comparison between the Original and KinD Image. **(Ex-Dark Image 01468)**

The contour map of the enhanced image in Figure 49, displays more color variations and higher intensity values. The image's results given in KinD's Table 6, are the same as the image's results in the Original Image Table 3. The same applies to the Sobel results, with the enhanced image providing higher levels of detail for the objects within the image. YOLOv4 performed well for both, without any statistical loss or gain.

The same applies for Figures 50, although, KinD's image results, Table 6, have a statistical loss compared to the original image results, Table 3. To quote the KinD developers, KinD "over-smoothed" the image to a point of loss of information. The contours and Sobel derivative images of Figure 50, display how KinD degraded the image by denoising it. The expected output of the Figure 50, is displayed in the below Figure 51



Expected (a) Original YOLOv4 (b) KinD YOLOv4

Figure 51: Original and KinD YOLOv4 result, compared to the Expected result. **ExDark Image 01468**

11 Conclusion

It is clear that YOLOv4 does not receive any major benefits in terms of performance from any of the given enhancement algorithms, mostly in extremely low light-conditioned images, due to several problems mentioned in past sections for each algorithm. Although in low to very low light images, YOLOv4 performs slightly better, improving the overall accuracy of the model.

The algorithms Gamma Correction and KinD, decreased the average Recall and precision by a small margin. Histogram equalization reduced Recall and precision, by a much larger margin, with Recall at 62% with an increased count of missing detections and Precision at 76.5%. As mentioned in Section 9.4 this is because Histogram Equalization distorts images with low light conditions, due to the lack of detail and color variation for the algorithm to process, resulting in complete distortion.

Gamma Correction, overall had a negative effect on YOLOv4's performance by a minimal margin compared to the original image results. This is due to the uplifting of the already existing noise obscuring any information that is of use to the model. This resulted in YOLOv4 producing unsatisfactory results compared to the original image results. The algorithm in certain images did improve the given results as presented in Table 4, meaning the efficiency of the algorithm is situational. This concludes that simply scaling the intensity values of the image is not enough for any object detector to perform increasingly better than before.

Histogram Equalization performed with the least amount of overall performance. Applying this algorithm under images captured in extremely low-light conditions, the images were degraded in such a state rendering it impossible to detect anything using the YOLOv4 model. Considering the results in Table 5, it is now clear and can be concluded that Histogram Equalization, is an algorithm that will drastically decrease the efficiency and performance of any object detector including YOLOv4. Various studies have been created to analyze improved versions of HE, such as RMSHE, RSIHE, RSWHE, and RSWHE-M [57] that preserve the brightness and contrast of the enhanced images more efficiently than HE can.

Although KinD uses CNN [36] architectures and deep learning techniques to enhance im-

ages, the evaluation results proved that KinD performed worse than the original image results and Gamma Correction a simple non-linear enhancement algorithm. As mentioned and analyzed in Section 9.5, this happens because of KinD's degradation removal algorithm. An improved version of KinD has been introduced. Zhang, Y., Guo, X., Ma, J. et al. state that the improved version of KinD "can alleviate visual defects (e.g. non-uniform spots and over-smoothing) left in KinD", where "over-smoothing" was the root of the problem where KinD was not performing so well with YOLOv4.

With the given results, it can be concluded that KinD and Gamma Correction, perform well in certain levels of low light conditions, although it is also expected that the improved version of KinD, KinD_Plus[58], will improve YOLOv4's performance on the enhanced low light images.

12 Future Work

This study proved to be useful to determine the effectiveness of various enhancement algorithms on the object detector YOLOv4, although there is more work to be done. More advanced enhancement algorithms can be implemented and evaluated with the help of the YOLOv4 model, including KinD plus [58]. Various other techniques such as noise or degradation removal can be explored to further increase the potential of this study and improve the effectiveness of various object detector models. In this study the only model that was evaluated is YOLOv4, evaluating more object detectors such as YOLOv7-Tiny [59] & YOLOv7 [60] will draw a clearer image on the performance of low-light enhancement algorithms on various object detectors.

References

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," 2020.

[2] Y. P. Loh and C. S. Chan, "Getting to Know Low-light Images with The Exclusively Dark Dataset," *Computer Vision and Image Understanding*, vol. 178, pp. 30–42, 2019.

[3] A. Rosebrock, "OpenCV Gamma Correction," Oct 5 2015.

[4] S. Bhattacharyya, "Histogram Equalization — a simple way to improve the contrast of your image," Oct 25 2019.

[5] Y. Zhang, J. Zhang, and X. Guo, "Kindling the Darkness: A Practical Low-light Image Enhancer," in *Proceedings of the 27th ACM International Conference on Multimedia*, ser. MM '19. New York, NY, USA: ACM, 2019, pp. 1632–1640. [Online]. Available: http://doi.acm.org/10.1145/3343031.3350926

[6] S. Charette, "DarkHelp, C++ wrapper library for Darknet," June 24 2022.

[7] OpenCV, “OpenCV modules.”

[8] D. Mpouziotas, “A Perceptron Implementation By Dimitrios Mp.” May 17 2021.

[9] S. Verma, “Implementing the Perceptron Algorithm in Python,” *Machine Learning From Scratch: Part 6*, Apr 17 2021.

[10] D. A. Forsyth and J. Ponce, *Computer Vision - A Modern Approach, Second Edition.* Pitman, 2012.

[11] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, “A survey of modern deep learning based object detection models,” 2021. [Online]. Available: https://arxiv.org/abs/2104.11892

[12] J. Redmon, “Darknet: Open Source Neural Networks in C,” 2013–2016.

[13] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” 2015. [Online]. Available: https://arxiv.org/abs/1511.08458

[14] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “Scaled-YOLOv4: Scaling Cross Stage Partial Network,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 13 029–13 038.

[15] C. Chen, Q. Chen, J. Xu, and V. Koltun, “Learning to see in the dark,” 2018. [Online]. Available: https://arxiv.org/abs/1805.01934

[16] C. Kanellakis, S. Sharif Mansouri, M. Castaño, P. Karvelis, D. Kominiak, and G. Nikolakopoulos, “Where to look: a collection of methods formav heading correction in underground tunnels,” *IET Image Processing*, vol. 14, no. 10, pp. 2020–2027, 2020. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-ipr.2019.1423

[17] C. Kanellakis, P. Karvelis, and G. Nikolakopoulos, “Image enhancing in poorly illuminated subterranean environments for mav applications: A comparison study,” in *Computer Vision Systems*, D. Tzovaras, D. Giakoumis, M. Vincze, and A. Argyros, Eds. Cham: Springer International Publishing, 2019, pp. 511–520.

[18] G. Li, Y. Yang, X. Qu, D. Cao, and K. Li, “A deep learning based image enhancement approach for autonomous driving at night,” *Knowledge-Based Systems*, 11 2020.

[19] S. Rahman, M. M. Rahman, M. Abdullah-Al-Wadud, G. D. Al-Quaderi, and M. Shoyaib, “An adaptive gamma correction for image enhancement,” *EURASIP Journal on Image and Video Processing*, vol. 35, 10 2016.

[20] O. Patel, Y. Maravi, and S. Sharma, “A comparative study of histogram equalization based image enhancement techniques for brightness preservation and contrast enhancement,” *Signal & Image Processing : An International Journal*, vol. 4, 11 2013.

[21] S. Charette, “What about negative samples?”

[22] B. Hong, Y. Zhou, H. Qin, Z. Wei, H. Liu, and Y. Yang, “Few-shot object detection using multimodal sensor systems of unmanned surface vehicles,” *Sensors*, vol. 22, no. 4, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/4/1511

[23] J. Redmon, A. Bochkovskiy, and S. Charette, “Darknet GitHub.”

