ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

MSc THESIS

Halit ERİŞ

IMPLEMENTATION OF TARGET TRACKING METHODS ON IMAGES TAKEN FROM UAV (UNMANNED AERIAL VEHICLES)

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ADANA-2020

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

IMPLEMENTATION OF TARGET TRACKING METHODS ON IMAGES TAKEN FROM UAV (UNMANNED AERIAL VEHICLES)

Halit ERİŞ

MSc THESIS

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

We certify that the thesis titled above was reviewed and approved for the award of degree of the Master of Science by the board of jury on 07/01/2020.

……	……..	…….
Prof. Dr. Ulus ÇEVİK	Prof. Dr. Mustafa GÖK	Asst. Prof. Dr. Gökay DİŞKEN
SUPERVISOR	MEMBER	MEMBER

This MSc Thesis is written at the Department of Electrical and Electronics Engineering of Institute of Natural And Applied Sciences of Çukurova University.

Registration Number:

Prof. Dr. Mustafa GÖK
Director
Institute of Natural and Applied Sciences

ABSTRACT

MSc THESIS

IMPLEMENTATION OF TARGET TRACKING METHODS ON IMAGES TAKEN FROM UAV (UNMANNED AERIAL VEHICLES)

Halit ERİŞ

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

Supervisor : Prof. Dr. Ulus ÇEVİK
Year: 2020, Pages: 75
Jury : Prof. Dr. Ulus ÇEVİK
: Prof. Dr. Mustafa GÖK
: Asst. Prof. Dr. Gökay DİŞKEN

In this thesis, a software has been developed by applying image processing, and deep learning methods to the images obtained with the camera on an unmanned aerial vehicle (UAV). Basic traffic objects are detected, placed in boundary boxes, labeled, and their confidence ratings are calculated as a percentage to provide an improved vision system for UAV users. The vision system has an important role in the professional field of UAVs to accomplish their tasks, and the target objects can be detected within the vision system by using Artificial Neural Networks (ANN).

In this study, Faster-R CNN and YOLOv2 models were used, and compared for object detection. The Microsoft-COCO pre-trained data set is used for the training, and a new data set which is prepared with the images taken from UAV. Classes are person, car, and motorcycle. The objects were detected successfully. The methods and results were compared according to the performance and accuracy ratios both on the pre-trained data and on the data set prepared on the images taken from UAV.

Keywords: Deep Learning, Object Detection, Image Processing, Unmanned Aerial Vehicles

ÖZ

YÜKSEK LİSANS TEZİ

İNSANSIZ HAVA ARAÇLARINDAN (İHA) ALINAN GÖRÜNTÜLER ÜZERİNDE HEDEF İZLEME YÖNTEMLERİNİN UYGULANMASI

Halit ERİŞ

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK - ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

Danışman : Prof. Dr. Ulus ÇEVİK
Yıl: 2020, Sayfa: 75
Jüri : Prof. Dr. Ulus ÇEVİK
: Prof. Dr. Mustafa GÖK
: Dr. Öğr. Üyesi Gökay DİŞKEN

Bu tezde, insansız hava aracı (İHA) üzerinde bulunan kamera ile elde edilen görüntülere görüntü işleme ve derin öğrenme metotları uygulanarak bir yazılım geliştirilmiş, temel trafik nesnelerinin saptanarak sınır kutuları içerisine yerleştirilerek, etiketlenmesi ve güven puanlamaları yüzde olarak hesaplanarak İHA kullanıcıları için gelişmiş bir görüş sistemi kazandırılmıştır.

Profesyonel alanda kullanılan İHA'ların üstlendikleri görevleri tamamlamaları için görüş sistemi önemli bir yere sahiptir ve Yapay Sinir Ağları (YSA) kullanılarak görüş sistemi içerisinde hedeflenen nesneler saptanabilir. Bu çalışmada nesnelerin saptanması işlemi için Faster-R CNN ve YOLOv2 modelleri kullanılmış ve karşılaştırılmıştır. Öğrenmenin gerçekleşmesi için Microsoft-COCO veri setine ait önceden eğitimli verileri ve özgün şekilde hazırlamış olduğumuz veri seti kullanılmıştır. İnsan, araba ve motorsiklet olmak üzere 3 adet sınıf belirlenmiş ve İHA ile elde edilen görüntüler üzerinde saptamaları başarılı şekilde gerçekleştirilmiştir. Metotlar ve sonuçlar performans ve isabetlilik oranlarına göre hem önceden eğitilmiş veriler üzerinde, hemde özgün şekilde hazırlanmış veri seti ile eğitilerek karşılaştırılmıştır.

Anahtar Kelimeler: Derin Öğrenme, Nesne Algılama, Görüntü İşleme, İnsansız Hava Araçları

EXTENDED ABSTRACT

In general, unmanned aerial vehicles, in short UAVs, are aircrafts that are intended to operate with no pilot on board. They have various types according to their flight method, their weight, and capabilities. Unmanned aerial vehicles can be remote-controlled by a pilot or they can be autonomously controlled according to their tasks. Even though unmanned aerial vehicles were invented at the beginning of the nineteenth century, unmanned aerial vehicles' professional usage has become wider in the last twenty years. Today, there are many governments, which have been exporters, operators, manufacturers, and developers of unmanned aerial vehicles. Fourteen of the countries in the world are unmanned aerial vehicle exporters, twenty of them are unmanned aerial vehicle operators, twenty of them unmanned aerial vehicle manufacturers, and twenty-four of them are unmanned aerial vehicle developers.

Special sensors such as LIDARs (Light Detection and Ranging sensor or also known as Laser Imaging Detection and Ranging sensors), proximity sensors, thermal sensors, and cameras such as optical and thermal cameras make unmanned aerial vehicles to interact with their environment. These sensors give unmanned aerial vehicles the capability of executing a more complex mission related to target detection, and target recognition. We can give Common examples for such complex applications of unmanned aerial vehicles as Security applications, inspecting terrain, pipelines, power lines, buildings, etc., Crisis and disaster assisting, search and rescue missions, environment monitoring, aerial mapping, meteorological purposes, research applications for universities, firefighting using a thermal camera, wildlife surveys, etc. In order to operate such complex tasks dynamically, UAV's vision system should operate fast and accurate similar to the human visual system. The human visual system analyzes the environment and implements object detection and recognition simultaneously on objects on sight with real-time speed. If we consider the movement of UAVs, UAV's vision and

detection system are required to be fast, and accurate. The system must be adaptive to the main role classes of its task. This requires data about the classes, learning, and detection. Object detection for the targets can be implemented by using image processing, and deep learning techniques.

The ability of computer systems to learn from experience, having an understanding of the connections between concepts and hierarchy, and giving a solution for such relations between concepts is called by many names. Today, it is commonly called Deep Learning. Deep learning is one of the most used techniques and highly profitable. Many top technology companies such as Microsoft, Google, Facebook, IBM, Apple, Baidu, Netflix, YouTube, NVIDIA, and many others use deep learning during their operations so that they can predict current habits of the operation, and customers.

In this thesis, an investigation, and implementation of modern object detection models on the target classes we chose to improve the UAV vision and decision system. It is aimed to correctly classify the test images, draw boundary boxes around the objects, and label three fundamental classes for traffic in actual traffic scene test images. The target object classes are person, car, and motorbike. To capture the images a quadcopter was used. A 12 megapixels onboard camera is located on the front edge of the UAV which can capture 1080p with 96 fps, and 4k images with 30 fps. Three different GPUs were used to compare the performance for all of the methods on each GPU: NVIDIA GeForce 750M®, NVIDIA Jetson TX2 Developer Kit®, and NVIDIA GTX 1060®. All the GPUs were chosen from NVIDIA due to its compatibility with used GPU libraries: CUDA, and CuDnn. Two models were chosen according to use in the application which requires fast speed, and high accuracy. The first one is the YOLOv2 model for high-speed process, and the second one is Faster R-CNN for high accuracy. Also, we created a new dataset for our operations. This dataset includes both training, and test images from traffic scenes. During the implementation of the methods, models were tested under two objectives. Firstly, models were implemented with a pre-trained weight trained on a benchmark dataset: Microsoft COCO: Common Objects in Context,

and tested on the test data we gathered with a quadcopter. Secondly, these models were implemented using the training data we gathered and tested on our test data. During the implementation, 3 different Graphics Processing Units (GPU) were used, and their performances were compared. When the last 5 years of the literature is taken into consideration, there are several object detection methods, successfully developed on well-known benchmark datasets with powerful GPUs. Most of the object detection methods are using feature extraction on images and proposing regions of interest. This regional proposal gives a higher accuracy but it is a slow process. On the other hand, methods like YOLO using a single or several CNN networks to make detection without regional proposals. This is why YOLO trades accuracy for speed.

When the general process of the thesis is summarized, a pre-process that consists of image enhancement, and image resizing was used. If the test image is under bad condition, which represents the image captured during foggy, rainy, or cloudy weather, Contrast Limited Adaptive Histogram Equalization (C.L.A.H.E) was used. Then the test images were resized to 608x608 resolution for the YOLOv2 method, and 640x480 resolution for the Faster R-CNN method. Then the images were installed to the network architecture. We separated the objectives into 2 sub-objectives in order to make a comparison. In the first objective, after the pre-process YOLOv2 model is implemented with pre-trained weights that trained on Microsoft COCO. Then the YOLOv2 model is implemented with training data trained on the new data set. Both results were tested on UAV test images with 3 different GPUs. In objective 2, The Faster R-CNN model is implemented with pre-trained weights that trained on Microsoft COCO. Then, the Faster R-CNN model is implemented with training data trained on the new data set. Both results were tested on UAV test images with 3 different GPUs.

Results were compared according to each model, and each objective with a test on every GPU model to observe the performance of the models for specific GPUs. The test images were captured from 5, 10, 15, 25, 35, and 45 meters altitudes. The minimum threshold was %20 for the YOLOv2 model, and it was %80 for Faster R-CNN as default. The results showed that YOLOv2 was a fast

object detection model that was applicable to real-time operations if it had been trained on a very large dataset with target object class instances. The new dataset training set was not enough to increase the accuracy, and precision. The state-of-the-art datasets such as Microsoft COCO provided better results with YOLOv2 high speed. YOLOv2 with Microsoft COCO dataset had %48,74 confidence, the new dataset decreased the confidence to under %20 threshold. Faster R-CNN was slow but a confident model with much higher accuracy, and precision with pre-trained weights from Microsoft COCO. In addition, we could increase the confidence of the model by using the new dataset which is more adaptive to the UAV's perspective with the scenes we captured. The total confidence of the model with Microsoft COCO dataset was %87,71 on our test data, we could increase the confidence to %98,81 with training data collected from actual scenes. The performance through 3 GPUs was always better with more powerful GPUs such as NVIDIA GTX 1060. Besides, the GPUs used %100 memory, which means if it is aimed to process such models, a higher GPU memory is a need. In our case, we used 6 gigabytes of GPU memory during the process. The localization error which occurred as detecting an object on a place where there was no object instance. It was decreased by increasing the training data of the object class in similar scenes. Faster R-CNN was more accurate, but slower algorithm than the YOLOv2. YOLOv2 required to be trained with a large dataset or the model suffers under-sampling when it is trained on the new dataset. Faster R-CNN received a better result when it was trained on the new dataset. The precision and accuracy were improved. YOLOv2 operated with 24 fps speed, the model is applicable for real-time operations. The Faster R-CNN should be used in cases in which UAV has a mission to detect the target objects on a certain scene, and the accuracy is critical. Instead of using the algorithm on a video stream, the method should be used on well-caught image frames. As a future work, the new dataset can be enriched by adding more images, and optimization algorithms such as particle swarm algorithm to utilize the parameters, and to reach higher accuracy, and precision rates.

ACKNOWLEDGMENTS

I would like to express my deep and sincere gratitude to, my supervisor, Prof. Dr. Ulus ÇEVİK for his guidance, supervision, encouragement, and for leading my interests to the research and scholarship. It has been a great honor to have Prof. Dr. Ulus ÇEVİK as a supervisor.

I also thank my parents for their precious support, and my brother Mehmet Alp ERİŞ for his companionship during my study.

I also want to thank my colleague Kemal AYGÜL, and my friend Devrim KAYALI, for their valuable discussions and technical help.

Finally, I would like to express my deepest gratitude to my fiancée, Elif BİRDİR for her patience, encouragement and continuous moral support.

TABLE OF CONTENT **PAGE**

LIST OF TABLES **PAGE**

LIST OF FIGURES **PAGE**

LIST OF SYMBOLS

β	:	Noise function
σ	:	Variance
θ	:	The angle between u and the gradient
$*$	:	Convolution operation
a	:	Average value
E	:	Experience
e	:	Division function
f	:	Function
g	:	Output function
h	:	Summation function
Im	:	Image function
i	:	Gray level intensity
k	:	Intensity number
K	:	Image number
l	:	Subtraction function
L	:	Maximum pixel ranges
m	:	Pixel rows
n	:	Pixel columns
P	:	Performance
p	:	Pixel number
s	:	Feature map function
T	:	Task
t	:	Pixels in operation
$t(x)$	:	Multiplication function
u	:	Uphill and downhill direction
x	:	Pixel pair of x

y	:	Pixel pair of y
w	:	Weight function
$w(x, y)$	:	Kernel filter

ABBREVIATIONS

AI	:	Artificial Intelligent
ANN	:	Artificial Neural Network
API	:	Application Programming Interface
CFG	:	Configuration File
CLAHE	:	Contrast Limited Adaptive Histogram Equalization
CNN	:	Convolutional Neural Network
COCO	:	Common Objects in Context
CPU	:	Central Processing Unit
D	:	Dimension
EQ	:	Equation
FIG	:	Figure
FPS	:	Frame Per Seconds
GPU	:	Graphics Processing Unit
IMGDIR	:	Image Directory
IR	:	Infrared
LIDAR	:	Light Detection and Ranging
MRI	:	Magnetic Resonance Imaging
MT	:	Meter
MTCR	:	Missile Technology Control Regime
MTE	:	Mean Test Error
RGB	:	Red,Green,Blue
R-CNN	:	Regional Proposal Based Convolutional Neural Network
ROI	:	Region of Interest
TF	:	Tensorflow
UAV	:	Unmanned Aerial Vehicle
UI	:	User Interface
V	:	Version

YOLO	:	You Only Look Once
XML	:	Extensible Markup Language

1. INTRODUCTION

1.1. Overview

Unmanned Aerial Vehicles (UAV) are aircraft that are intended to operate with no pilot on board. UAVs currently use various sensors such as Proximity sensors, Light Detection and Ranging (LiDAR) sensors Infrared (IR) sensors, Ultrasonic sensors, and Smoke sensors, etc. In addition, they may have image capturers such as optical cameras, and thermal cameras on board. The use of sensors and image capturers gives UAV the ability to interact with their surroundings. This ability widens the indoor and outdoor application areas of UAVs. There are various applications operated by UAVs such as aerial photography, film-making, agricultural and wildlife surveys, illegal hunting detection, search and rescue missions, monitoring power and pipe-lines, medical and military delivery to inaccessible area points, reconnaissance, attacking, smuggler detection, traffic rule violation detection, and civil applications as a hobby. To complete such complex tasks, the vision system of the UAV must be optimal for the operating environment and the main role classes of the tasks must be learned and detected. The vision system can be improved by detecting the target objects using image processing and Artificial Neural Network (ANN) methods with a powerful GPU and pre-trained weights.

In this study, object detection was implemented on images taken from UAV for 3 fundamental classes for traffic: person, car, and motorbike. Image processing, Artificial Neural Network, and Deep Learning techniques and their modern models were investigated for pre-trained and self-made image datasets. The self-made dataset consists of 15 different scenes and 800 unique images. Moreover, the performance and the accuracy of different Deep Learning methods and different datasets were compared.

1.1.1. Purpose of the Thesis

In this section, an overview of the purpose and contributions of the thesis are presented. The main purpose of this thesis is to investigate and implement modern object detection models for purposes that require high accuracy, and for which requires high-speed process on images taken from UAV's perspective to improve the UAV vision systems.

Traditional object detection algorithms have a trade-off between high accuracy and processing speed. The application and the scene determine the trade-off. Although generating region-based proposals, implementing feature extraction, and then implementing a classification algorithm process has a deeper approach to object detection and a higher accuracy, the process is slow when the UAV's aerial movement is considered. On the other hand, algorithms which implement classification and localization by a single convolutional neural network reaches higher speeds but sacrifices some accuracy.

Transfer learning method (Goodfellow, et al., 2016) is time-saving when it is compared to creating your own dataset, resizing the images, labeling every frame for every class object, creating annotations and training it according to model. It requires a powerful Graphics Processing Unit (GPU) with high memory capacity and plenty of time. Instead, pre-trained models that are trained on a large benchmark dataset to solve similar problems can solve the task. Otherwise, creating an original dataset and following the steps is costly but efficient for the purpose of the task.

In this study, aerial images from UAV's perspective from many different scenes were captured by a quadcopter. Then the images were processed on the land base with three GPU simultaneously: Nvidia GT750M®, Nvidia Jetson TX2 Developer Kit®, and Nvidia GTX 1060®. It is aimed to correctly classify and label 3 fundamental classes for traffic in the scene: person, car, and motorbike. The Faster-R CNN: Region-based Convolutional Neural Network Inception (Shaoqing, et al.) model was used for higher accuracy, "You Only Look Once v2" (YOLOv2)

(Redmon, et al., 2018) was used for higher speed in frame per second (fps). For images under severe conditions, histogram equalization (Szeliski, 2010) was used. Input images were letterboxed into square-shaped images. Both the transfer learning and a new original dataset were used, results were investigated and compared according to three different GPUs. The transfer learning was used by using Microsoft COCO: Common Object in Context pre-trained weights for both models (Lin, et al., 2015). A new original dataset was created with images taken from the UAV. The dataset had 800 images, which had the 3 classes simultaneously and together. Images were taken in high resolution, then cropped into small pieces with desired scenes. Contrast noises were added in order to enrich the dataset and simulate weather effects. Also, every image was labeled, using boundary boxes from the upper left corner of the class object to the bottom right corner. Then annotations were written for every frame, which indicated the classes and their boundary coordinates as pixels in the given frame. Both methods could detect the target objects in the images. According to the results obtained, while Faster R-CNN operated too slow for real-time operations it produced a high accuracy. On the other hand, YOLOv2 operated faster, reached almost a real-time operation speed with Nvidia GTX 1060 GPU, but had low accuracy. When the altitude was higher the accuracy was lower. The results show that both models and learning methods can be implemented for images taken from UAV if a powerful GPU is used. Faster R-CNN was more reliable for the cases when a high accuracy is needed, and YOLOv2 for cases for real-time operations.

1.2. Unmanned Aerial Vehicles (UAV)

1.2.1. UAV Market

Nowadays, exporting, operating, manufacturing, developing of the UAVs are reported and being resstricted their sales on market by Missile Technology Control Regime (MTCR) because of the potential role of UAVs. In table 1., there are member countries and their summary of the UAV market is shown.

Table 1.1. (Missile Technology Control Regime) UAV exporter, operator, manufacturer, and developer table of MTCR members.

MTCR member	**UAV exporter**	**UAV operator**	**UAV manufacturer**	**UAV developer**
Argentina	No	Yes	Yes	Yes
Australia	Yes	Yes	Yes	Yes
Austria	Yes	No	Yes	Yes
Belgium	No	Yes	Yes	Yes
Brazil	No	No	No	No
Canada	Yes	No	Yes	Yes
Czech Republic	No	Yes	Yes	Yes
Denmark	No	Yes	No	No
Finland	No	Yes	No	No
France	Yes	Yes	Yes	Yes
Germany	Yes	Yes	Yes	Yes
Greece	No	No	No	Yes
Hungary	No	No	No	Yes
Iceland	No	No	No	No
Ireland	No	No	No	No
Italy	Yes	Yes	Yes	Yes
Japan	Yes	Yes	Yes	Yes
Luxembourg	No	No	No	No
The Netherlands	No	Yes	No	No
New Zealand	No	No	No	No
Norway	No	No	No	Yes
Poland	No	No	No	No
Portugal	No	No	No	Yes
Russia	Yes	Yes	Yes	Yes
South Africa	Yes	Yes	Yes	Yes
South Korea	No	Yes	Yes	Yes
Spain	No	No	Yes	Yes
Sweden	No	Yes	Yes	Yes
Switzerland	Yes	Yes	Yes	Yes
Turkey	Yes	Yes	Yes	Yes
Ukraine	Yes	Yes	Yes	Yes
United Kingdom	Yes	Yes	Yes	Yes
United States	Yes	Yes	Yes	Yes

1.2.2. Application Areas of UAVs

The next-generation UAVs are capable of executing more complex missions that are related to target detection, recognition, network and communication, aerial delivery by using cameras, GPUs, and many other sensors. Some of the civil applications which serve the civil market can be given as follows:

- Security applications.
- Inspecting terrain, pipelines, power lines, buildings, etc.
- Crisis and disaster assisting, search and rescue.
- Environment monitoring.
- Aerial mapping.
- Meteorology.
- Research applications for universities.
- Firefighting using a thermal camera.
- Wildlife surveys.

1.3. 2D Image Processing

Images can be described as two-dimensional functions $f(x, y)$. (x, y) are the coordinates of the function of an image which has m rows and n columns and function's amplitude of any pair of coordinates (x, y) is called the intensity or gray level i of any image at that point. Images are composed of a finite number of elements which each of them has a specific location and intensity. These elements are called with many names such as image elements, pels, picture elements, and pixels. Pixel is the most commonly used term in image processing. Pixels can be shown with their coordinates and intensity. An image with 8-bit color space and its pixel coordinates and intensities are shown in figure 1.1. This 9x9 image consists of only black and white-colored pixels. We expect that the image color intensity to be in a range from 0 to 255. 0 represents black color intensity and 255 represents white.

a)

(0,0,0)	(1,0,255)	(2,0,255)	(3,0,0)	(4,0,0)	(5,0,0)	(6,0,255)	(7,0,255)	(8,0,0)
(0,1,255)	(1,1,255)	(2,1,255)	(3,1,0)	(4,1,255)	(5,1,0)	(6,1,255)	(7,1,255)	(8,1,255)
(0,2,255)	(1,2,255)	(2,2,255)	(3,2,0)	(4,2,0)	(5,2,0)	(6,2,255)	(7,2,255)	(8,2,255)
(0,3,0)	(1,3,255)	(2,3,255)	(3,3,0)	(4,3,0)	(5,3,0)	(6,3,255)	(7,3,255)	(8,3,0)
(0,4,0)	(1,4,0)	(2,4,0)	(3,4,0)	(4,4,255)	(5,4,0)	(6,4,0)	(7,4,0)	(8,4,0)
(0,5,0)	(1,5,255)	(2,5,255)	(3,5,0)	(4,5,0)	(5,5,0)	(6,5,255)	(7,5,255)	(8,5,0)
(0,6,255)	(1,6,255)	(2,6,255)	(3,6,0)	(4,6,0)	(5,6,0)	(6,6,255)	(7,6,255)	(8,6,255)
(0,7,255)	(1,7,255)	(2,7,255)	(3,7,0)	(4,7,255)	(5,7,0)	(6,7,255)	(7,7,255)	(8,7,255)
(0,8,0)	(1,8,255)	(2,8,255)	(3,8,0)	(4,8,0)	(5,8,0)	(6,8,255)	(7,8,255)	(8,8,0)

(b)

Figure 1.1. (a) A 8-bit image. (b) Coordinates and intensities of each pixel.

Image processing is strongly related to computer graphics and computer vision (Kuruvilla, et al., 2016) (Basavaprasad, et al., 2014). There are several goals of image processing:

- Monitoring any object that is not visible by hallucination.
- Increasing the quality of an image by image restoration and sharpening.
- Searching for the image of any interest by image repossession.
- Measuring any range of object coordinated in an image by pattern measurement.
- Differentiating any objects in an image by image acknowledgment.

1.3.1. Digital Processing

Image processing operations are implemented in order to achieve the goals shown above. However, before using the operations, the images are needed to be in a suitable format. Sampling and quantization are used for converting to digital form, called digitization. Sampling is simply digitizing the coordinate values of an image, and Quantization is digitizing the amplitude values. To use sampling on a function, samples were equally spaced in 2-dimension lines (2D). The spatial location of each sample is indicated by a vertical point mark in the lower part of the figure. Image sampling is shown in figure 1.2, for an image that includes a rainbow, and an apartment scene from 512x512 resolution to 8x8.

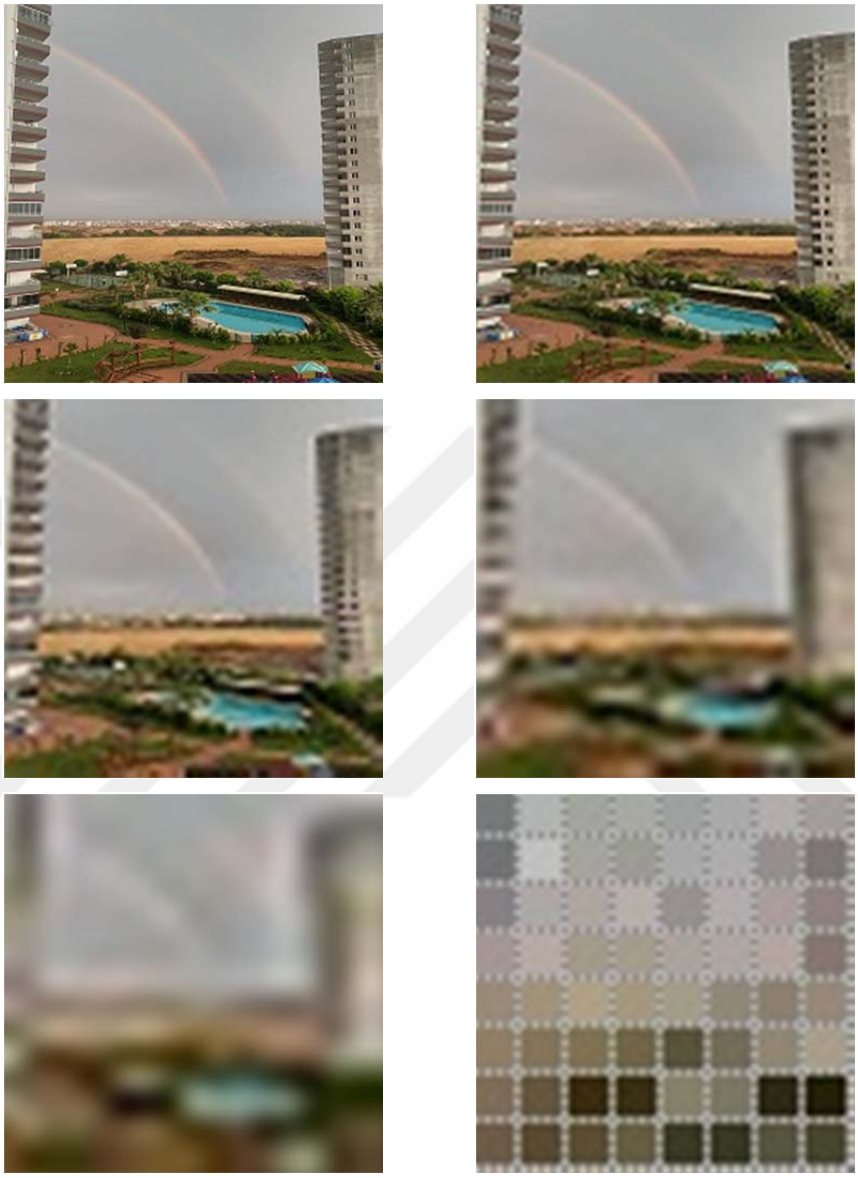


Figure 1.2. Using sampling of an image from 512x512 resolution to 8x8.

In order to use an image as a digital function, intensity values for the image must be also converted into discrete quantities. This is called quantization. Color quantization of the image, which is also shown in figure 1.2., is converted from 512-bits to 2-bit and shown step by step in figure 1.3.

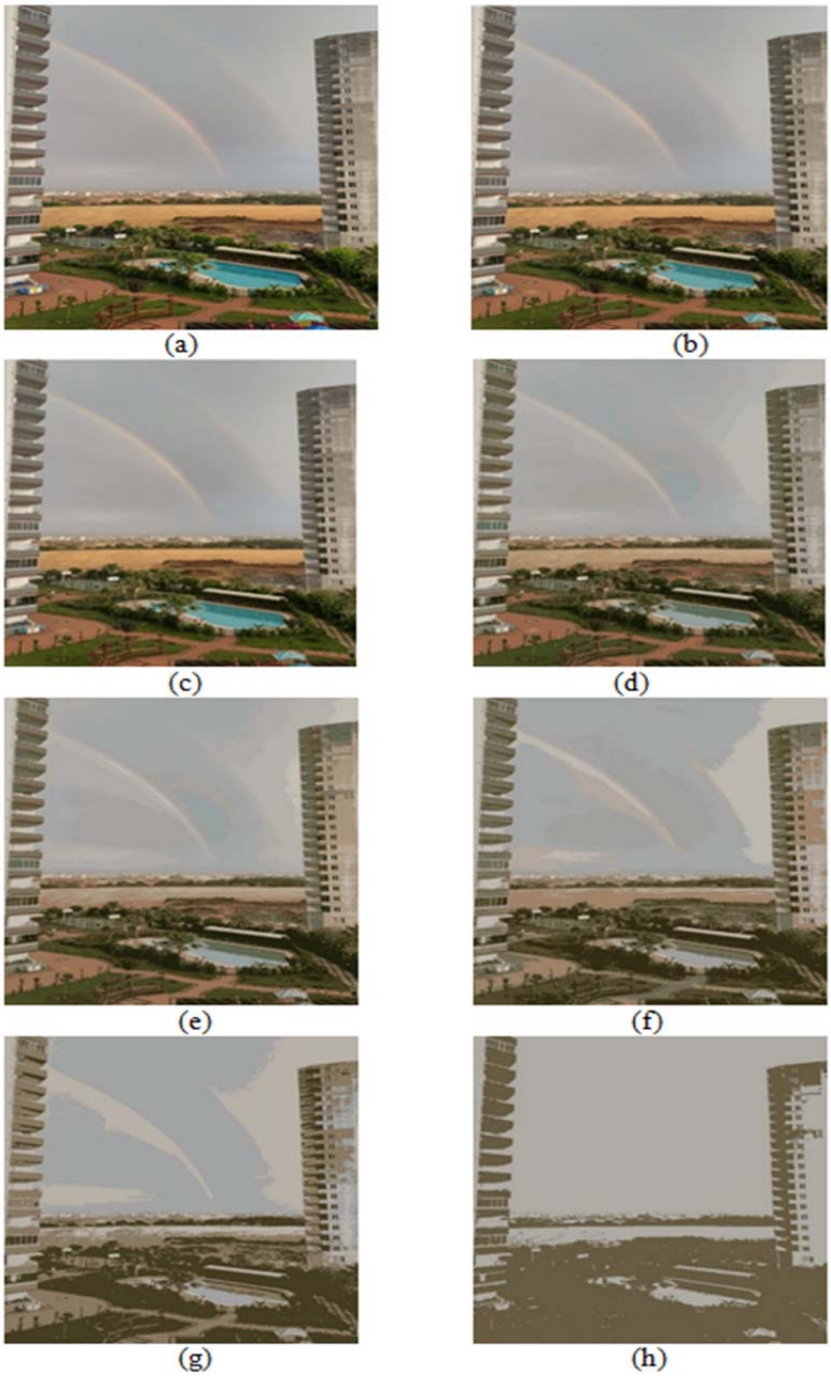


Figure 1.3. A sample image's colors are converted (quantized) from 512-bit to 256-bits, 128-bits, 64-bits, 32-bits, 16-bits, 8-bits,4-bits, 2-bits.