[24] S. Charette, “DarkMark C++ GUI Tool for Darknet - Code Run,” 2019 - 2023.

[25] ——, “DarkMark GitHub Page.”

[26] ——, “Stéphane’s Darknet FAQ,” Apr 2022.

[27] A. A. Awan, “A Complete Guide to Data Augmentation,” *Learn about data augmentation techniques, applications, and tools with a TensorFlow and Keras tutorial.*, November 2022.

[28] J. Liu, D. Xu, W. Yang, M. Fan, and H. Huang, “Benchmarking Low-Light Image Enhancement and Beyond,” *International Journal of Computer Vision*, vol. 129, 04 2021.

[29] E. A. da Silva and G. V. Mendonça, “4 - digital image processing,” in *The Electrical Engineering Handbook*, W.-K. CHEN, Ed. Burlington: Academic Press, 2005, pp. 891–910. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780121709600500645

[30] D. E. Kaplan, *Introduction to Psychology – Chapter 4.2 Seeing*, Feb 16 2017.

[31] B. Ly, E. Dyer, J. Feig, A. Chien, and S. Bino, “Research Techniques Made Simple: Cutaneous Colorimetry: A Reliable Technique for Objective Skin Color Measurement,” *The Journal of investigative dermatology*, vol. 140, pp. 3–12.e1, January 2020.

[32] C. Derouet and B. Parzysz, “How can histograms be useful for introducing continuous probability distributions?” *ZDM*, vol. 48, pp. 757 – 773, Mar. 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03618056

[33] “Introduction to Contour Maps,” November 11 2022.

[34] OpenCV, “Contours in OpenCV.”

[35] ——, “Sobel Derivatives,” Jun 24 2022.

[36] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,” Dec 15 2018.

[37] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[38] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," 2014. [Online]. Available: https://arxiv.org/abs/1405.0312

[39] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN," 2019. [Online]. Available: https://arxiv.org/abs/1911.11929

[40] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018. [Online]. Available: https://arxiv.org/abs/1804.02767

[41] C. Kozyrkov, "What is "Ground Truth" in AI? (A warning.)," *A demo that shows why you shouldn't treat AI like a magical box of magic*, February 2020.

[42] C. L. of Ornithology, "eBird - The Cornell Lab of Ornithology."

[43] M. Tkachenko, M. Malyuk, A. Holmanyuk, and N. Liubimov, "Label Studio: Data labeling software," 2020-2022, open source software available from https://github.com/heartexlabs/label-studio. [Online]. Available: https://github.com/heartexlabs/label-studio

[44] G. Reina, R. Panchumarthy, S. Thakur, A. Bastidas, and S. Bakas, "Systematic evaluation of image tiling adverse effects on deep learning semantic segmentation," *Frontiers in Neuroscience*, vol. 14, p. 65, 02 2020.

[45] G. Andrews, "What is synthetic data?" *Synthetic data generated from computer simulations or algorithms provides an inexpensive alternative to real-world data that's increasingly used to create accurate AI models*, June 8 2021. [Online]. Available: https://blogs.nvidia.com/blog/2021/06/08/what-is-synthetic-data/

[46] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, 2020. [Online]. Available: https://www.mdpi.com/2078-2489/11/2/125

[47] S. Charette, Apr 2022. [Online]. Available: https://www.ccoderun.ca/programming/darknet_faq/

[48] J. Solawetz, "What is Mean Average Precision (mAP) in Object Detection?" *The computer vision community has converged on the metric mAP to compare the performance of object detection systems. In this post, we will dive into the intuition behind how mean Average Precision (mAP) is calculated and why mAP has become the preferred metric for object detection models.*, May 6 2020.

[49] F. Liang, "Evaluating the Performance of Machine Learning Models," *One of the most common and quickest ways to evaluate a model*, Apr 18 2020.

[50] D. Nikolaiev, "Overfitting and Underfitting Principles," *Understand basic principles of underfitting and overfitting and why you should use particular techniques to deal with them*, Nov 2 2021.

[51] OpenCV, "Histogram Equalization," Jun 24 2022.

[52] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising with block-matching and 3D filtering," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 6064, pp. 354–365, February 2006.

[53] Y. P. Loh and C. S. Chan, "Getting to Know Low-light Images with The Exclusively Dark Dataset," *Computer Vision and Image Understanding*, vol. 178, pp. 30–42, 2019.

[54] M. Dimitrios, "Algorithms implemented in this Thesis," june 2022.

[55] S. Charette, "Using CUDA with OpenCV," Apr 2022.

[56] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[57] O. Patel, Y. Maravi, and S. Sharma, "A Comparative Study of Histogram Equalization Based Image Enhancement Techniques for Brightness Preservation and Contrast Enhancement," *Signal & Image Processing : An International Journal*, vol. 4, 11 2013.

[58] Y. Zhang, X. Guo, J. Ma, W. Liu, and J. Zhang, "Beyond brightening low-light images," *International Journal of Computer Vision*, vol. 129, no. 4, pp. 1013–1037, Apr 2021. [Online]. Available: https://doi.org/10.1007/s11263-020-01407-x

[59] S. Hu, F. Zhao, H. Lu, Y. Deng, J. Du, and X. Shen, "Improving yolov7-tiny for infrared and visible light image object detection on drones," vol. 15, no. 13, 2023. [Online]. Available: https://www.mdpi.com/2072-4292/15/13/3214

[60] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022.

Appendices

A Gamma Correction

Programming Language: Python
Requirements: opencv, numpy, Backend.py file

```python
import numpy as np
from src.Backend import Backend
import cv2 as cv


inputFolder = "input/"
outputFolder = "out/"

backend = Backend()


def GammaCorrectionMulti():
    Gamma = 2
    inverseGamma = 1 / Gamma
    for index, img in enumerate(backend.imagesBuffer):
        table = [((i / 255) ** inverseGamma) * 255 for i in range(256)]

        table = np.array(table, np.uint8)

        gammaCorrected = cv.LUT(img, table)
        backend.writeImages(index, gammaCorrected,
        ↪  backend.imageName[index])
    print("\n\nImage Filtering Complete. View outputGCmulti folder...")


def run():
    backend.setInputPath(inputFolder + "inputGC")
    backend.setOutPath(outputFolder + "outputGC")

    backend.run()
    GammaCorrectionMulti()


if __name__ == "__main__":
    run()
```

M Histogram Equalization

Programming Language: Python
Requirements: opencv, numpy, Backend.py file

```python
from src.Backend import Backend
import cv2 as cv
import numpy as np


inputFolder = "input/"
outputFolder = "out/"

backend = Backend()

def histogramEqualizationMulti():
    for index, img in enumerate(backend.imagesBuffer):
        b, g, r = cv.split(img)
        h_b, bin_b = np.histogram(b.flatten(), 256, [0, 256])
        h_g, bin_g = np.histogram(g.flatten(), 256, [0, 256])
        h_r, bin_r = np.histogram(r.flatten(), 256, [0, 256])
        # calculate cdf
        cdf_b = np.cumsum(h_b)
        cdf_g = np.cumsum(h_g)
        cdf_r = np.cumsum(h_r)

        # mask all pixels with value=0 and replace it with mean of the
        ↪  pixel values
        cdf_m_b = np.ma.masked_equal(cdf_b, 0)
        cdf_m_b = (cdf_m_b - cdf_m_b.min()) * 255 / (cdf_m_b.max() -
        ↪  cdf_m_b.min())
        cdf_final_b = np.ma.filled(cdf_m_b, 0).astype('uint8')

        cdf_m_g = np.ma.masked_equal(cdf_g, 0)
        cdf_m_g = (cdf_m_g - cdf_m_g.min()) * 255 / (cdf_m_g.max() -
        ↪  cdf_m_g.min())
        cdf_final_g = np.ma.filled(cdf_m_g, 0).astype('uint8')
        cdf_m_r = np.ma.masked_equal(cdf_r, 0)
        cdf_m_r = (cdf_m_r - cdf_m_r.min()) * 255 / (cdf_m_r.max() -
        ↪  cdf_m_r.min())
        cdf_final_r = np.ma.filled(cdf_m_r, 0).astype('uint8')
        # merge the images in the three channels
        img_b = cdf_final_b[b]
        img_g = cdf_final_g[g]
        img_r = cdf_final_r[r]
```

```
37
38          img_out = cv.merge((img_b, img_g, img_r))
39          # validation
40          equ_b = cv.equalizeHist(b)
41          equ_g = cv.equalizeHist(g)
42          equ_r = cv.equalizeHist(r)
43          equ = cv.merge((equ_b, equ_g, equ_r))
44          backend.writeImages(index, equ, backend.imageName[index])
45
46      print("\n\nImage Filtering Complete. View outputHE folder...")
47
48
49  def run():
50      backend.pathInput = inputFolder + "inputHE"
51      backend.pathOutput = outputFolder + "outputHE"
52      backend.run()
53
54      histogramEqualizationMulti()
55
56
57  if __name__ == "__main__":
58      run()
```