1.3.2. Image Processing Operations

Once the image is digitized by using sampling, and quantization, and is successfully stored in the computer, there are certain operations that can be implemented according to the goals. The operations can be categorized according to the location and numbers of the pixels of the input image. The operations which were investigated are shown below in figure 1.4.

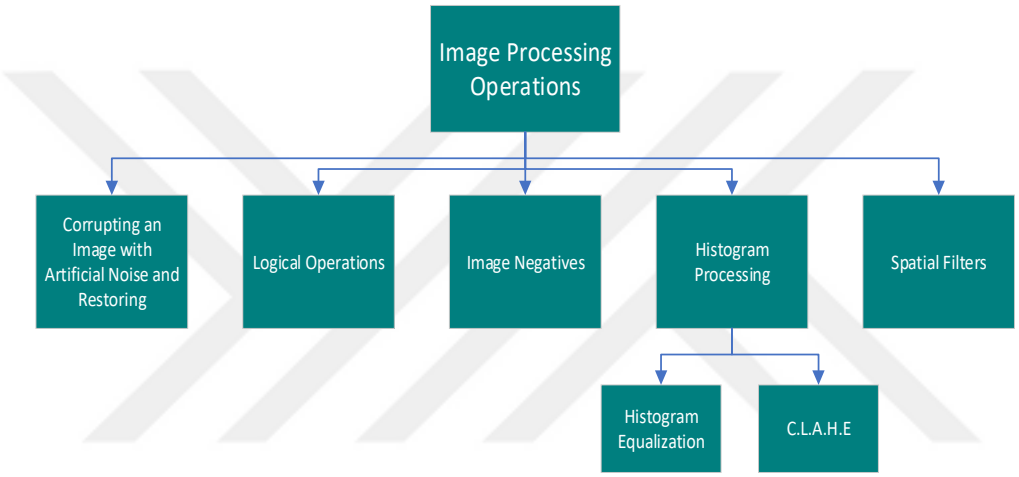


Figure 1.4. Investigated image processing operations.

1.3.2.1. Corrupting an Image with Artificial Noise and Restoring

Arithmetical operations are used to carry out the array operations between pixel pairs for an image. The four operations: summation, subtraction, multiplication, and division are shown in equation 1.1.

$$\begin{aligned} h(x,y) &= f(x,y) + g(x,y) \\ l(x,y) &= f(x,y) - g(x,y) \\ t(x,y) &= f(x,y) \times g(x,y) \\ e(x,y) &= f(x,y) \div g(x,y) \end{aligned} \tag{1.1}$$

The operations are implemented between pixel pairs in image functions *f* and *g* for *x* = 0,1,2,3,…, *m* – 1 and *y* = 0,1,2,3,…, n – 1 where, as commonly, *m* and *n* are the row and column sizes of the images, respectively. That means the images *h*, *l*, *t*, and *e* are in the size of *m* × *n*.

If *g(x, y)* is a corrupted image created by additional noise, *β(x,y)*, on a noiseless image *f(x,y)*. Then,

$$g(x,y) = f(x,y) + \beta(x,y) \tag{1.2}$$

In this example, it is assumed that every pair of coordinates *(x,y)* have uncorrelated noise and the following objective is to reduce the noise by adding a set of noisy images, $g_i(x,y)$. By averaging *K* different noisy image, the image, $\bar{g}(x,y)$, can be formed as in equation 1.3, when the noise satisfies the constraints stated.

$$\bar{g}(x,y) = \frac{1}{K}\sum_{i=1}^{K} g_i(x,y) \tag{1.3}$$

Let *E*, which is the expected value of $\bar{g}$, and let $\sigma^2_{\bar{g}(x,y)}$ and $\sigma^2_{\eta(x,y)}$ be the variances of $\bar{g}$ and η at all coordinates of (x,y). The relationship is shown below in equation 1.4, 1.5, and 1.6.

$$E\{\bar{g}(x,y)\} = f(x,y) \tag{1.4}$$

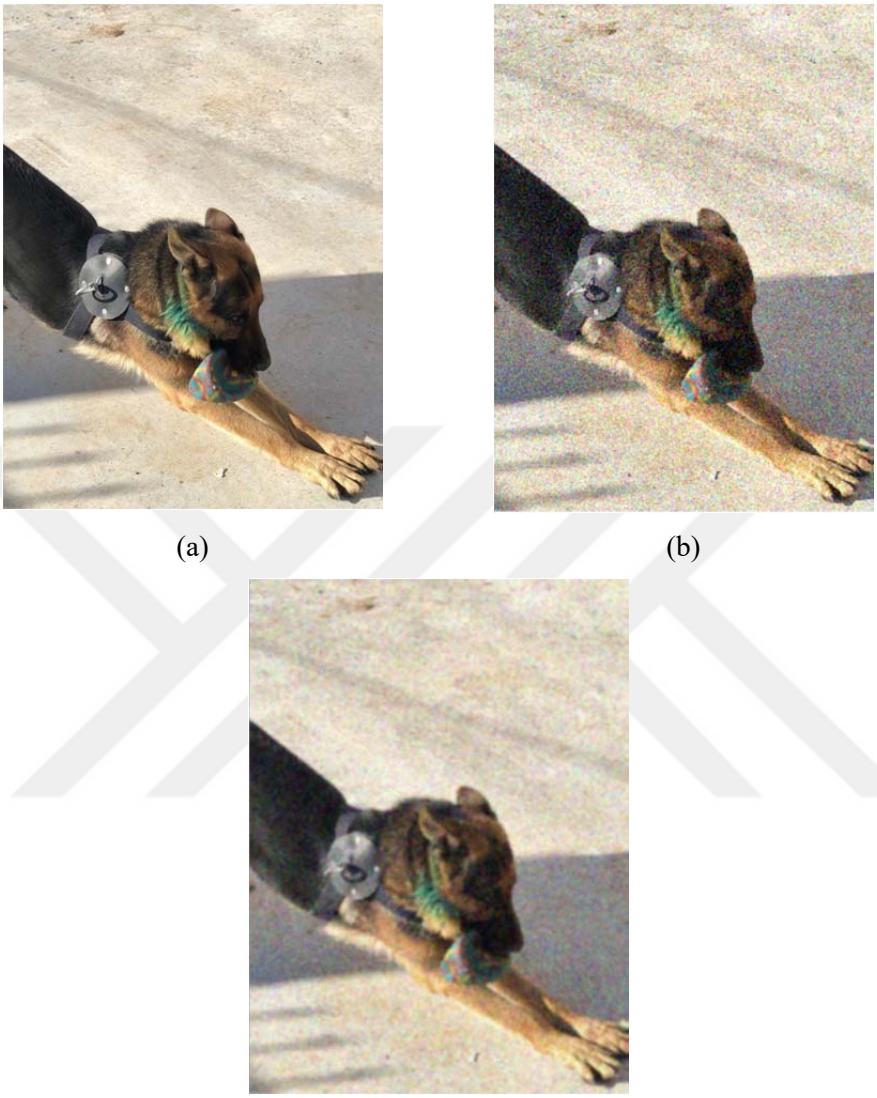
and

$$\sigma^2_{\bar{g}(x,y)} = \frac{1}{K}\sigma^2_{\eta(x,y)} \tag{1.5}$$

Therefore:

$$\sigma_{\bar{g}(x,y)} = \frac{1}{\sqrt{K}}\sigma_{\eta(x,y)} \tag{1.6}$$

It is shown that when *K* increases Eq. (1.5) and (1.6) indicate that the variability of the pixel values at every each location as (x,y) decreases. Because Eq. (1.4) indicates that $\bar{g}(x,y)$ is approaching $f(x,y)$ when the number of noisy images is used in the averaging process increases. In figure 1.5(a), a dog image is shown. The image is corrupted by using the gaussian noise using Eq. 1.2, with zero, mean in figure 1.5(b) (Sokolov, 2016). In figure 1.5(c) shows that, by averaging all the pixels under a kernel area of 5x5, the corrupted image became reasonably clean. That process can be considered as an additive operation.



(a) (b)

(c)

Figure 1.5. An image sample, corrupted and restored. (a) The original image, (b) corrupted image, (c) restored image.

Shading correction can be used on sensor images, especially in x-ray images, denoted by $f(x,y)$, using products with a shading function denoted by $h(x,y)$ to produce the improved image denoted by $g(x,y)$ as shown in Eq. 1.7 (Kaeppler, et al., 2016). Masking is another common application of the image

multiplication, it is also called a region of interest (ROI), operations. ROI determines the region where the operation is focused on. The methods which are decided to be used are used in the ROI region in order to complete the task of the application.

$$g(x,y) = f(x,y) \times h(x,y) \tag{1.7}$$

1.3.2.2. Logical Operations

Binary images are composed of pixels, which are the set of foreground pixels valued as 1, and background pixels valued in 0. The regions or objects can be defined as the composition of foreground and background pixels. In figure 1.6, A and B are the sets of coordinates in 2-D space. Shaded spaces represent the output of the logical operation. Between foreground and background pixels, the regions or objects can be used as input for logical operations. In figure 1.7, common practices referred to as union, intersection, and the complements, are obtained using the logical operations: OR, AND, and NOT, respectively.

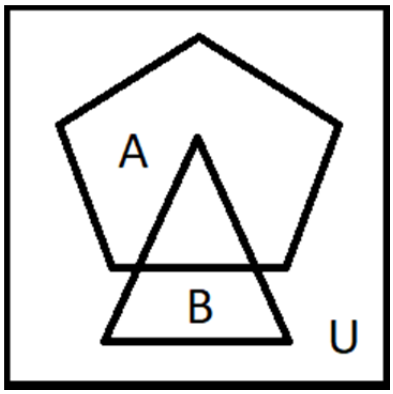
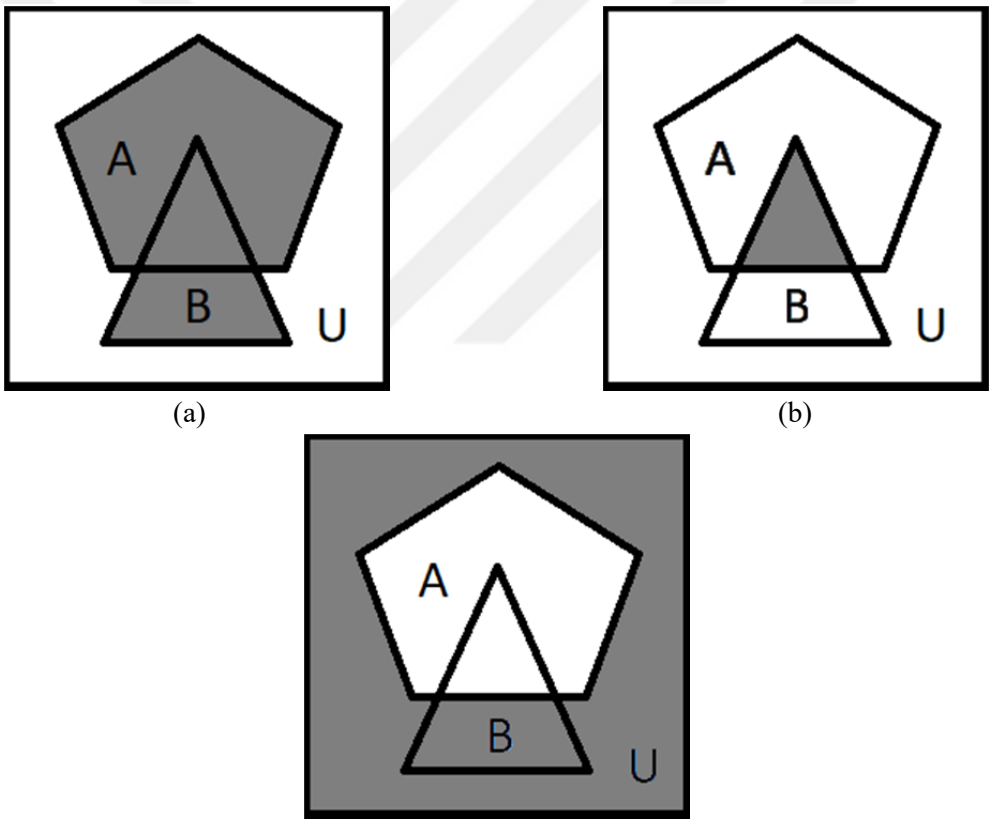


Figure 1.6. A and B are two sets of coordinates; U is a 2-D plane.

When OR operation is implemented in two regions (sets) A and B, the output is the union of the two regions, shown as foreground pixels valued as 1. AND operation sets the intersection region between the sets A and B foreground valued 1, the rest of the regions are valued as 0. Not operation implemented on A, is the region of elements not in A. During logical operations, elements of logical 0 are indicated with black, and the elements of logical 1 are indicated with white color. Since the spatial dimensions of the pixels are the same, this operation can be counted as grayscale operations.



(a) (b)

(c)

Figure 1.7. The logical operations on 2-D sets of coordinates, A and B: (a) OR operation, (b) AND operation, (c) NOT operation

1.3.2.3. Image Negatives

Suppose the normalized values of the pixels as before and after the process are b and a, where T is the process function shown as $a = T(b)$. The negative of an image is the intensity level of the pixels in a range of $[0, L-1]$ can be obtained by the negative transformation. The expression of the negative transformation is given in Eq. 1.8.

$$s = L - 1 - r \tag{1.8}$$

In its simplest form, reversing the intensity level of any image's pixels turns the input image into the photographic negative. In figure 1.8 (a), an MRI (Magnetic Resonance Imaging) of a brain tumor is processed and the negative of the image is shown in figure 1.8 (b) (Kayalı, et al., 2018).

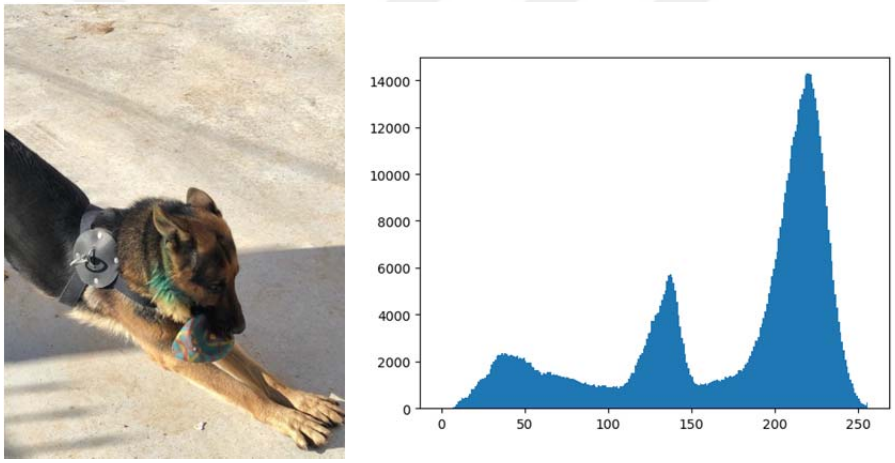
(a) (b)

Figure 1.8. An MRI image of a tumor example, (a) before the image negative, (b) the image negative.

1.3.2.4. Histogram Processing

A histogram is a diagram that shows the intensity levels of the pixels in the range $[0, L-1]$ of a discrete function $h(r_k) = p_k$, where k is the intensity

number, r_k is the k^{th} intensity value, and p_k shows the numbers of pixels which have the r_k the level intensity in the input image. Normalizing is implemented on a histogram by dividing each of the input image's components by the total number of spatial pixels located in the input image. If the row and column dimensions of the input image are m and n, then the total number of the pixels is the multiplication of the row and column. Operation $Norm(r_k) = r_k/mn$ for each k to $L-1$ is implemented in order to normalize the intensities so that the sum of all components of the input histogram is 1. In figure 1.9, the histogram of the input image is shown. Two type of the histogram equalization will be discussed in this section: Histogram Equalization and Contrast Limited Adaptive Histogram Equalization.