N Backend for GC & HE

Programming Language: Python

```python
import os
import os.path

# Change this to True if you do not want Progress Bar visuals, otherwise > pip install curses
disableProgressBar = False

if not disableProgressBar:
    try:
        import curses
    except ImportError:
        print("\033[91mModuleNotFoundError: No module named 'curses'\033[92m\n"
              "This module is primarily for visuals and can be disabled\n"
              "Enable if you need the Progress Bar visuals\n"
              "\nPossible Solutions:\033[0m\n"
              "  > Disable Progress Bar in src/Backend.py\n"
              "  > pip install curses")
        exit(-1)

try:
    import cv2 as cv
except ImportError:
    print("ModuleNotFoundError: No module named 'cv2'\n"
          "Please Install OpenCV to your Python environment")
    exit(-1)

barChar = "="

bufferedString = ""
total = 0

def setMaxLimit(max):
    global total
    total = max


def progressBar(current, stdScr):
    try:
        completed = 100 * (current / float(total))
    except ZeroDivisionError:
```

```
        completed = 0
    barProgress = barChar * int(completed) + "-" * (100 -
    ↪  int(completed))

    stdScr.addstr(0, 117, f"| {completed:.0f} %")
    curses.init_pair(1, 22, -1)
    stdScr.addstr(0, 0, f"Total Progress: |")
    stdScr.addstr(0, 17, f"{barProgress}", curses.color_pair(1))
    stdScr.addstr(1, 0, bufferedString)
    stdScr.refresh()
    if completed >= 100.0:
        print(f"Total Progress: |\033[92m{barProgress}\033[0m|
        ↪  {completed:.0f} %")

class Backend:
    def __init__(self):
        if not disableProgressBar:
            self.stdScr = curses.initscr()
            curses.noecho()
            curses.nocbreak()

        self.imagesBuffer = []
        self.imageName = []
        self.pathInput = ""
        self.pathOutput = ""
        self.outputTemplate = "/"

        self.numFiles = 0
        self.numProcesses = 0
        self.currentProcess = 0

        self.files = []


    def verify(self):
        if not disableProgressBar:
            global bufferedString
            bufferedString = "Verifying Files...\n"
            self.numFiles = len(self.files)
            self.numProcesses = self.numFiles + self.numFiles * 2
            setMaxLimit(self.numProcesses)

            progressBar(0, self.stdScr)

        for i, file in enumerate(self.files):
            if not (file.endswith(".png")) and not
            ↪  (file.endswith(".jpg")) and not (file.endswith(".JPG"))
            ↪  and not \
                    (file.endswith(".JPEG")) and not
                    ↪  (file.endswith(".jpeg")):
```

```
85                  print("Reading incorrect or unsupported file formats.
                    ↪  Supported file formats (jpg, jpeg, png")
86                  print("ERROR AT FILE:", file)
87                  exit(-1)
88              if file.endswith(".png") or file.endswith(".PNG"):
89                  os.rename(os.path.join(self.pathInput, file),
                    ↪  os.path.join(self.pathInput, ''.join([file])))
90              elif file.endswith(".jpg") or file.endswith(".JPG"):
91                  os.rename(os.path.join(self.pathInput, file),
                    ↪  os.path.join(self.pathInput, ''.join([file])))
92              elif file.endswith(".JPEG"):
93                  os.rename(os.path.join(self.pathInput, file),
                    ↪  os.path.join(self.pathInput, ''.join([file])))
94              self.currentProcess += 1
95              if not disableProgressBar:
96                  progressBar(self.currentProcess, self.stdScr)
97
98
99      def readingImages(self):
100         if not disableProgressBar:
101             global bufferedString
102             bufferedString = "Reading Images...\n"
103             # sys.stdin.flush()
104             progressBar(self.currentProcess, self.stdScr)
105         for i, file in enumerate(self.files):
106             if i == self.numFiles:
107                 return
108
109             name = "/" + file
110             img = cv.imread(self.pathInput + name)
111
112             self.imagesBuffer.append(img)
113             self.imageName.append(name)
114             if not disableProgressBar:
115                 self.currentProcess += 1
116                 progressBar(self.currentProcess, self.stdScr)
117         if not disableProgressBar:
118             bufferedString = "Applying Filter..."
119
120
121     def writeImages(self, index, img, file):
122         if file.endswith(".png") or file.endswith(".PNG"):
123             cv.imwrite(self.pathOutput + file, img)
124         elif file.endswith(".jpg") or file.endswith(".JPG"):
125             cv.imwrite(self.pathOutput + file, img)
126         elif file.endswith(".JPEG"):
127             cv.imwrite(self.pathOutput + file, img)
128         if not disableProgressBar:
129             self.currentProcess += 1
130             progressBar(self.currentProcess, self.stdScr)
131         return
```

```
132
133
134     def run(self):
135         if not (os.path.exists(self.pathInput)) or not
            ↪  os.path.exists(self.pathOutput):
136             print("Input or Output Path does not Exist")
137             exit(-1)
138
139         self.files = os.listdir(self.pathInput)
140
141         if not disableProgressBar:
142             curses.start_color()
143             curses.use_default_colors()
144         self.verify()
145         self.readingImages()
146
147
148     def setInputPath(self, input):
149         self.pathInput = input
150
151
152     def setOutPath(self, input):
153         self.pathOutput = input
```

O Generate ExDark's Groundtruth Images

Programming Language: Python
Requirements: opencv

```python
import os
import cv2 as cv


datasetPath = "ExDark/Dataset"
groundtruthPath = "ExDark/Groundtruth"

subFoldersGT = [f for f in os.listdir(groundtruthPath)]
subFoldersData = [f for f in os.listdir(datasetPath)]
CategoryPathsGTruth = []
CategoryPathsData = []
for subFolder in subFoldersGT:
    CategoryPathsGTruth.append(groundtruthPath + "/" + subFolder)
for subFolder in subFoldersData:
    CategoryPathsData.append(datasetPath + "/" + subFolder)

CategoryPathsData.sort()
CategoryPathsGTruth.sort()

ClassColors = [(0, 0, 255), (0, 255, 0), (255, 0, 0), (255, 218, 90), (0, 110, 204), (255, 110, 192),
          (192, 201, 255), (0, 255, 255), (255, 125, 111), (255, 255, 255), (129, 180, 128), (0, 48, 122)]

count = 0
for (pathGT, pathData) in zip(CategoryPathsGTruth, CategoryPathsData):
    FilesGTruth = [f for f in os.listdir(pathGT)]
    FilesDataset = [f for f in os.listdir(pathData)]
    FilesGTruth.sort()
    FilesDataset.sort()
    for (fileGT, fileData) in zip(FilesGTruth, FilesDataset):
        FileName = fileGT.replace(".txt", "")

        # if FileName != fileData:
        #     print("Groundtruth: ", pathGT + "/" + FileName, "\t fileDataset: ", pathData + "/" + fileData)
        if FileName != fileData:
            print("Incorrect File Matching!")
            print("Groundtruth: ", FileName, "\tDataset: ", fileData)
        else:
```

```
38              fGT = open(pathGT + "/" + fileGT)
39              fGT.readline()  # Skips first line
40              fData = cv.imread(pathData + "/" + fileData)
41              HEIGHT, WIDTH, _ = fData.shape
42              Lines = fGT.readlines()
43              for line in Lines:
44                  Data = line.split()
45                  ClassType = Data[0]
46                  start_point = (int(Data[1]), int(Data[2]))
47                  if ClassType == "Bicycle":
48                      fData = cv.rectangle(fData, start_point,
                        ↪  (int(Data[3]) + int(Data[1]), int(Data[4]) +
                        ↪  int(Data[2])), ClassColors[0], 2)
49                  if ClassType == "Boat":
50                      fData = cv.rectangle(fData, start_point,
                        ↪  (int(Data[3]) + int(Data[1]), int(Data[4]) +
                        ↪  int(Data[2])), ClassColors[1], 2)
51                  if ClassType == "Bottle":
52                      fData = cv.rectangle(fData, start_point,
                        ↪  (int(Data[3]) + int(Data[1]), int(Data[4]) +
                        ↪  int(Data[2])), ClassColors[2], 1)
53                  if ClassType == "Bus":
54                      fData = cv.rectangle(fData, start_point,
                        ↪  (int(Data[3]) + int(Data[1]), int(Data[4]) +
                        ↪  int(Data[2])), ClassColors[3], 2)
55                  if ClassType == "Car":
56                      fData = cv.rectangle(fData, start_point,
                        ↪  (int(Data[3]) + int(Data[1]), int(Data[4]) +
                        ↪  int(Data[2])), ClassColors[4], 1)
57                  if ClassType == "Cat":
58                      fData = cv.rectangle(fData, start_point,
                        ↪  (int(Data[3]) + int(Data[1]), int(Data[4]) +
                        ↪  int(Data[2])), ClassColors[5], 2)
59                  if ClassType == "Chair":
60                      fData = cv.rectangle(fData, start_point,
                        ↪  (int(Data[3]) + int(Data[1]), int(Data[4]) +
                        ↪  int(Data[2])), ClassColors[6], 2)
61                  if ClassType == "Cup":
62                      fData = cv.rectangle(fData, start_point,
                        ↪  (int(Data[3]) + int(Data[1]), int(Data[4]) +
                        ↪  int(Data[2])), ClassColors[7], 1)
63                  if ClassType == "Dog":
64                      fData = cv.rectangle(fData, start_point,
                        ↪  (int(Data[3]) + int(Data[1]), int(Data[4]) +
                        ↪  int(Data[2])), ClassColors[8], 2)
65                  if ClassType == "Table":
66                      fData = cv.rectangle(fData, start_point,
                        ↪  (int(Data[3]) + int(Data[1]), int(Data[4]) +
                        ↪  int(Data[2])), ClassColors[9], 2)
67                  if ClassType == "People":
```

```
68                      fData = cv.rectangle(fData, start_point,
                        ↪  (int(Data[3]) + int(Data[1]), int(Data[4]) +
                        ↪  int(Data[2])), ClassColors[10], 1)
69                  if ClassType == "Motorbike":
70                      fData = cv.rectangle(fData, start_point,
                        ↪  (int(Data[3]) + int(Data[1]), int(Data[4]) +
                        ↪  int(Data[2])), ClassColors[11], 2)
71                  cv.putText(fData,
72                             ClassType,
73                             tuple(map(sum, zip(start_point, (0, -7)))),
74                             cv.FONT_HERSHEY_SIMPLEX,
75                             0.5,
76                             (255, 255, 255),
77                             2)
78              cv.imwrite("Groundtruth_Images/" + FileName, fData)
```

P Determine Low Light Level using means

Programming Language: Python
Requirements: opencv, numpy

```python
import cv2
import os
import numpy as np

pathInput = "../images/"
pathOutput = "../images/"
files = os.listdir(pathInput)
Images = []
files.sort()
for index, file in enumerate(files):
    print("Image ", index, ": ", file)


def ReadImages():
    for index, file in enumerate(files):
        Images.append(cv2.imread(pathInput + file))
    if Images:
        print("Files Read...\n")
    else:
        print("Folder Empty")
        exit(-1)


def getLuminanceChannel(index, img):
    LABimg = cv2.cvtColor(img, None, cv2.COLOR_BGR2GRAY)
    # cv2.imwrite(pathOutput + "LumChannel" + str(index) + ".jpg",
    ↪  LABimg[:, :, 0])
    meanLum = np.mean(LABimg)
    return meanLum


def checkLuminanceThreshold(index, img):
    LChannel = getLuminanceChannel(index, img)
    # lumsum = 0
    # count = 0
    # for x in range(len(LChannel)):
    #     for y in range(len(LChannel[x])):
    #         lumsum += int(LChannel[x][y])
    #         count += 1
    # rows = len(LChannel)
```

```
40      # cols = len(LChannel[0])
41
42      # AverageLum = lumsum / (cols * rows)
43      return LChannel
44
45
46  def begin():
47      for index, img in enumerate(Images):
48          luminance = checkLuminanceThreshold(index, img)
49          if luminance > 100:
50              message = "Normal Lighting"
51          elif luminance > 85 and luminance <= 100:
52              message = "Low Lighting"
53          elif luminance >= 75 and luminance <= 85:
54              message = "Very Low Lighting"
55          elif luminance < 75:
56              message = "Extremely Low Lighting"
57          print("Image ", files[index], " Luminance: ", luminance,
            ↪  "\nLevel: ", message, "\n")
58
59  ReadImages()
60  begin()
```

Q Generate Image Histograms 1 (Code to show RGB channels separately)

Programming Language: Python
Requirements: opencv, matplotlib, numpy

```python
import cv2 as cv
import os
import numpy as np
import matplotlib.pyplot as plt

inputFolder = "Input - GenerateHist/"
outputFolder = "Output - GenerateHist/"
ImageList = os.listdir(inputFolder)
colorList = ('b', 'g', 'r')

for idx, imgName in enumerate(ImageList):
    img = cv.imread(inputFolder + imgName)
    fig = plt.figure()

    channels = img.shape[2]
    if channels > 1:
        for i, col in enumerate(colorList):
            histr = cv.calcHist([img], [i], None, [256], [0, 256])
            plt.plot(histr, color=col)
            plt.xlim([0, 256])
        plt.savefig(outputFolder + "Histogram_" + imgName)
    plt.clf()
```

R Generate Image Histograms 2 (Simple Histogram)