(a) (b)

Figure 1.9. (a) An input image of a dog, (b) histogram of the input image.

1.3.2.4.(1). Histogram Equalization

It is mentioned that the intensity of a black colored pixel is shown as 0, and that of white-colored is 1. When the input image is too bright or too dark, adjusting the grayscale of the image helps to map the image onto a uniform histogram (Acharya et al.,2005). Implementation of the histogram equalization for a grayscale image has 4 steps. Firstly, every gray level value is collected by r_k for each pixel

by a histogram. Secondly, the cumulative frequency of the histogram received in the first step is calculated by $h(r_{k0}) + h(r_{k1}) \ldots + h(r_k)$. Thirdly, the equalized histogram is calculated as shown in Eq.1.9. Finally, all the new gray levels are replaced with their coordinates so that the new mapping is completed.

$$histeq(r_k) = \frac{\left(L * h(r_k)\right) - mn}{mn} \tag{1.9}$$

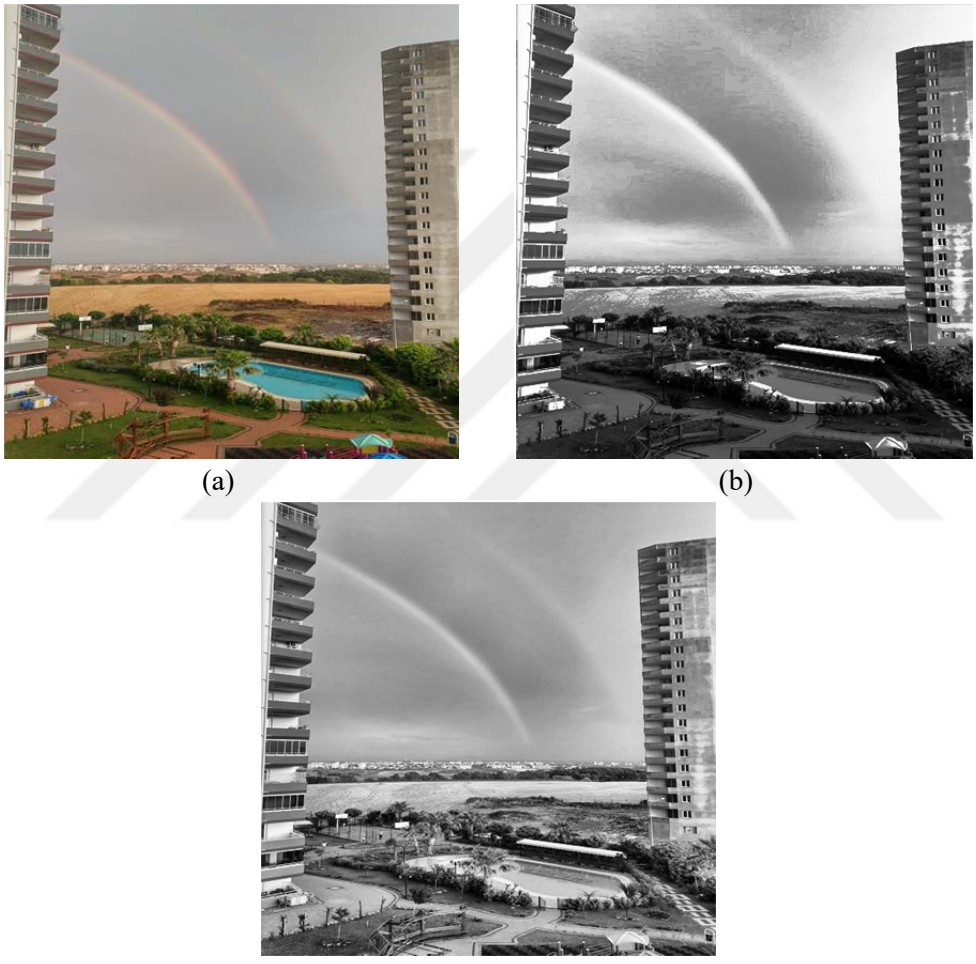
(a) (b)

Figure 1.10. (a) original grayscale image, (b) after histogram equalization.

1.3.2.4.(2). Contrast Limited Adaptive Histogram Equalization (CLAHE)

Contrast Limited Adaptive Histogram Equalization (CLAHE) is an enhancement method that is used to improve the quality of images when the image has fog or needs to be improved (Yadav, et al., 2014). This method has 6 steps, the steps are shown in figure 1.11. The implementation of the CLAHE method is shown in fig.1.12.

Figure 1.11. CLAHE algorithm steps.



(a) (b)

(c)

Figure 1.12. Histogram processing methods are summarized, (a) Input image, (b) after Histogram Equalization, (c) after C.L.A.H.E.

1.3.2.5. Spatial Filters

Spatial filters are one of the most common methods for image enhancement which is used in this field for a broad spectrum of implementation. Filters accept or reject certain frequency components. Blur (smooth) of an image, can be produced by using a low-pass filter which lets the lower frequencies only. Such operations can be implemented by spatial filters also known as a spatial mask, kernels. Spatial filtering can be described as in Eq.1.10, using a predefined operation on a pixel and its neighbors (Dawson-Howe, 2014). The operation determines the spatial filter's linearity. If the operation is linear, the filter is called a linear spatial filter. Otherwise, the filter is called a nonlinear spatial filter. Suppose x and y is a point on an image, and g is the response of the operation. During the operation x and y vary so that each pixel in operation t visits and uses every pixel in operation f. In Eq. 1.11, the illustration of a spatial filter using neighborhood at the size of 3x3, where the row and column number indicated with m and n.

$$g(x,y) = t(-1,-1)f(x-1,y-1) + t(-1,0)f(x-1,y) + \cdots + t(0,0)f(x,y) + \cdots t(1,1)f(x+1,y+1) \quad (1.10)$$

It is assumed that the size of the filter is determined by $m = 2a + 1$ and $n = 2b + 1$ where the a and b are integer numbers. The components of the Eq.1.10 and the filtering process are summarized in figure 1.13. The linear spatial filter of the input image with the size of mxn is shown in Eq.1.11. Each pixel in w visits every pixel inside of the f.

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t-b}^{b} w(s,t)f(x+s,y+t) \quad (1.11)$$

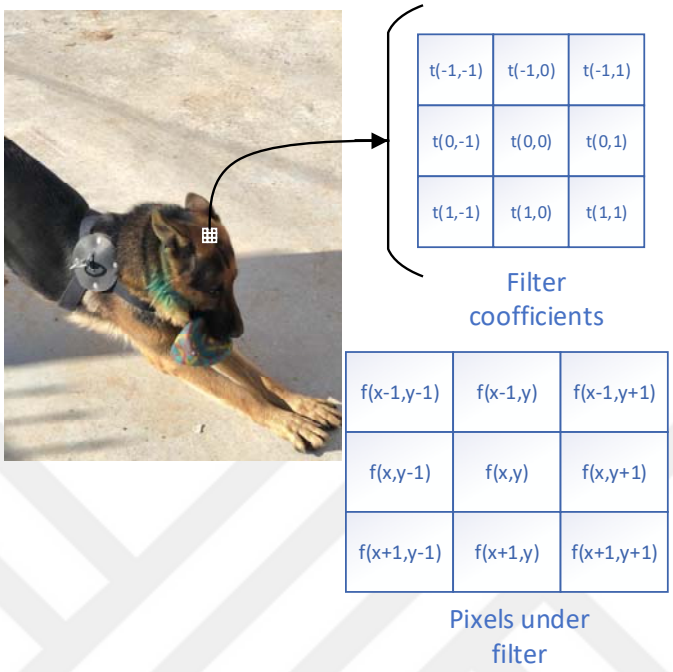


Figure 1.13. A 3x3 spatial filter is summarized according to Eq.1.10.

1.4. Deep Learning

The ability of computer systems to learn from experience, having an understanding of the connections between concepts and hierarchy, and giving a solution for such relations between concepts is called by many names. Today, it is commonly called Deep Learning (Goodfellow, et al., 2017). Deep learning is one of the most used abilities and highly profitable. Many top technology companies such as Microsoft, Google, Facebook, IBM, Apple, Baidu, Netflix, YouTube, NVIDIA and many others use deep learning during their operations, so that they can predict current habits of the operation and customers. In this section historical approach to deep learning, fundamentals of applied math and machine learning, numerical computation basics for deep learning, machine learning basics, fundamentals of convolutional networks shall be discussed.

1.4.1. Historical Approach to Deep Learning

One of the biggest dreams of the inventors was inventing machines that think and decide. After the invention of the programmable computers, machines started to become intelligent, and in practical applications, this ability is called Artificial Intelligence (AI). In early developments in the artificial intelligence field, solutions for problems that are challenging for human beings but straight-forward tasks for computers were found such as formula dependent mathematical operations. But the most challenging tasks were those which are relatively easy for humans but hard to accomplish for computer systems, such as recognizing spoken words or identifying faces in any image.

Deep learning is known by many names. During the development of biological learning (McCulloch, et al., 1943) in the 1940s-1960s, it was called Cybernetics, in the 1980s-1990s when back-propagation was invented (Rumelhart, et al., 1986) it was known as Connectionism. After 2006, the name “deep learning” started to appear in researches referring to multi-layer neural networks (Hinton, et al., 2006). There are many examples of successes of AI even in early developments such as IBM’s Deep Blue chess-player system which defeated the world chess champion, Garry Kasparov, in 1997, logistic regression algorithm was invented in 1990 and it could determine whether the delivered output was recommended or not (Mor-Yosef, et al., 1990).

In the next section, certain computation, machine learning, convolutional network basics to accomplish such AI applications shall be discussed.

1.4.2. Numerical Computation Basics

In this section, numerical computation basics such as overflow, underflow, poor conditioning, and gradient-based optimization shall be discussed to have a better understanding of the coming sections.

1.4.2.1. Overflow and Underflow

During the continuous math operations on digital devices, minimizing the accumulation of rounding error is important. One of the rounding problems is called underflow. Underflow errors occur when the operation rounds the values close to the zero. Most of the software environments avoid the division by zero, and underflow causes the system to fail the operation. Another rounding error is overflow. Overflow causes the system to fail when the system takes the values with a high magnitude to the maximum value of the system. So that most of the arithmetic operations do not take the magnitude as a number-values.

1.4.2.2. Poor Conditioning

In deep learning, conditioning means the reaction of a function to the small variations in the input. If a small variation in the input brings a large reaction in the output, this may cause failures in the operations, such as rounding. The term Poorly conditioning shows that the operation is not suitable for practical application because the sensitivity is at a critical level between input and output. Poorly conditioned matrices reveal the pre-existing error when the multiplication is proposed by the true matrix inverse.

1.4.2.3. Gradient-Based Optimization

Current deep learning frameworks use gradient-based optimization such as Tensorflow (Abadi, et al., 2015). Optimization process aims to minimize or maximize any function. The target function is called the objective function or criterion. During the minimization, it is also called cost function, loss function, or error function. To minimize the function f, the direction of f in which f decreases the fastest must be found. To find the direction, Eq 1.12 and Eq.1.13 are used:

$$\min_{u,u^T u=1} u^T \nabla_x f(x) \tag{1.12}$$

$$= \min_{u,u^T u=1} ||u||_2 ||\nabla_x f(x)||_2 cos\theta \tag{1.13}$$

In Eq. 1.12 and Eq.1.13, θ is the angle between u and the gradient. $||u||_2$ equals to 1. In these equations, u points if the direction is uphill or downhill as the direction of the gradient. Therefore, it is possible to decrease the function f by moving in the negative direction of the gradient. This method is known as the steepest descent or gradient descent.

1.4.3. Machine Learning Basics

In this section, machine learning and its basics are discussed, since deep learning is a specific type of the machine learning. Learning algorithms, capacity, overfitting, underfitting, hyperparameters, and cross-validation shall be discussed.

1.4.3.1. Learning Algorithms

The main principle of the machine learning is an algorithm that can learn from data in short. The fundamentals of the learning are an experience E to learn from, class of tasks T, and measurement of the performance P. Then by measuring learning performance while operating the tasks T, as measured by P, and improving the learning with experience E. The tasks T, the performance P, and the experience E shall be discussed in sections below.

1.4.3.1.(1). The Task, *T*

The task is not the learning, the task represents the ability to perform as an output of the learning. As an example, if we want a robot to swim, then the task is swimming. Machine learning tasks are described by features of the task example.

Features are the collection of needed parameters of the process. Typically, we describe the features by vectors. There are many machine learning tasks according to their purposes. The well-known tasks are:

- Classification is a task type which system is asked to find out the category the inputs belong to. One of the most common examples is object recognition where input is often an image and the output is a numerical class of an object.
- Classification with missing inputs task, is more challenging than a casual classification, because it uses inputs that are not fully provided. The learning must be implemented on set of functions. So, every single function can correspond to a different sub-set of the inputs which have missing data. Medical diagnosis can be given as an example. In some cases, hospitals may use it when they have not all the equipment or some of the test may be too expensive to perform, but the results are needed to marginalize out of the missing variables as a task (Laencina, et al., 2007).
- Regression is a task for which a computer program predicts numerical values based on numerical inputs. It is common in banking, where banks try to predict the future prices of securities.
- Transcription is another task where a computer program observes and tries to predict the unstructured form of data, and then transcribe it into a textual form. For example, in speech recognition in which the program takes an audio waveform as an input, observes the sequence of characters then describes the words which are included in the audio. Another example is text recognition which program takes a handwriting or any image of a text as an input and tries to describe the text as characters.
- Machine Translation is a task where input already consists of symbols in some other language and the program tries to convert it to another

language where online language translator programs commonly do the task.

- Anomaly Detection is a task where the computer program observes a sequence of events, detects the habits and routines. Then flags the unusual changes in the routines. This task is often used in bank account management, where clients' accounts are observed and when a suspicious withdrawal occurs, the bank interferes with the operation or informs the client about it.

1.4.3.1.(2). The Performance Measure, *P*

Measurement of the Performance P is specific to the task *T*. Tasks measure the accuracy to determine the performance. This indicates how accurate the model. Measuring the proportion of examples that failed or generated incorrect outputs, and the correct outputs gives the error rate. In such measurements, the correct output is represented by 1 and incorrect output represented by 0. This is called the 0-1 loss. The main interest is having the best performance of the algorithm when deployed in the real world. Also, testing the model with test data of the dataset can indicate how accurate is the model.

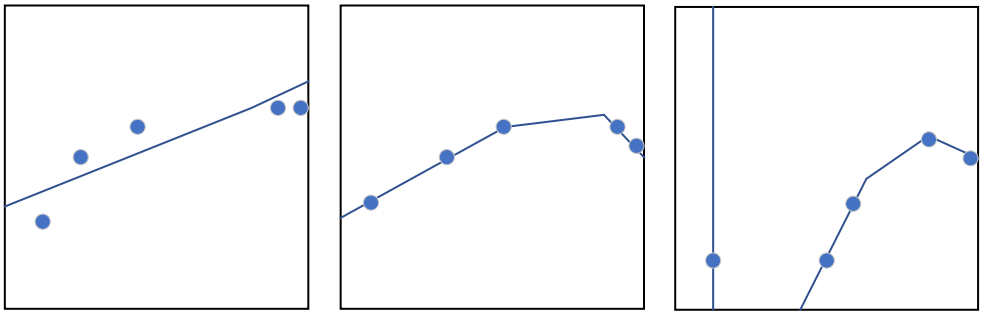
1.4.3.1.(3). The Experience, *E*

The experience from the task flows through the learning. During the learning, a dataset is used to have the features for the task. A certain percentage of the data is labeled as training data, and the rest is labeled as the test data. For example, one of the most well-known datasets is the Iris dataset (Fisher, 1936). It has 4 features: sepal length, sepal width, petal length, petal width and 3 classes: Iris Setosa, Iris Versicolour, and Iris Virginica. The machine learning algorithms can be commonly separated into two groups as unsupervised and supervised during the learning. Unsupervised learning is a type of learning that, the training data have no target attribute. The model is not supervised. Common uses of unsupervised

learning are clustering and association. On the other hand, supervised learning algorithms use the dataset which includes features, and each featured example is also associated with a label or target (annotation).