Programming Language: Python
Requirements: opencv

```python
import os, os.path
import sys
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

img = cv.imread("2015_02453.jpg")
b, g, r = cv.split(img)
plt.hist(cv.cvtColor(img, cv.COLOR_BGR2GRAY).flatten(), 256, [0, 256], ec='k')
plt.title("Original Image Histogram")
plt.savefig('OriginalHist.jpg')

h_b, bin_b = np.histogram(b.flatten(), 256, [0, 256])
h_g, bin_g = np.histogram(g.flatten(), 256, [0, 256])
h_r, bin_r = np.histogram(r.flatten(), 256, [0, 256])

cdf_b = np.cumsum(h_b)
cdf_g = np.cumsum(h_g)
cdf_r = np.cumsum(h_r)


cdf_m_b = np.ma.masked_equal(cdf_b, 0)
cdf_m_b = (cdf_m_b - cdf_m_b.min()) * 255 / (cdf_m_b.max() - cdf_m_b.min())
cdf_final_b = np.ma.filled(cdf_m_b, 0).astype('uint8')

cdf_m_g = np.ma.masked_equal(cdf_g, 0)
cdf_m_g = (cdf_m_g - cdf_m_g.min()) * 255 / (cdf_m_g.max() - cdf_m_g.min())
cdf_final_g = np.ma.filled(cdf_m_g, 0).astype('uint8')
cdf_m_r = np.ma.masked_equal(cdf_r, 0)
cdf_m_r = (cdf_m_r - cdf_m_r.min()) * 255 / (cdf_m_r.max() - cdf_m_r.min())
cdf_final_r = np.ma.filled(cdf_m_r, 0).astype('uint8')

img_b = cdf_final_b[b]
img_g = cdf_final_g[g]
img_r = cdf_final_r[r]

```

```
37 img_out = cv.merge((img_b, img_g, img_r))
38
39 equ_b = cv.equalizeHist(b)
40 equ_g = cv.equalizeHist(g)
41 equ_r = cv.equalizeHist(r)
42 equ = cv.merge((equ_b, equ_g, equ_r))
43 plt.figure()
44
45 plt.hist(cv.cvtColor(equ, cv.COLOR_BGR2GRAY).ravel(), 256, [0, 256],
   ↪  ec='k')
46 plt.title("Equalized Image Histogram")
47 plt.savefig('EqualizedHist.jpg')
48 plt.show()
```

S Data Augmentation

Programming Language: Python
Requirements: albumentations, opencv

```python
import albumentations as album
import cv2
import random
import os
from pathlib import Path

inputDir = "images/"
outputDir = "out/"
images = os.listdir("images")

DatasetPercent = 0.1
numImages = len(images)

startIndex = random.randint(0, numImages - int(numImages * DatasetPercent))
endIndex = startIndex + int(numImages * DatasetPercent)
print("Start: ", startIndex)
print("End: ", endIndex)

count = 0
for file in images[startIndex:endIndex]:
    imageName = file.split(".jpg")

    count += 1

    filePath = inputDir + file
    image = cv2.imread(filePath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    randLower = random.randint(10, 30)
    randUpper = random.randint(randLower, 30)
    # print("Rand Lower: ", randLower, "\nRand Upper: ", randUpper)

    transform =  album.Compose([
        album.Blur(blur_limit=(3, 7), always_apply=True, p=0.9),

        album.HorizontalFlip(p=0.5),

        album.ImageCompression(quality_lower=randLower, quality_upper=randUpper, p=1),
```

```
        # album.CLAHE(6.0, (10, 10), p=1.0),

        # album.Superpixels(p_replace=0.5, n_segments=1, p=1.0)

        # album.ChannelShuffle(p=.5),

        album.MedianBlur(blur_limit=7, p=0.5),

        album.RandomBrightnessContrast(p=0.4),
    ])


    transformed = transform(image = image)
    transformed_image = transformed["image"]


    transformed_image = cv2.cvtColor(transformed_image,
    ↪  cv2.COLOR_RGB2BGR)
    cv2.imwrite(outputDir+imageName[0] + "_Augmented.jpg",
    ↪  transformed_image)

print(count)
```

T Perceptron with Data Visualization

Programming Language: Python
Requirements: pandas, numpy, matplotlib (pyplot)

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import pandas as pd
import random
#Copyright (c) 2022, Chilled Ferrum All rights reserved.

# Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

# Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

# Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

# All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the ChilledFerrum.

# Neither the name of the ChilledFerrum nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

# THIS SOFTWARE IS PROVIDED BY ChilledFerrum AS IS
# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
# THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
# IN NO EVENT SHALL ChilledFerrum BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
# OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
```

```
# OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


# # Parameters...
# Play around with the Parameters Here is some hints

# urlDataset = 'dataset1.csv'
# n_rows = 100
# n_inputs = 3
# act_outputs = [1, -1]
# For the 3D dataset change to act_outputs = ['C1', 'C2']
# For the 3D dataset change urlDataset = 'dataset3D.csv'
# For the 3D dataset change n_inputs = 3



urlDataset = 'dataset1.csv'  # Dataset URL
# Change this to train the dataset faster
learning_rate = 0.05
# Number of rows in the dataset
n_rows = 100
# Number of Inputs
n_inputs = 2  # Changes with the dataset
# Number of Epochs
seasons = 100
# Activation Function Outputs [First , Second]. Change this according to the dataset that is being used
act_outputs = [1, -1]

# Set Target Label Name...
t_label = 'Y'
# Set Label Starting Character
input_label = 'X'

# Execution Process parameters...
# View Real Time Plots (This will take time to finish according to the computer's process speed)
ViewRealTimePlots = True

# Stop at Set Seasons if Seasons is 10 it will stop at season 10 if StopAtSetSeasons = True
StopAtSetSeasons = True

# Use My Dynamic Learning Rate method
useDynamicLR = True
if useDynamicLR:
    # Higher LRrate means higher Learning Rate values, 2D recommended [0.01 - 0.015] 3D recommended at 0.2 +/-
    # LRrate is based on how big the linear Output is
```

```
    LRrate = 0.01
    DynamicLearningRate = learning_rate

# Use weight & bias Randomization
useWeightRandomization = True

weights = np.zeros(n_inputs)  # +1 is for the included weight for the bias
bias = 1

# Randomize weights & bias...
if useWeightRandomization:
    for i in range(0, n_inputs):
        weights[i] = random.uniform(-2.0, 2.0)
        if weights[i] == 0:
            weights[i] += 0.1
    bias = random.uniform(-2.0, 2.0)


def initDataset(UrlDataset, n):
    data = pd.read_csv(UrlDataset, nrows=n)
    return data


# Returns 0 if the function of the linear output is greater or equal than 0
def activationFunc(Y):
    return 1.0 if Y >= 0 else 0.0


# Load Data
dataset = initDataset(urlDataset, n_rows)
dataset = pd.DataFrame(dataset)
dataset = dataset.sample(frac=1)

df_target = pd.DataFrame(dataset, columns=[t_label])
target = df_target.to_numpy()


# Convert target outputs to 1 & 0
for idx, x in enumerate(target):
    if x == act_outputs[0]:
        target[idx] = 1.0
    elif x == act_outputs[1]:
        target[idx] = 0.0
# Invert StopAtSetSeasons (Works like a switch button)
StopAtSetSeasons = np.invert(StopAtSetSeasons)

Iter = 0
dfX_train = pd.DataFrame()
for col in dataset:
```

```
    if col != t_label:
        Iter += 1
        label = input_label + str(Iter)
        dfX_train[Iter] = pd.DataFrame(dataset, columns=[label])

X_train = dfX_train.to_numpy()

if n_inputs == 2:
    x = np.arange(0, np.max(X_train[:, [0]]), 0.1)
    y = -(weights[0] / weights[1]) * x + (-bias / weights[1])
    if ViewRealTimePlots:
        plt.ion()
        fig = plt.figure()
        ax = fig.add_subplot()
        ax.set_xlim([min(X_train[:, [0]]) - 2, max(X_train[:, [0]]) +
        ↪  2])
        ax.set_ylim([min(X_train[:, [1]]) - 2, max(X_train[:, [1]] +
        ↪  2)])

        line, = ax.plot(x, y)