1.4.3.2. Capacity, Overfitting, and Underfitting

During learning the model is trained on the training data set, but the model must have the ability to perform the operation on unseen inputs as well. This is called generalization. Typically, the generalization error, also called the test error, is desired to be as low as possible. Because, it shows the performance of the machine learning algorithm's ability to perform the task. The factors to improve the machine learning algorithm's abilities are making these training errors small, and making the gap between the test error, and training as small as possible. These factors create two challenges: underfitting and overfitting. Underfitting occurs when the training error value is not low enough on the training set, and overfitting occurs when the gap between the test error and training is too large. Capacity describes the control of the eagerness of the model between overfitting, and underfit. In figure 1.14, three models as an example for each satiation for underfitting, appropriate capacity, and overfitting. In figure 1.14 (a) there is a function that suffers from underfitting, and from random inputs x could not fit the outputs y. In figure 1.14 (b), the function generalizes the data well to unseen points and it has an appropriate capacity. In figure 1.14 (c), the function suffers from overfitting and the gap between the test error, and training is too large.



(a) (b) (c)

Figure 1.14. Examples of three different model's training set: (a) Underfitting, (b) appropriate capacity, (c) overfitting.

1.4.3.3. Cross-Validation

Problems may occur if the fixed training test set is too small. A small test dataset makes it difficult to understand if the algorithm works well on the task. Because, a small dataset implies uncertainty. This is not a serious concern if the dataset includes hundreds of thousands of examples, or more. If this problem occurs, it is needed to increase the computational cost, and use all the examples in the estimation of the mean test error (mte). This includes using one of the solution algorithms, which is repeating the training process on different or random parts of the dataset which is also called k-fold cross-validation.

1.4.4. Convolutional Networks

Convolutional Neural Networks (CNN) are special kind of neural networks which contains a grid-like topology. Convolution is a specialized kind of linear mathematical operation in a place of matrix multiplication. CNN examples change according to the inputs. The operations are called a 1D grid sample if the samples are regular time intervals.

1.4.4.1. The Convolutional Operation

The general form of the convolutional operation s is given in Eq. 1.14. x is the input, w is the weight function, also known as kernel which gives weighted values to recent measurements at a time t, and with an average value of a. The output is referred to as the feature map.

$$s(t) = \int x(a)\, w(t-a) da \tag{1.14}$$

The convolution operation is often described by an asterisk, as shown in Eq.1.15.

$$s(t) = \int (x * w)(t) \tag{1.15}$$

Inputs of convolutional neural networks are usually multidimensional arrays of data of any type. In neural networks, multidimensional arrays are called tensors. Because each unit of the input has a meaning, it must be stored separately. Usually, it is assumed that, these tensors are valued zero everywhere, but it has finite values on set points.

In our case, we use two-dimensional images Im(m,n) as inputs for the networks. So, our kernel needs to be two-dimensional, too. The expression is shown in Eq. 1.16:

$$s(i,j) = (\mathrm{I} * \mathrm{K})(\mathrm{i},\mathrm{j}) = \sum_m \sum_n I(m,n)\, K(i-m, j-n) \tag{1.16}$$

The example of a 2D convolution operation is shown in figure 1.15. The operation is indicated by arrows, and boxes to show the target input tensor, and kernel tensor.

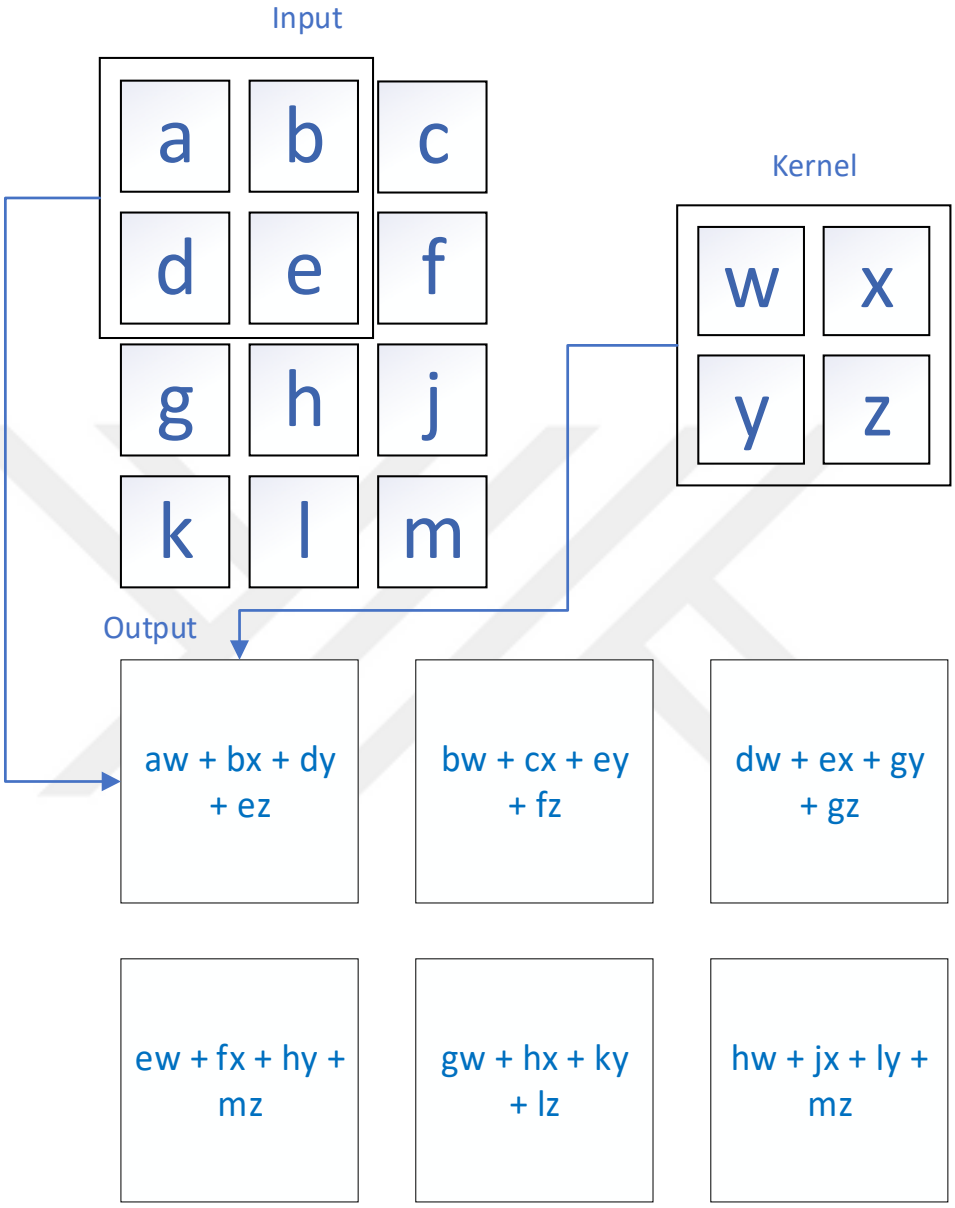


Figure 1.15. A 2-D Convolution operation of an input image.

1.4.4.2. Pooling

There are 3 different stages of a convolutional network layer as shown in Fig.1.16. In the first stage of the layer, multiple convolutions are carried out in parallel to create a set of linear activations. In the second stage, these linear activations are run through a single non-linear activation function. This stage is also referred to as the detector stage. In the final stage, a pooling function is used to modify the output through the next layers of the convolution.

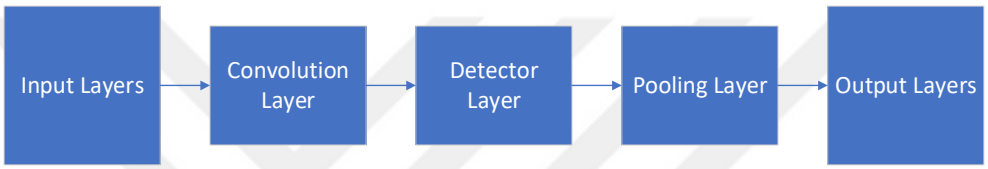


Figure 1.16. Three stages of a convolutional network.

Maximum pooling operation in the third stage with a layer of 2x2 modifies the specific output data into half of the height and the width, reducing the size in the total of a quarter in cases where the next layer uses the quarter size of the data, as shown in Fig. 1.17.

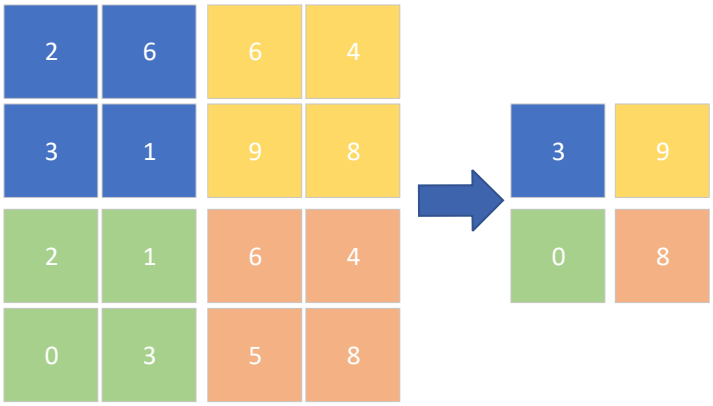


Figure 1.17. Maximum pooling layer.

2. IMPLEMENTATION OF THE METHODS

2.1. Introduction

In this chapter, all the phases of the thesis from the equipment and datasets to the methods, and their results are discussed individually.

The images required for training and testing were captured using a quadcopter according to the scenes which simulate and consist of a traffic site with three different classes: Person, Car, and Motorbike. After the images were captured, they were processed in a computer system containing a GPU. The computer system processed the images through the layers of the models and labeled the classes in the test images in boundary boxes as outputs. The flowchart of the process is shown in Figure 2.1. During the process, pre-trained data, and a new dataset that was created from the captured images from UAV were used and compared. Also, three different GPUs were used individually and compared.

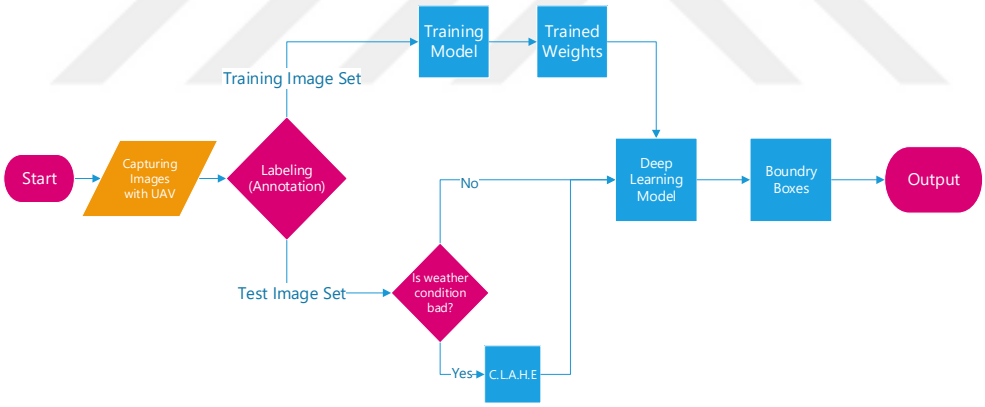


Figure 2.1. Flowchart of the process of the images taken from the UAV.

2.2. Inventory and Graphics Processing Units (GPU)

To capture the images a quadcopter was used. The quadcopter has the ability to move in 6 directions (front, backward, right, left, hover down, hover up). The maximum endurance flight time is 27 minutes. The 12 megapixels onboard camera is located on the front edge of the UAV which can capture 1080p with 96

fps, and 4k images with 30 fps. The quadcopter UAV is shown in figure 2.2. The camera's angle is adjustable within a range of 0º - 90º. The camera angle was 45º during the recording to make images realistic for multiple applications, such as monitoring a room, security applications, etc.

Graphic Processing Units (GPU) operates faster than the Central Processing Unit (CPU) with its ability to operate in parallel. This ability made GPU popular in many applications, such as video game industries, etc. GPUs also proved themselves in convolutional operations and become common in deep learning applications. Three different GPUs were used to compare the performance for all of the methods on each GPU: NVIDIA GeForce 750M®, NVIDIA Jetson TX2 Developer Kit®, and NVIDIA GTX 1060®. All the GPUs were chosen from NVIDIA due to its compatibility with used GPU libraries: CUDA, and CuDnn. The NVIDIA Jetson TX2 Developer Kit is a development GPU with an Ubuntu® 16.04 operating system in it and it was used to demonstrate the operations. The Jetson TX2 is shown in Figure 2.3.

The images were captured by the quadcopter, to feed the training, and test datasets. 400 unique images were obtained, the images and their replicas, in order to prevent under-sampling, were used in a new dataset, and the test data.

Figure 2.2. The quadcopter with an onboard camera used to capture images.

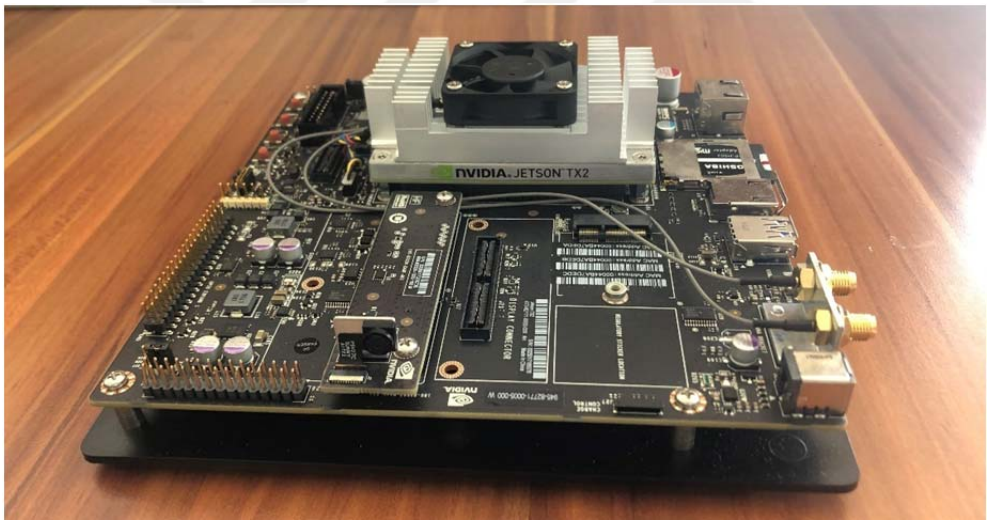


Figure 2.3. NVIDIA Jetson TX2 Developer Kit.

2.3. Transfer Learning, and Datasets

During the operation, two different training data were used. The first one was the Microsoft COCO: Common Objects in Context, and the second one was a new dataset created with the images taken from UAV. In this section, these two datasets will be discussed.

2.3.1. Microsoft COCO: Common Objects in Context

Microsoft COCO: Common Objects in Context is an image dataset released on 21 February 2015. The images were taken from everyday scenes. All the target objects were labeled using per-instance segmentation, and boundary boxes. The dataset consists of 91 different objects, and 2.5 million labeled instances in 328 thousand images. The classes of the instances are person, bicycle, car, motorcycle, airplane, bus, train, truck, boat, traffic light, fire hydrant, street sign, stop sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, hat, backpack, umbrella, shoe, eye glasses, handbag, tie, suitcase, frisbee, skis, snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket, bottle, plate, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, couch, potted plant, bed, mirror, dining table, window, desk, toilet, door, tv, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, blender, book, clock, vase, scissors, teddy bear, hair drier, tooth brusher, and hair brush. Since the Microsoft COCO includes the 3 classes of our operation which are person, car, and motorbike, it was suitable to use the dataset. We used compatible pre-trained convolutional weights which were trained on Microsoft COCO. During the network of the framework, changing the fully connected layer of the architectures into 3 made it possible to use it in our application.