# Activates only if n_inputs == 3
elif n_inputs == 3:
    x = np.arange(np.min(X_train[:, [0]]), np.max(X_train[:, [0]]),
    ↪  0.1)
    y = np.arange(np.min(X_train[:, [1]]), np.max(X_train[:, [1]]),
    ↪  0.1)
    x, y = np.meshgrid(x, y)
    if ViewRealTimePlots:
        plt.ion()
        fig = plt.figure()
        ax = fig.add_subplot(projection='3d')
        ax.set_xlim([min(X_train[:, [0]]) - 2, max(X_train[:, [0]]) +
        ↪  2])
        ax.set_ylim([min(X_train[:, [1]]) - 2, max(X_train[:, [1]] +
        ↪  2)])

print("Starting Weights: ", weights, " bias: ", bias)
predicted = np.ones(n_rows)
current_season = 0
currently_predicted = np.zeros(n_rows)
while current_season < seasons:
    miss = 0
    hit = 0
    for current_iter, xi in enumerate(X_train):
        linear_output = 0
        for i in range(0, n_inputs):
            linear_output += xi[i] * weights[i]

        linear_output += bias
        predicted_y = activationFunc(linear_output)
```

```
            currently_predicted[current_iter] = predicted_y
            if predicted_y != target[current_iter]:
                if useDynamicLR:
                    DynamicLearningRate = learning_rate * (1 / 10) **
                    ↪  (linear_output * LRrate)
                    print("Season ", current_season + 1, " & Iter ",
                    ↪  current_iter, " Learning Rate: ",
                    ↪  DynamicLearningRate)
                for i in range(0, n_inputs):
                    weights[i] = weights[i] + learning_rate *
                    ↪  (target[current_iter] - predicted_y) * xi[i]
                miss += 1

                bias = bias + learning_rate * (target[current_iter] -
                ↪  predicted_y)
            else:
                predicted[current_iter] = predicted_y
                hit += 1
            if n_inputs == 2 and ViewRealTimePlots:
                y = -(weights[0] / weights[1]) * x + \
                    (-bias / weights[1])
                ax.set(xlabel='X1',
                       ylabel='X2',
                       title="Season: " + str(current_season) + " Iter: " +
                       ↪  str(current_iter)
                       )
                ax.set_xlim([min(X_train[:, [0]]) - 2, max(X_train[:, [0]])
                ↪  + 2])
                ax.set_ylim([min(X_train[:, [1]]) - 2, max(X_train[:, [1]]
                ↪  + 2)])
                ax.scatter(X_train[:, [0]], X_train[:, [1]], marker='o',
                ↪  c=currently_predicted)
                ax.scatter(xi[0], xi[1], marker='o', c='r')
                line, = ax.plot(x, y)
                line.set_ydata(y)

                fig.canvas.draw()
                plt.pause(0.0005)
                fig.canvas.flush_events()
                ax.clear()

            if n_inputs == 3 and ViewRealTimePlots:
                z = (-weights[0] / weights[2]) * x + \
                    (-weights[1] / weights[2]) * y + \
                    (-bias / weights[2])
                plt.title("Season: " + str(current_season) + " Iter: " +
                ↪  str(current_iter))
                ax.set_xlim([min(X_train[:, [0]]) - 2, max(X_train[:, [0]])
                ↪  + 2])
                ax.set_ylim([min(X_train[:, [1]]) - 2, max(X_train[:, [1]]
                ↪  + 2)])
```

```
202             ax.set_zlim([min(X_train[:, [2]]) - 2, max(X_train[:, [2]]
                ↪  + 2)])
203             ax.scatter(X_train[:, [0]], X_train[:, [1]], X_train[:,
                ↪  [2]], marker='o', c=currently_predicted)
204             ax.scatter(xi[0], xi[1], xi[2],  marker='o', c='r')
205             surf = ax.plot_surface(x, y, z, cmap=cm.coolwarm)
206
207             fig.canvas.draw()
208             plt.pause(0.005)
209             fig.canvas.flush_events()
210             ax.clear()
211
212     accuracy = (hit - miss) / n_rows * 100
213     if not useDynamicLR:
214         print("Season ", current_season + 1, " Accuracy Rate: ",
            ↪  str(accuracy))
215     else:
216         print("Season ", current_season + 1, " Accuracy Rate: ",
            ↪  str(accuracy))
217     temp = accuracy
218     print("Epoch Weights: ", weights, "\n")
219
220     if current_season + 1 == seasons and miss != 0 and
        ↪  StopAtSetSeasons:
221         seasons += 1
222     if miss == 0:
223         break
224     current_season += 1
225
226 if n_inputs == 2:
227     y = -(weights[0] / weights[1]) * x + (-bias / weights[1])
228     if ViewRealTimePlots:
229         plt.clf()
230         plt.close()
231
232     fig = plt.figure()
233     ax = fig.add_subplot()
234     ax.set_xlim([min(X_train[:, [0]]) - 2, max(X_train[:, [0]]) + 2])
235     ax.set_ylim([min(X_train[:, [1]]) - 2, max(X_train[:, [1]] + 2)])
236
237     plt.title("Season: " + str(current_season) + " Iter: " +
        ↪  str(current_iter))
238     ax.set_xlim([min(X_train[:, [0]]) - 2, max(X_train[:, [0]]) + 2])
239     ax.set_ylim([min(X_train[:, [1]]) - 2, max(X_train[:, [1]] + 2)])
240     ax.scatter(X_train[:, [0]], X_train[:, [1]], marker='o',
        ↪  c=currently_predicted)
241     ax.plot(x, y, 'r')
242     plt.draw()
243
244 if n_inputs == 3:
```

```
    z = (-weights[0] / weights[2]) * x + (-weights[1] / weights[2]) * y + (-bias / weights[2])
    if ViewRealTimePlots:
        plt.clf()
        plt.close()
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.set_xlim([min(X_train[:, [0]]) - 2, max(X_train[:, [0]]) + 2])
    ax.set_ylim([min(X_train[:, [1]]) - 2, max(X_train[:, [1]] + 2)])

    ax.set(xlabel='X1',
           ylabel='X2',
           zlabel='X3',
           title="Season: " + str(current_season) + " Iter: " + str(current_iter)
           )
    ax.set_xlim([min(X_train[:, [0]]) - 2, max(X_train[:, [0]]) + 2])
    ax.set_ylim([min(X_train[:, [1]]) - 2, max(X_train[:, [1]] + 2)])
    ax.set_zlim([min(X_train[:, [2]]) - 2, max(X_train[:, [2]] + 2)])

    ax.scatter(X_train[:, [0]], X_train[:, [1]], X_train[:, [2]], marker='o', c=currently_predicted)
    surf = ax.plot_surface(x, y, z, cmap=cm.coolwarm)
    fig.canvas.draw()

final_target = ["" for x in range(n_rows)]
final_prediction = ["" for x in range(n_rows)]

# Conversion... To original binary outputs
for i, x in enumerate(target):
    if x == 1:
        final_target[i] = act_outputs[0]
    elif x == 0:
        final_target[i] = act_outputs[1]

for i, x in enumerate(currently_predicted):
    if x == 1:
        final_prediction[i] = act_outputs[0]
    elif x == 0:
        final_prediction[i] = act_outputs[1]

AllPredicted = True
countSuccess = 0
for i in range(0, n_rows):
    if target[i] != currently_predicted[i]:
        AllPredicted = False
    else:
        countSuccess += 1
    print("Last Predicted Values: ", final_prediction[i], "\tExpected Value: ", final_target[i])
```

```
291         # print("Last Predicted Values: ", predicted[i], "\tExpected Value:
            ↪  ", target[i])
292
293     if AllPredicted:
294         print("All Training Data Predicted at ", countSuccess / n_rows *
            ↪  100, '% Accuracy')
295     if not AllPredicted:
296         print("Not all training Data was Predicted!", countSuccess / n_rows
            ↪  * 100, '% Accurate')
297
298     if n_inputs == 2:
299         print('bias =\t', bias, "\nW1 =\t\t", weights[0], "\nW2 =\t\t",
            ↪  weights[1])
300         print("Linear Function Formula -> Linear_Output = W0*bias + W1*x1 +
            ↪  W2*x2\n", )
301     if n_inputs == 3:
302         print('W3(bias) =\t', bias, "\nW0 = \t\t", weights[0], "\nW1 =
            ↪  \t\t", weights[1], "\nW2 = \t\t",
303               weights[2])
304         print("Linear Function Formula -> Linear_Output = W0*bias + W1*x1 +
            ↪  W2*x2 + W3*x3\n")
```

U Advanced IoU & Average Precision calculator on the groundtruth information for the YOLOv4 model using DarkHelp [6] & OpenCV [7]

Programming Language: C++
Requirements: DarkHelp, OpenCV

```cpp
#include <opencv2/opencv.hpp>
#include <DarkHelp.hpp>
#include <iostream>

#include <iterator>
#include <vector>
#include <typeinfo>
#include <algorithm>
#include <string>
#include <map>
#include <fstream>

using namespace cv;

#define rWidth 1280
#define rHeight 720

// Groundtruth Bounding Box Colors
std::map<int, Scalar> BboxColors = {
    {0, (Scalar(0, 255, 0))},
    {1, Scalar(255, 255, 0)},
    {2, Scalar(0, 0, 0)}
};


float Round(double var)
{
    float var2 = (float)var;
    float value = (int)(var2 * 100 + .5);
    return (float)value / 100;
}

struct Groundtruth{
    int id;
    Point2d pG1;
    Point2d pG2;
    bool predicted = false;
```

```
38
39  };
40
41  struct returnVal{
42      double IoU, intersection, Union;
43  };
44
45  std::string btoStr(bool x){
46      if (x)
47          return "True";
48      else
49          return "False";
50  }
51
52  returnVal calculateIoU(double xP, double yP, double wP, double hP,
    ↪  double xG, double yG, double wG, double hG){
53      double inter_box_top_left_x = std::max(xP, xG);
54      double inter_box_top_left_y = std::max(yP, yG);
55
56      double inter_box_bottom_right_x = std::min(xP + wP, xG + wG);
57      double inter_box_bottom_right_y = std::min(yP + hP, yG + hG);
58
59      double inter_box_w = std::max(0.0, inter_box_bottom_right_x -
        ↪  inter_box_top_left_x);
60      double inter_box_h = std::max(0.0, inter_box_bottom_right_y -
        ↪  inter_box_top_left_y);
61
62
63      inter_box_w = std::max(0.0, inter_box_bottom_right_x -
        ↪  inter_box_top_left_x);
64      inter_box_h = std::max(0.0, inter_box_bottom_right_y -
        ↪  inter_box_top_left_y);
65
66
67      double intersection = inter_box_w * inter_box_h;
68      double Union = wG*hG + wP * hP - intersection;
69      double IoU = Union > 0 ? intersection / Union : 0.0;
70
71      return returnVal{IoU, intersection, Union};
72  }
73
74  class flagChecker{
75  private:
76      bool enableTiles = false;
77      bool useDuration = false;
78      bool drawBbox = false;
79      bool showConfidence = false;
80      double thresh = 0.25;
81      std::string fileName;
82      bool mAP = false;
83      bool IoU = false;
```

```
84      bool Stats = false;
85
86      std::vector<std::string> imageList;
87  public:
88  bool isImageList = false;
89      flagChecker(char* argv[]){
90          tokenChecker(argv);
91      }
92
93      void flagNames(std::string flag){
94          if(flag == "enableTiles")
95              this->enableTiles = true;
96          else if(flag == "useDuration")
97              this->useDuration = true;
98          else if(flag == "drawBbox")
99              this->drawBbox = true;
100         else if (flag == "showConfidence")
101             this->showConfidence = true;
102         else if (flag == "map")
103             this->mAP = true;
104         else if(flag == "iou")
105             this->IoU = true;
106         else if(flag=="stats")
107             this->Stats = true;
108     }
109
110     std::string convertToString(char* message[], int si, int index){
111         int i;
112         std::string str = "\0";
113         for (i = 1; i < si; i++) {
114             str += message[index][i];
115         }
116         return str;
117     }
118     void tokenChecker(char* message[]){
119         int si = 0;
120         while(message[++si] != NULL);
121         for(int i = 1 ; i < si; ++i){
122             if(message[i][0] == '-'){
123                 int sizeMessage = strlen(message[i]);
124                 std::string str = this->convertToString(message,
                    ↪  sizeMessage, i);
125                 if(str == "thresh"){
126                     this->thresh = std::stod(message[i+1]);
127                     ++i;
128                 }else if(str == "img"){
129                     this->fileName = message[i+1];
130                     ++i;
131                 }else if(str == "txtInput"){
132                     std::ifstream file;
133                     file.open(message[i+1], std::ios::in);
```

```
                    if(!file.is_open()){
                        std::cout << "Warning: text Input File Path Not
                        ↪  Found..." << std::endl;
                        exit(0);
                    }
                    std::string line;
                    while(std::getline(file, line)){
                        this->imageList.push_back(line);
                        this->isImageList = true;
                    }
                }
                else
                    flagNames(str);
            }
        }
    }

    bool getStats(){return this->Stats;}
    bool getIoU(){return this->IoU;}
    bool getmAP(){return this->mAP;}
    std::vector<std::string> getImageList(){return this->imageList;}
    std::string getfileName(){return this->fileName;}
    bool getEnableTiles(){return this->enableTiles;}
    bool getuseDuration(){return this->useDuration;}
    bool getdrawBbox(){return this->drawBbox;}
    bool getshowConfidence(){return this->showConfidence;}
    double getThresh(){return this->thresh;}
};

int main(int argc, char* argv[]){

    flagChecker flagchecker(argv);

    DarkHelp::Config cfg("data/YOLOv4/birdsProject.cfg",
    ↪  "data/YOLOv4/birdsProject_best.weights",
    ↪  "data/YOLOv4/birdsProject.names");
  cfg.enable_tiles          = flagchecker.getEnableTiles();
  cfg.combine_tile_predictions    = true;
  cfg.annotation_auto_hide_labels    = false;
  cfg.annotation_include_duration    = false;
  cfg.annotation_include_timestamp   = false;
    cfg.threshold                       = flagchecker.getThresh();
    cfg.names_include_percentage        =
    ↪  flagchecker.getshowConfidence();
    int numImage = 0;
    double mAP = 0;
    double IoU = 0;
    int totalTP = 0, totalFP = 0, totalFN = 0;

    std::vector<std::string> imageList = flagchecker.getImageList();
    if(!flagchecker.isImageList)
```

```
181         imageList.push_back(flagchecker.getfileName());
182 
183     for(auto & image : imageList){
184         std::string filepath = image;
185         std::string fname, extension;
186         fname = filepath.substr(0, filepath.find("."));
187 
188         size_t pos = 0;
189         std::string token;
190         while ((pos = filepath.find(".")) != std::string::npos) {
191             token = filepath.substr(0, pos);
192             fname = token;
193             filepath.erase(0, pos + 1);
194         }
195         extension = "." + filepath;
196         filepath = image;
197 
198         std::ifstream file;
199         if(flagchecker.getdrawBbox()){
200             file.open(fname+".txt", std::ios::in);
201             if(!file.is_open()){
202                 std::cout << "ERROR: Image Text File path not found" <<
                    ↪  std::endl;
203                 return -1;
204             }
205         }
206         ++numImage;
207 
208         DarkHelp::NN nn(cfg);
209 
210         // you can further modify the configuration even after the
            ↪  neural network has been created
211         nn.config.annotation_line_thickness  = 1;
212         nn.config.combine_tile_predictions  = true;
213 
214         Mat img = imread(filepath, IMREAD_COLOR);
215 
216         int imWidth = img.size[0];
217         int imHeight = img.size[1];
218         std::cout << "\nImage: " << image <<std::endl;
219         std::cout << "Width: " << imWidth << "\nHeight: " << imHeight
            ↪  << "\n";
220         const auto results = nn.predict(img);
221 
222         img = nn.annotate();
223 
224         if(flagchecker.getdrawBbox()){
225             std::string line;
226             int numClasses = 0;
227             int id = 0;
228             Groundtruth Gtruth[1000];
```