2.3.2. The New Dataset

A new dataset was created by using the quadcopter. The dataset includes 400 unique images with 3 traffic classes: person, car, motorcycle. Since the first attempts had failed due to lack of data, replica images were created by adding and subtracting a gamma to the images, and mirroring them in the x-direction. The example images from the dataset are shown in figure 2.4. The images were captured in the size of 3840x2160, then the instances were cropped into 640x480.

Figure 2.4. Some of the images from the new dataset.

All the class objects in the images taken from the UAV were annotated one by one. Images were aimed to have at least one of the target class objects in an actual traffic scene with environmental décors such as roads, streetlights, bus stations, etc. Annotations, and boundary boxes of instances were labeled by a basic tool we designed in Python UI. The tool allows the user to draw, and move a boundary box that is drawn from the upper left corner to the bottom right corner, and then the user labels the box. As an output, a file in Extensible Markup Language (XML) format is created. The XML file consists of annotation parameters for an object. Parameters are the folder name, file name, folder's path directory, file's source, the image's size, segmentation status, object's name,

object's pose, truncation status, and difficulty. This format allows the dataset's images to be trained on Faster-R CNN, and YOLOv2 algorithms. An annotation example is shown in figure 2.6, for a processed image shown in figure 2.5.

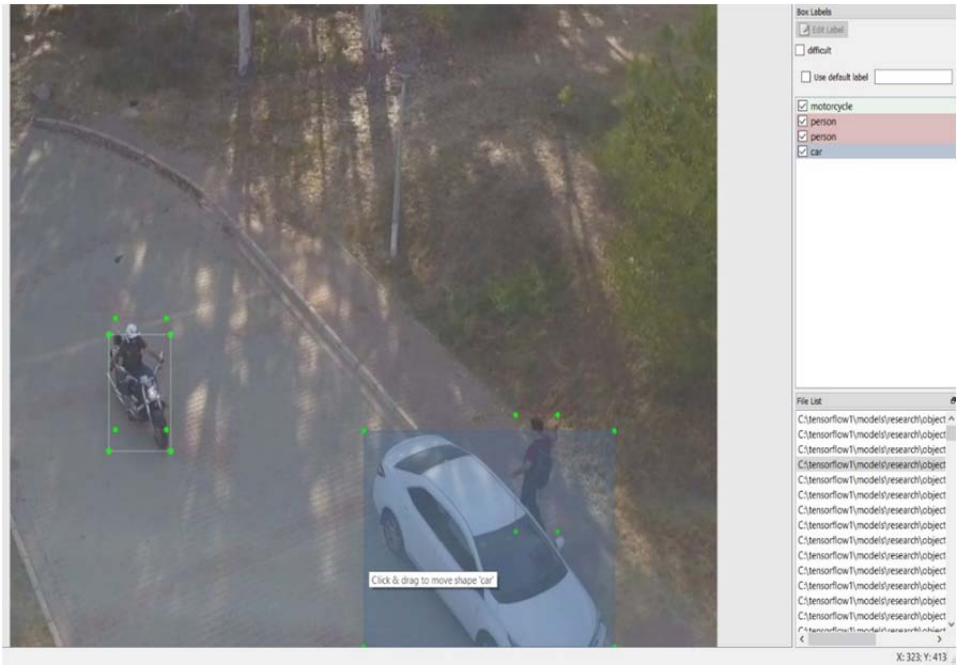


Figure 2.5. The basic tool we created, using Python UI.

```
<annotation>
        <folder>train</folder>
        <filename>103.jpg</filename>
        <path>C:\tensorflow1\models\research\object_detection\images\train\103.jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>640</width>
                <height>480</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>motorbike</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>86</xmin>
                        <ymin>243</ymin>
                        <xmax>148</xmax>
                        <ymax>339</ymax>
                </bndbox>
        </object>
        <object>
                <name>person</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>93</xmin>
                        <ymin>240</ymin>
                        <xmax>140</xmax>
                        <ymax>301</ymax>
                </bndbox>
        </object>
        <object>
                <name>person</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>449</xmin>
                        <ymin>305</ymin>
                        <xmax>493</xmax>
                        <ymax>398</ymax>
                </bndbox>
        </object>
        <object>
                <name>car</name>
                <pose>Unspecified</pose>
                <truncated>1</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>326</xmin>
                        <ymin>321</ymin>
                        <xmax>543</xmax>
                        <ymax>480</ymax>
                </bndbox>
        </object>
</annotation>
```

Figure 2.6. The annotation for a given image in figure 2.5 in XML format.

2.4. Faster R-CNN with Inception v2

Faster R-CNN: Region-Based Convolutional Neural Network is a state-of-the-art object detection network that focuses on regional proposals to detect the object locations. It has better results than the previous version: Fast R-CNN (Girshick, 2015), R-CNN (Girshick, et al., 2014). Faster R-CNN proved itself in ILSVRC: ImageNet Large Scale Visual Recognition Challenge (Russakovsky, et al., 2015) and Microsoft COCO 2015 competitions as the algorithm won the first place. The algorithm works in two phases called modules. The first module is the deep fully convolutional network which is used for the proposal of the regions. The second module is the detector of the Faster R-CNN which uses the regional proposals from the first module. The block diagram of the model is shown in figure 2.7. Faster R-CNN also uses an attention-based model technique which was introduced in 2015 for speech recognition (Chorowski, et al., 2015). Region Proposal Network RPN) uses input images of any size, and gives rectangular boundary boxes with object scores. Region proposals are produced by sliding small networks on feature maps using the last shared convolutional layers.

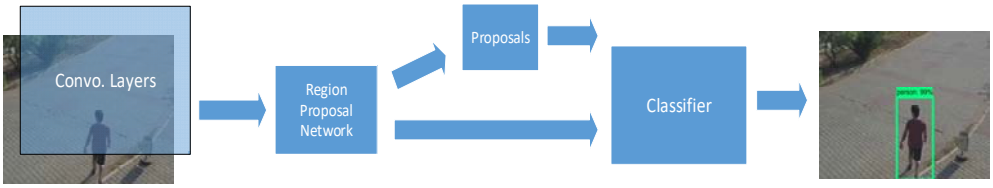
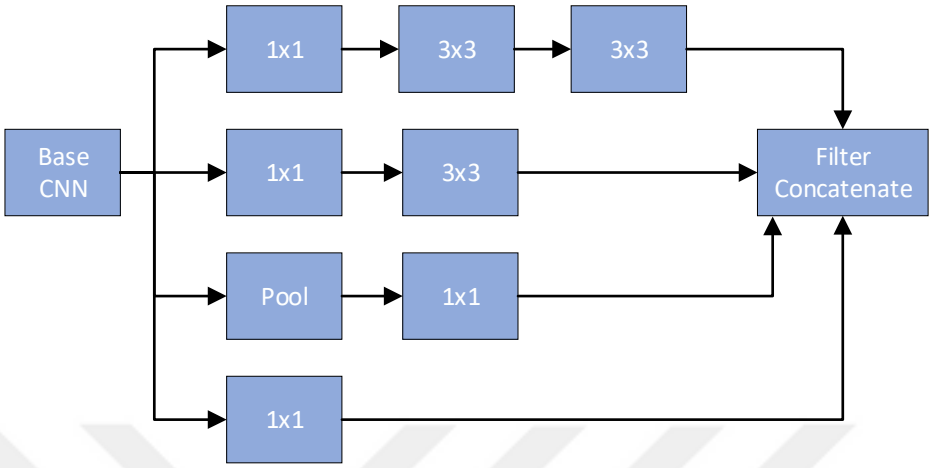
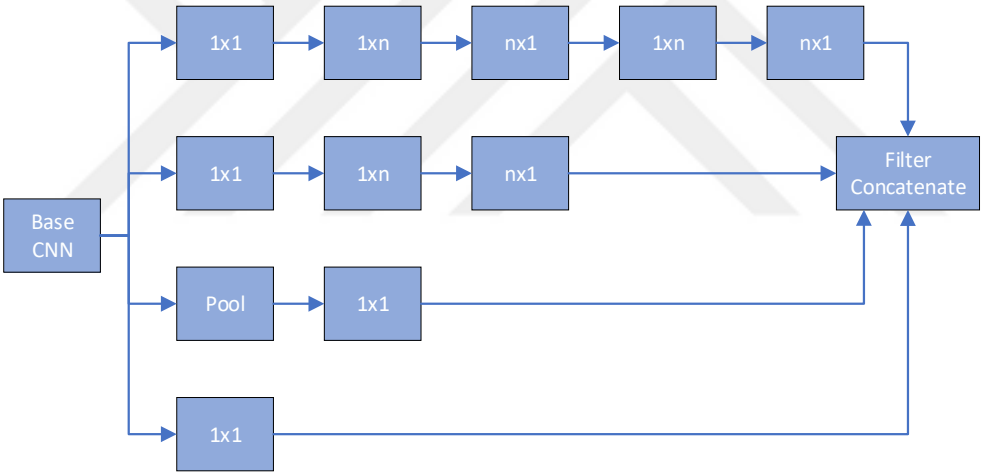


Figure 2.7. Block diagram of the Faster R-CNN model.

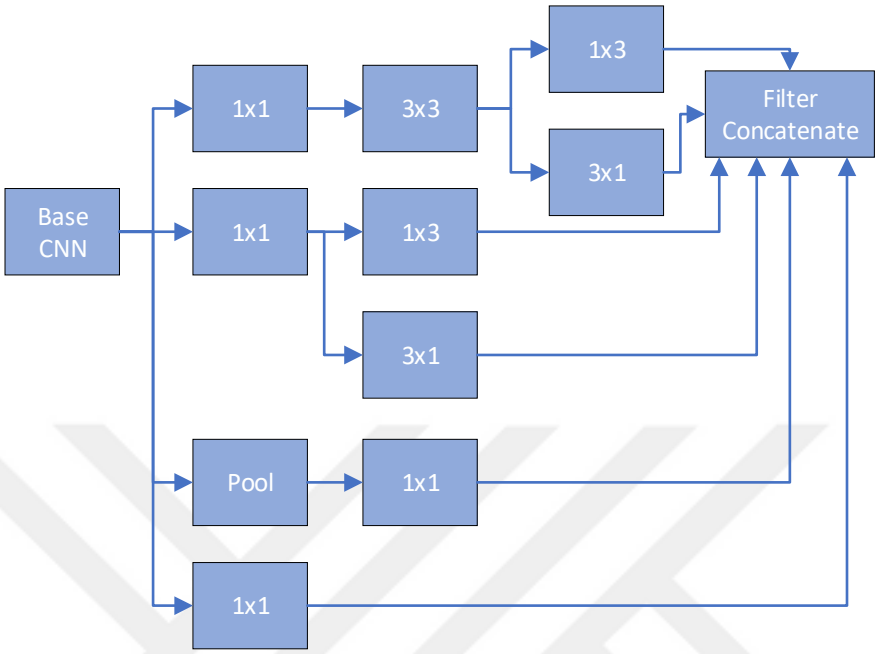
Inception v2 is a tool that is used to reduce the complexity of the deep convolutional neural network. It makes these convolutional networks wider than deeper (Alamsyah et al, 2019). Inception v2 has 3 different modules types: A, B, and C. Every module can change the dimensions of CNNs as shown in figure 2.8.


Base CNN
1x1
3x3
3x3
1x1
3x3
Pool
1x1
1x1
Filter Concatenate

(a)


Base CNN
1x1
1xn
nx1
1xn
nx1
1x1
1xn
nx1
Pool
1x1
1x1
Filter Concatenate

(b)



(c)

Figure 2.8. Inception modules: (a) A, (b) B, and (c) C.

2.5. You Only Look Once (YOLO) v2

You Only Look Once (YOLO) is an object detection algorithm which can apply classification and localization for target objects by a single convolutional neural network. This ability provides YOLO to run in a high speed, therefore it fits to real-time applications. Therefore, we used it as a fast object detector (Eriş, et al., 2019). YOLO trades accuracy for faster speed. YOLOv2 has a greater performance and has a change at the end of the network. A fully connected layer was changed with an additional convolutional, and average-pooling layer. The model's architecture is shown in Table 2.1.

Table 2.1. The network architecture table of YOLOv2.

Layer Description	Filters	Size/Stride	Output
1- Convolutional Layer	32	3x3	608x608
2- Maxpool Layer	32	2x2	304x304
3- Convolutional Layer	64	3x3	304x304
4- Maxpool Layer	64	2x2	152x152
5- Convolutional Layer	128	3x3	152x152
6- Convolutional Layer	64	1x1	152x152
7- Convolutional Layer	128	3x3	152x152
8- Maxpool Layer	128	2x2	76x76
9- Convolutional Layer	256	3x3	76x76
10- Convolutional Layer	128	1x1	76x76
11- Convolutional Layer	256	3x3	76x76
12- Maxpool Layer	256	2x2	38x38
13- Convolutional Layer	512	3x3	38x38
14- Convolutional Layer	256	1x1	38x38
15- Convolutional Layer	512	3x3	38x38
16- Convolutional Layer	256	1x1	38x38
17- Convolutional Layer	512	3x3	38x38
18- Maxpool Layer	512	2x2	19x19
19- Convolutional Layer	1024	3x3	19x19
20- Convolutional Layer	512	1x1	19x19
21- Convolutional Layer	1024	3x3	19x19
22- Convolutional Layer	512	1x1	19x19
23- Convolutional Layer	1024	3x3	19x19
24- Convolutional Layer	512	1x1	19x19
25- Convolutional Layer	1024	3x3	19x19
26- Router	512	16	38x38
27- Convolutional Layer	64	1x1	38x38
28- Local Flatten	256	2x2	19x19
29- Router	1280	27,24	19x19
30- Convolutional Layer	1024	3x3	19x19
31- Convolutional Linear	3	1x1	19x19

The YOLOv2 model consists of 3x3 convolutional layers, followed by a 2x2 max-pooling layer which reduces the image data into quarter during the training, and fully connected layer for detection. The router layers were used to concatenate the previous, and the next layers. A 2x2 local flattening layer was implemented to change the size form.

2.6. Software Implementation

In this section, the implementation steps, and parameters of YOLOv2, and Faster R-CNN were discussed. Firstly, the pre-process which is implemented for both methods was introduced. Secondly, YOLOv2 implementation, and the method's results were introduced statistically. Finally, Faster R-CNN, and results were shown, and the methods were compared in terms of performance, speed, and accuracy.

Software implementations were done in Python 3 with Anaconda® environment, and Ubuntu® 16.04 LS operating system. We used TensorFlow in both methods as these were composed of a convolutional neural network. For the YOLOv2 method we used Darkflow API which translates the Darknet API (Application Programming Interface) into Tensorflow functions (Redmon, 2016), and for Faster R-CNN method we used Tensorflow API. Tensorflow itself is not a user-friendly library, where creating a network with Tensorflow from scratch can be crucial for users. These APIs made Tensorflow more user-friendly, and they allowed us to train, and test our data on our own dataset with the parameters we chose. Also, we could compare the results by implementing the same methods on the data we gathered, and pre-trained values taken by the Microsoft COCO dataset. Open Source Computer Vision Library (OpenCV) allowed us to choose our images as inputs, and implement image processing algorithms on both pre-process and post-process during the operations (Pulli, et al., 2012). The libraries and importers that we used can be listed as Tensorflow, Numpy, Cython Matplotlib, Darkflow, Darknet, Os, Imp, Opencv, TFNet, and Time.