```
            double xPoint_center=0, yPoint_center=0, wVal =0, hVal=0;
            while(std::getline(file, line)){
                std::stringstream ss(line);
                int wordCount=0;
                while(!ss.eof()){
                    std::string word;
                    getline(ss, word, ' ');
                    ++wordCount;
                    switch(wordCount){
                        case 1:
                            id = std::stoi(word);
                            break;
                        case 2:
                            xPoint_center = std::stod(word);
                            break;
                        case 3:
                            yPoint_center = std::stod(word);
                            break;
                        case 4:
                            wVal = std::stod(word);
                            break;
                        case 5:
                            hVal = std::stod(word);
                            break;
                    }
                }
                Point2d p1(xPoint_center, yPoint_center);
                Point2d p2(wVal, hVal);
                Gtruth[numClasses].id = id;
                Gtruth[numClasses].pG1 = Point2d(xPoint_center,
                ↪  yPoint_center);
                Gtruth[numClasses].pG2 = Point2d(wVal, hVal);
                ++numClasses;
            }
            // std::cout << "Num Classes: " << numClasses << std::endl;
            double totalIoUcount = 0, totalIoUsum = 0, TotalIoU = 0;
            double x, y, w, h;
            double AveragePrecision;
            int TP = 0, FP = 0, FN = 0;
            std::vector<Point2d> occupiedPositions;
            for(const auto & det : results){
                // Get Coordinates for every prediction
                bool containsGroundtruth = false;
                x = det.rect.x;
                y = det.rect.y;
                w = det.rect.width;
                h = det.rect.height;
                int predID = det.best_class;

                for(int i=0; i < numClasses; i++){
                    double originalxG = Gtruth[i].pG1.x;
```

```
                    double originalyG = Gtruth[i].pG1.y;
                    double originalwG = Gtruth[i].pG2.x;
                    double originalhG = Gtruth[i].pG2.y;

                    // Converting Coordinates to fit Canvas
                    double xG = originalxG*imWidth -
                    ↪  (originalwG*imWidth)/2;
                    double yG = originalyG*imHeight -
                    ↪  (originalhG*imHeight)/2;
                    double wG = originalwG*imWidth;
                    double hG = originalhG*imHeight;

                    // Calculating IoU and returning a structure of
                    ↪  three values (IoU, intersection, Union)
                    ↪  (double)
                    returnVal stats = calculateIoU(x, y, w, h, xG, yG,
                    ↪  wG, hG);
                    // Creating points for Groundtruth Bounding Box
                    Point2d p1(xG, yG);
                    Point2d p2(wG+xG, hG+yG);

                    // Creating Location and color for Text message
                    Point2d pos(xG, hG+yG+12);

                    // Text Message Color
                    Scalar color(0, 0, 0);

                    // IoU threshold if message and BBox is drawn
                    if(stats.IoU > 0.5){
                        containsGroundtruth = true;
                        Gtruth[i].predicted = true;
                        if(Gtruth[i].id == predID)
                            ++TP;
                        else
                            ++FP;

                        stats.IoU *= 100;
                        stats.IoU = floor(stats.IoU*100.0)/100.0;
                        // std::cout << stats.IoU << std::endl;
                        std::string str = std::to_string(stats.IoU);
                        str.erase(str.find_last_not_of('0') + 1,
                        ↪  std::string::npos);
                        str.erase(str.find_last_not_of('.') + 1,
                        ↪  std::string::npos);

                        // Drawing Bounding box to image at with values
                        ↪  p1 and p2 (Point2d)
                        Point2d temp1, temp2, temp3, temp4;
                        temp1 = pos;
                        temp2 = pos;
                        temp3 = pos;
```

```
322                         temp4 = pos;
323                         // Checks if there is an occupied position from
                            ↪ the vector, if there is increase Y axis
324                         bool foundPosition = false;
325                         bool positionOccupied = false;
326                         while (!foundPosition) {
327                             for (const auto& occupiedPos :
                                ↪ occupiedPositions) {
328                                 if (std::abs(pos.x - occupiedPos.x) <
                                    ↪ 1e-5 && std::abs(pos.y -
                                    ↪ occupiedPos.y) < 1e-5) {
329                                     positionOccupied = true;
330                                     break;
331                                 }
332                             }
333                             if (!positionOccupied)
334                                 foundPosition = true;
335                             else
336                                 break;
337                         }
338                         if(positionOccupied)
339                             break;
340                         // Drawing Bounding box to image at with values
                            ↪ p1 and p2 (Point2d)
341                         rectangle(img, p1, p2,
                            ↪ BboxColors[Gtruth[i].id], 1.5);
342                         ++totalIoUcount;
343                         totalIoUsum += stats.IoU;
344                         // Putting Text message to image at pos
                            ↪ coordinates
345                         putText(img, str + "%", pos,
                            ↪ FONT_HERSHEY_TRIPLEX, 0.4, color, 1, 16);
346                         for(int j = 0; j < 10; j++){
347                             // Create Occupied Position radius Points
                                ↪ and push_back to occupiedPositions
                                ↪ Vector
348                             temp1.y += j;
349                             temp2.y -= j;
350                             temp3.x += j;
351                             temp4.x -= j;
352                             occupiedPositions.push_back(temp1);
353                             occupiedPositions.push_back(temp2);
354                             occupiedPositions.push_back(temp3);
355                             occupiedPositions.push_back(temp4);
356                         }
357                     }
358                 }
359                 if(!containsGroundtruth)
360                     ++FN;
361             }
362             for(int i = 0; i < numClasses; ++i){
```

```
                if(!Gtruth[i].predicted)
                    ++FP;
            }
            if(totalIoUcount != 0)
                TotalIoU = totalIoUsum/totalIoUcount;
            else
                TotalIoU = 0;
            AveragePrecision = double(TP)/double(TP+FP)*100;

            IoU += TotalIoU;
            mAP += AveragePrecision;
            std::cout << "Total IoU: " << TotalIoU << std::endl <<
            ↪  "Average Precision: " << AveragePrecision << std::endl
            << "TP: " << TP << std::endl
            << "FP: " << FP << std::endl
            << "FN: " << FN << std::endl;
            totalTP += TP;
            totalFP += FP;
            totalFN += FN;
        }

        Mat Resized;
        resize(img, Resized, Size(1024, 1024), INTER_LINEAR);
        if(!flagchecker.isImageList){
            imshow("MAIN",  Resized);
            waitKey(0);
        }
        if(flagchecker.isImageList)
            imwrite("output/Image_" + std::to_string(numImage) +
            ↪  "result.jpg", img);
        else
            imwrite("result.jpg", img);

        destroyAllWindows();
    }
    mAP /= numImage;
    IoU /= numImage;
    if(flagchecker.getmAP())
        std::cout << "Mean Average Precision: " << mAP << "%" <<
        ↪  std::endl;
    if(flagchecker.getIoU())
        std::cout << "Mean Intersection over Union: " << IoU << "%" <<
        ↪  std::endl;
    if (flagchecker.getStats()){
        std::cout << totalTP << "TP  - " << totalFP << "FP - " <<
        ↪  totalFN << "FN" << std::endl;
    }
    return 0;
}
```