2.6.1. Pre-process

The test images in a scene where the weather is not clear, or the image quality is low were indicated manually as scenes under bad conditions. In such cases, Contrast Limited Adaptive Histogram Equalization (CLAHE) was implemented to clear such bad conditions located on photos during both operations YOLOv2, and Faster R-CNN. The images were resized according to the operation without changing the aspect ratio. If the input was a rectangle, the missing pixels were filled with RGB (0,0,0), black color, to fit the image into a square size. For YOLOv2 input images were resized into 608x608. For Faster R-CNN, images were resized into 640x640.

2.6.2. YOLOv2 Implementation

The high-resolution input test images were processed according to the pre-process discussed in the previous section. Then two different ways of object detection were implemented.

Firstly, Microsoft COCO pre-trained weights were used to feed the network by using Darkflow API. The training parameters, and detection parameters to complete the process were shown in table 2.2. The flow diagram of the algorithms we used is summarized in figure 2.9. The parameters, and the layers were defined in a configuration file in the format of .cfg to feed the Darkflow API. Every layer step shown in Table 2.1, were listed in the configuration file, therefore it allowed us to build the network. Pre-trained weights were stored in a yolov2.weight file. To use the Darkflow API, there are specific commands which are required to be used with the python code to run the object detection. --model command calls the configuration file which held the YOLOv2 network architecture including all the layers and filter parameters in our case. After --model command, the directory of the config file was shown. --load command calls the weight files in order to carry the convolutional neural networks out, in our case we showed the directory of the pre-trained weights trained on Microsoft COCO. --imgdir

command was used to show the directory of the test images. --gpu command allows us to choose the percentage of the required GPU performance to run the network. In our case, we needed a full performance in order to operate a real-time system, we set the --gpu parameter into 1.0 which meant %100 performance access. If the user needs any other operations to run at the same time, the user should leave a spare GPU memory. The python code we wrote included the TFNet library which provides the user to use multiple TensorFlow commands with its own functions. We coded the options of the model directory, weight directory, and the confidence score threshold of 0.4. Using the OpenCV library, we connected the images into our code. According to the weights, the network and parameters, we reached results by returning predictions of the object from the detection. We set a color randomizer for boundary boxes which started from the upper left corner to bottom right corner and the labels. The labels were shown near the boundary box as shown with examples in figures 2.10, 2.11, 2.12, 2.13. The object names and the confidence scores of the given examples were shown in tables 2.3, 2.4, 2.5, 2.6, 2.7. For multiple instances in an output, image were numbered in the tables with an order from left to right.

Table 2.2. The YOLOv2 parameters for training and detection.

YOLOv2 Parameters	
Learning Rate	0.002
Momentum	0.9
Weight Decay	0.0005
Hue Rates	0.1
Saturation	1.5
Exposure	1.5

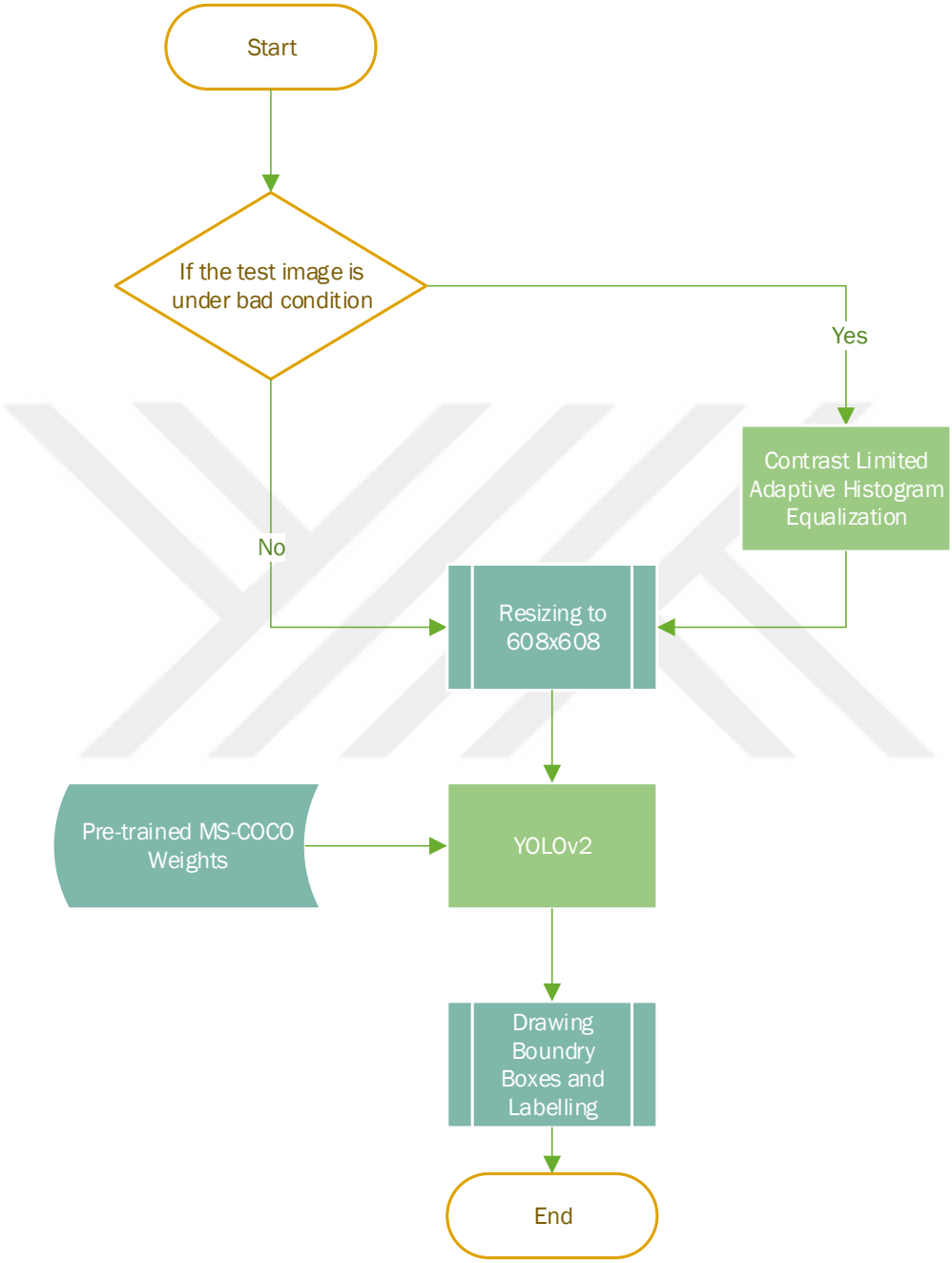


Figure 2.9. The flowchart of the YOLOv2 implementation with pre-trained weights on the Microsoft COCO dataset.

Figure 2.10. Output of the aerial images of a person, and a car with labels, and boundary boxes at 10 mt. altitude.

Table 2.3. The confidence score table with labels for figure 2.10.

Fig.2.10. Detected Object Labels	**Confidence Scores**
Person	0.67
Car	0.45

Figure 2.11. Output of the aerial images of two cars, and a person on a motorbike with labels, and boundary boxes at 15 mt. altitude.

Table 2.4. The confidence score table with labels for figure 2.11.

Fig.2.11. Detected Object Labels	**Confidence Scores**
Person	0.31
Car-1	0.74
Car-2	0,70
Motorcycle	0.32

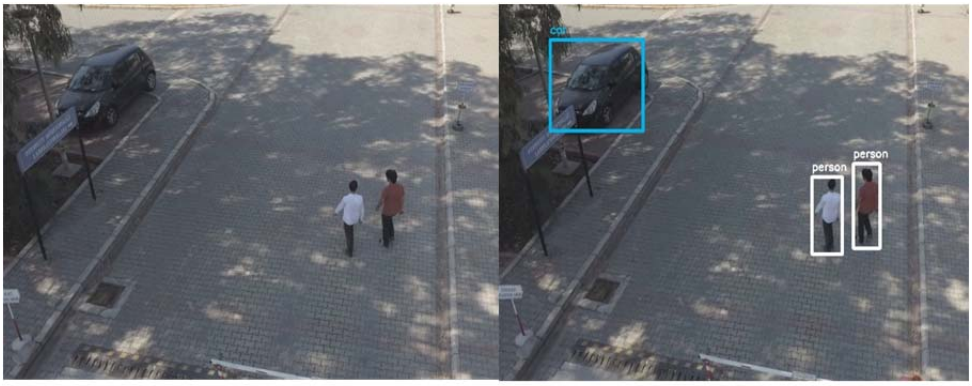


Figure 2.12. Output of the aerial images of a car, and two people with labels and boundary boxes at 10 mt. altitude.

Table 2.5. The confidence score table with labels for figure 2.12.

Fig.2.12. Detected Object Labels	**Confidence Scores**
Person -1	0.75
Person -2	0.78
Car	0.73

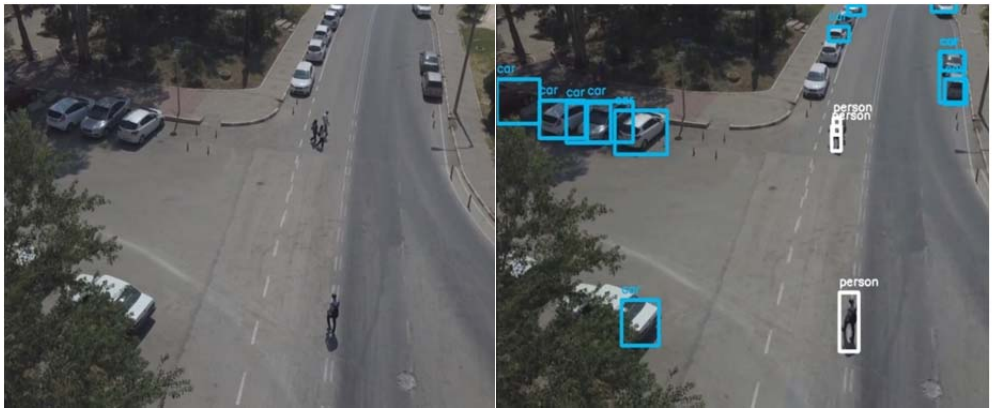


Figure 2.13. Output of the aerial images of multiple cars, and multiple person instances with labels, and boundary boxes 25 mt. altitude.

Table 2.6. The confidence score table with labels for figure 2.13.

Fig.2.13. Detected Object Labels	Confidence Scores
Person -1	0.22
Person -2	0.16
Person -3	0.24
Car -1	0.15
Car -2	0.26
Car -3	0.46
Car -4	0.63
Car -5	0.42
Car -6	0.24
Car -7	0.30
Car -8	0.26
Car -9	0.24
Car -10	0.23
Car -11	0.23

The test images were captured from 5 meters, 10 meters, 15 meters, and 25 meters. Naturally when the altitude was increased, the confidence score of the objects was decreased. The statistics of the results for 400 test images were shown

in table 8. From 5 meters to 15 meters the confidence scores varied in a range of 0.45 – 0.79. Over 20 meters, we were forced to reduce the confidence threshold to 0.20 to receive a detection result. The rest of the results were useless since their confidence score was less than 0.20. In addition, in some of the test images, there were localization errors which occurred as detecting an object although the boundary area had no target objects. Such kind of errors were indicated as a localization error. Finally, the first objective was completed using pre-trained weights and YOLOv2.

Table 2.7. Results of the YOLOv2 model with Microsoft COCO pre-trained weights on test images.

Object Classes		Unclassified	Precision (%)	Accuracy (%)	Detections with Localization Error
Total Number of Person	128	37	71,10	52,54	12
Total Number of Car	109	35	67,90	59,43	9
Total Number of Motorbike	64	21	67,18	41,25	2
Total Target Objects	301	93	69,10	53,16	23

Secondly, we trained the network with 640 training images from the new dataset using Darkflow API. In this objective, the network used the weights which were produced by training. The flow chart of the objective is shown in figure 2.14. The training was implemented with the parameters as mentioned in table 2.2. The training time was 13 hours and 34 minutes. Although the training data consisted of images from the target scenes, the results were not improved when compared to the previous objective. The YOLOv2 architecture needed a very large dataset for training, and testing. Under-sampling occurred and its precision got lower. The results were shown in table 2.8.

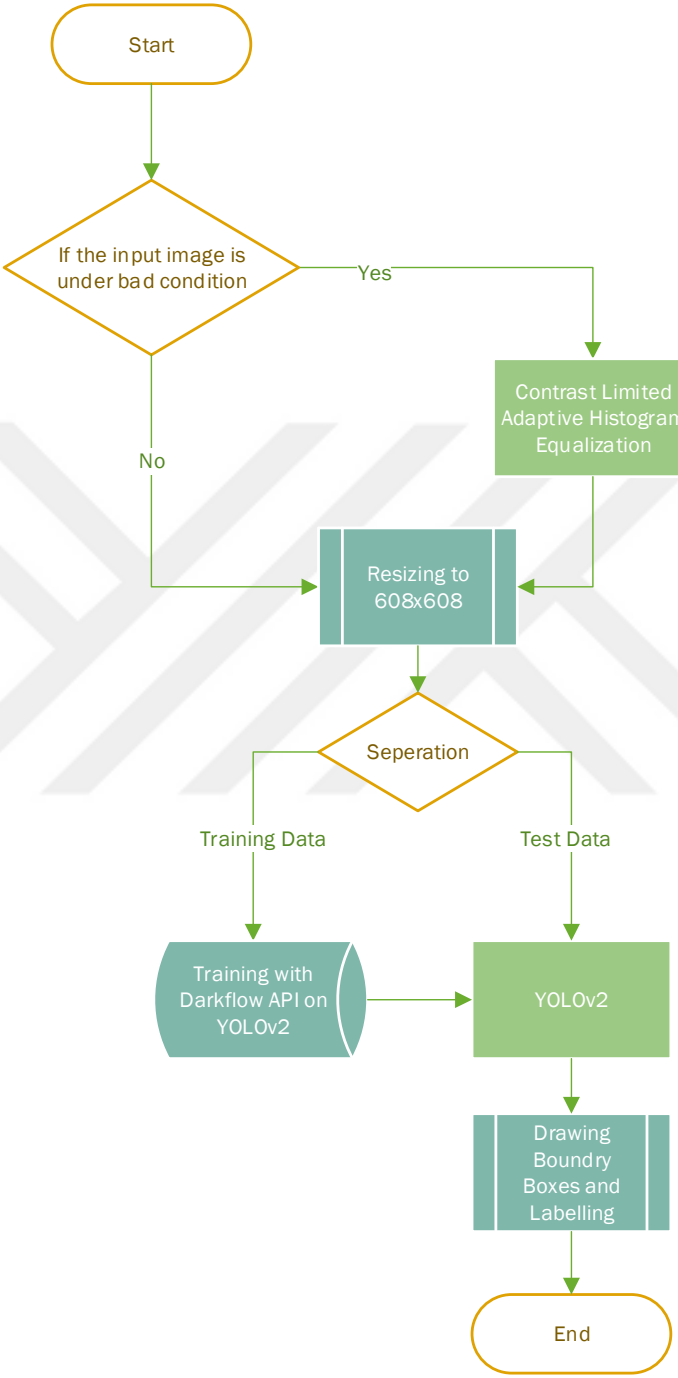


Figure 2.14. The flowchart of the YOLOv2 implementation with weights that were trained on the new dataset.

Table 2.8. Results of the YOLOv2 model with training weights produced with training data.

Object Classes		Unclassified	Precision (%)	Accuracy (%)	Detections with Localization Error
Total Number of Person	128	93	27,34	17,22<20,00	31
Total Number of Car	109	88	19,26	21,24	19
Total Number of Motorbike	64	54	15,62	27,54	2
Total Target Objects	301	235	21,92	19,78<20,00	52

Although the processing speed was fast, for higher altitudes over 20 meters concluded in too low confidence scores when pre-trained weights were used. When the training was done manually, the data we used was not sufficient, because of the under-sampling performance of the operation could not be improved. Three different GPUs were compared during the operations, all the GPUs were tested under the same conditions including CPU, and CUDA versions. The GPU performance comparison is shown in table 2.9.

Table 2.9. GPU comparison for real-time performance on the YOLOv2 model with full access to GPU.

Graphic Processing Unit	Real-time Performance
NVIDIA GT750M	1 fps
NVIDIA Jetson TX2	6 fps
NVIDIA GTX 1060	24 fps

YOLOv2 proposes a fast object detection with decent precision, and accuracy when trained on a large training dataset, and useful on UAV applications.

2.6.3. Faster R-CNN Implementation

The objectives used in YOLOv2 implementation were followed during the Faster R-CNN implementation. The pre-process was used in two different ways depending on whether the image condition was good or bad. After the pre-process Faster R-CNN method was implemented on test images with the help of the python 3 code we developed, and the official Tensorflow API. In this section, the method implementation, and the results were discussed in two objectives. The first objective was to use Faster R-CNN with pre-trained weights which were trained on the Microsoft COCO dataset on test data. The second objective was to train our weights using the training data we gathered, and test them on our test data.

Firstly, Microsoft COCO pre-trained weights were installed on the model using Tensorflow API with the Tensorflow functions. In addition to the model, Inception v2 modules were used to reduce the aggression of the convolutions on lower dimensions. After we made the configurations in the python code by setting the directories, creating label maps, setting box parameters, and creating a loop to process all the images in order, and save all the files one by one with the confidence scores on them. The flow chart of the process is shown in 2.15. Regional proposals were slowing the performance down dramatically, but the increase in both precision and accuracy had shown the trade-off. The highest operation speed with the GPUs we had was 2,64 fps. Although the operation with Faster R-CNN was not suitable for real-time operation through multiple frames, the operations showed that for critical image frames that require high accuracy and precision for object detection Faster R-CNN was useful. Output examples of the objectives are shown in figure 2.16., 2.17., 2.18. The confidence scores of the examples are given in table 2.10., 2.11., 2.12. The total summary of the objective is given in table 2.13.

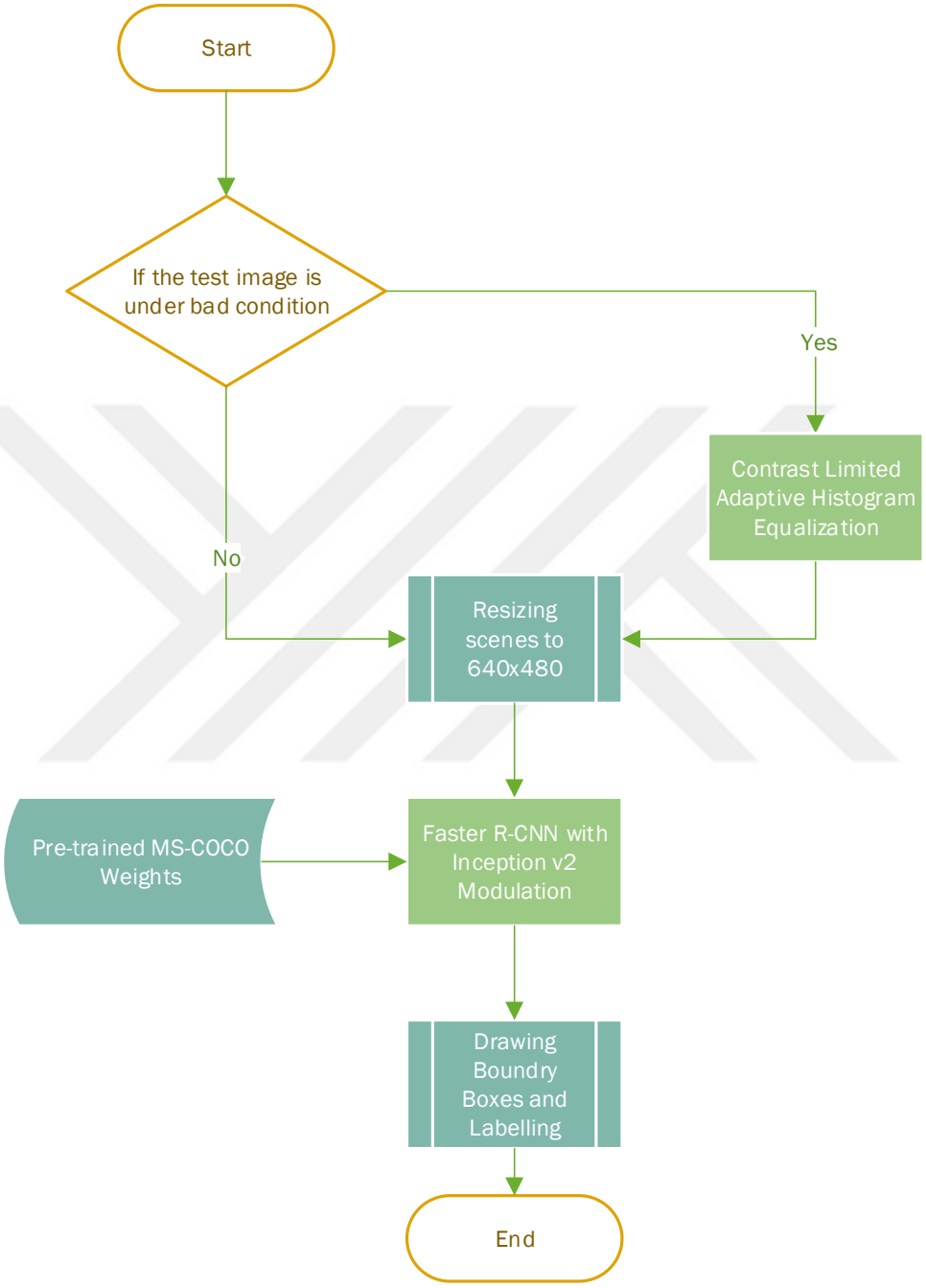


Figure 2.15. The flowchart of the Faster R-CNN model implementation with pre-trained weights on the Microsoft COCO dataset.

Figure 2.16. Output of the aerial images of two people with labels, and boundary boxes at 25 mt. altitude.

Table 2.10. The confidence score table with labels for figure 2.16.

Fig.2.16. Detected Object Labels	Confidence Scores
Person-1	0.90
Person-2	0.92

Figure 2.17. Output of the aerial images of three cars with labels, and boundary boxes at 25 mt. altitude.

Table 2.11. The confidence score table with labels for figure 2.17.

Fig.2.17. Detected Object Labels	Confidence Scores
Car-1	0.88
Car-2	0.99
Car-3	0.99

Figure 2.18. Output of the aerial images of three cars with labels, and boundary boxes at 30 mt. altitude.

Table 2.12. The confidence score table with labels for figure 2.18.

Fig.2.18. Detected Object Labels	Confidence Scores
Motorbike	0.99
Person-1	0.99
Person-2	0.99
Person-3	0,96
Person-4	0,98
Person-5	0,99
Person-6	0,99

Table 2.13. Summary of the objective of implementing Faster R-CNN on Microsoft COCO pre-trained weights.

Object Classes		Unclassified	Precision (%)	Accuracy (%)	Detections with Localization Error
Total Number of Person	128	15	88,28	87,34	12
Total Number of Car	109	13	88,07	85,22	9
Total Number of Motorbike	64	1	98,43	90,56	0
Total Target Objects	301	29	91,60	87,71	21

Secondly, the Faster R-CNN method was implemented with trained weights that were trained on the new dataset. The training parameters were the same as previously used, as shown in table 2.3. Microsoft COCO dataset has images of our target objects, but the perspective of the UAVs has a large angle. Therefore, in this objective, we trained with a dataset which was gathered by using the quadcopter. Even though the data was not enough and did not improve the process when it was implemented with YOLOv2, the training with a new dataset on Faster R-CNN with Inception v2 modulation using Tensorflow API improved the results. The localization errors were reduced, and precision-accuracy averages were increased. The flow chart of the operation was introduced in figure 2.19. The result examples of the objective were presented for target objects, person, car, and motorbike in figure 2.20., 2.21., 2.22., 2.23. The confidence scores of the examples were presented in table 2.14., 2.15., 2.16., 2.17. The training time of the data was a minimum 23 hours and 33 minutes with the same computer system introduced in the YOLOv2 method. Training time was relatively higher, and GPU usage was %100. The total summary of the objective is presented in table 2.18.

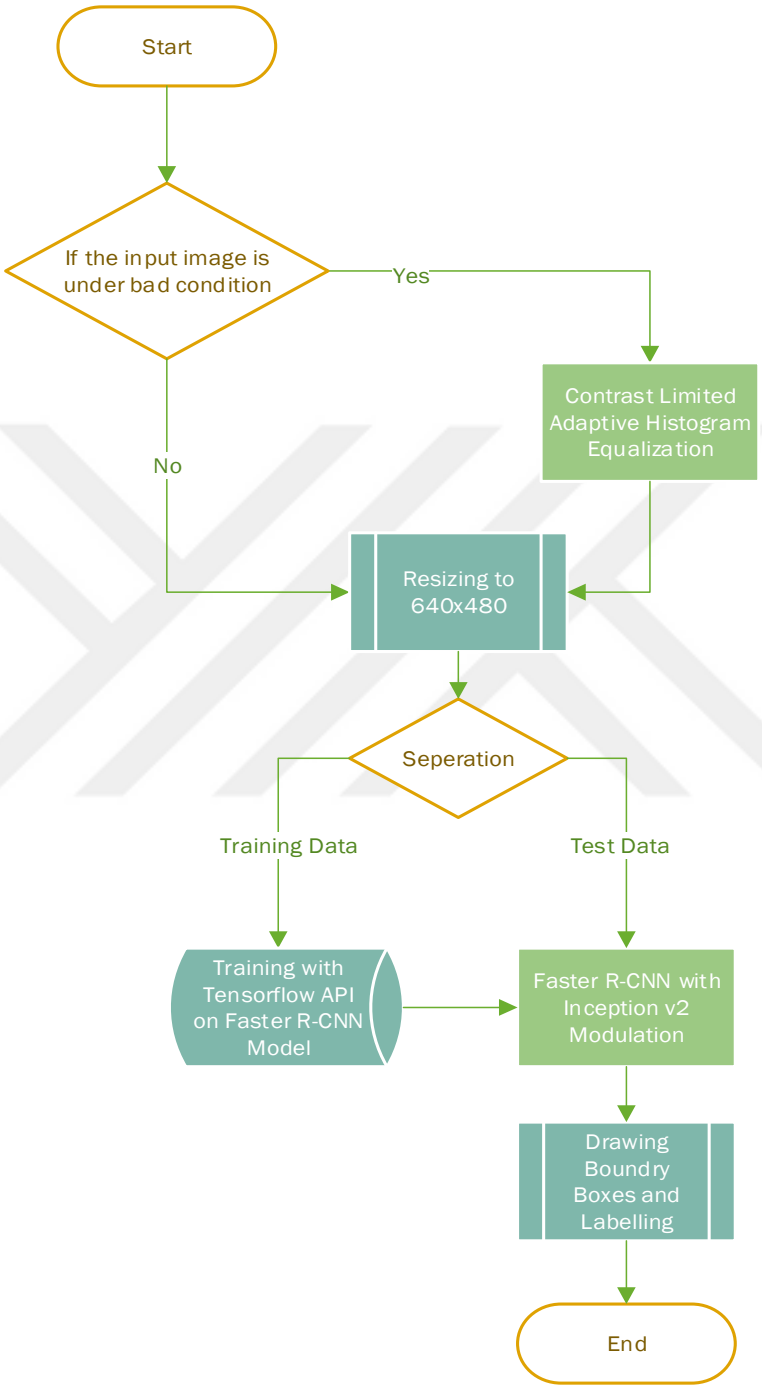


Figure 2.19. The flowchart of the Faster R-CNN model implementation with weights trained on the new dataset using Inception v2 modulation.

Figure 2.20. Detection of four cars, and two walking persons with labels, and boundary boxes at 30 mt. altitude.

Table 2.14. The confidence score table with labels for figure 2.20.

Fig.2.20. Detected Object Labels	Confidence Scores
Car-1	0.99
Car-2	0.99
Car-3	0.99
Car-4	0.99
Person-1	0,99
Person-2	0,97

GPU performances were also compared during the operation. Three GPUs were used to test the effect of the GPU during the Faster R-CNN operation with the Inception module. The comparison is shown in table 2.19.

Figure 2.21. Detection of a motorbike, a walking person and a localization error of a car with labels, and boundary boxes at 30 mt. altitude.

Table 2.15. The confidence score table with labels for figure 2.21.

Fig.2.21. Detected Object Labels	**Confidence Scores**
Motorbike-1	0.99
Person-1	0.99

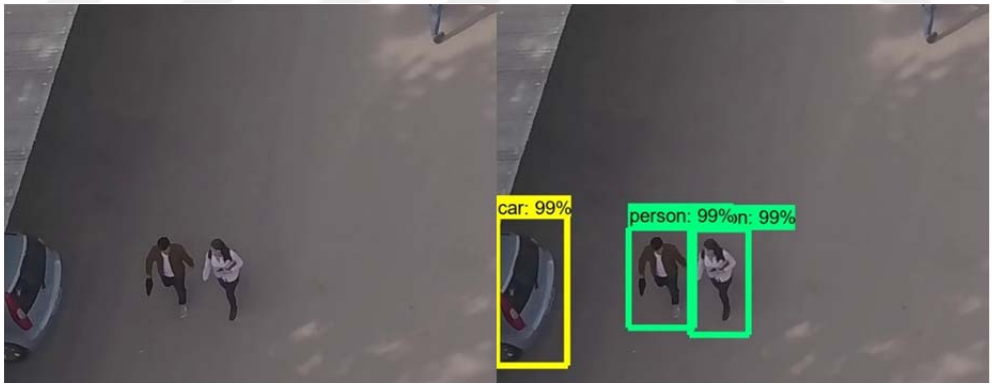


Figure 2.22. Output of the aerial images two walking person, and a car with labels, and boundary boxes at 20 mt. altitude.

Table 2.16. The confidence score table with labels for figure 2.22.

Fig.2.22. Detected Object Labels	Confidence Scores
Person-1	0.99
Person-2	0.99
Car	0,99

Figure 2.23. Output of the aerial images a walking person, and five cars with labels, and boundary boxes at 40 mt. altitude.

Table 2.17. The confidence score table with labels for figure 2.22.

Fig.2.23. Detected Object Labels	Confidence Scores
Person-1	0.99
Car-1	0.99
Car-2	0,99
Car-3	1,0
Car-4	0,99
Car-5	0,99

Table 2.18. Summary of the objective of implementing Faster R-CNN with training data that trained on the new dataset.

Object Classes		Unclassified	Precision (%)	Accuracy (%)	Detections with Localization Error
Total Number of Person	128	12	90,62	98,53	14
Total Number of Car	109	8	92,66	99,07	13
Total Number of Motorbike	64	4	93,75	98,83	3
Total Target Objects	301	24	92,34	98,81	30

Table 2.19. GPU comparison for real-time performance on Faster R-CNN with Inception v2 modules with full access to GPU.

Graphic Processing Unit	Real-time Performance
NVIDIA GT750M	0,08 fps
NVIDIA Jetson TX2	0,47 fps
NVIDIA GTX 1060	**2,64 fps**

The frames were processed continuously to test the model using the real-time performance. The speed was low, but the trade-off between accuracy-precision and speed was clear.

3. DISCUSSION

In the previous section, the results were shown with each model, and each objective with a test on every GPU model to observe the performance of the models for specific GPUs. The test images were captured from 5, 10, 15, 25, 35, and 45 meters altitude. The minimum threshold was %20 for the YOLOv2 model, and it was %80 for Faster R-CNN as default. The results showed that YOLOv2 was a fast object detection model that was applicable to real-time operations if it were trained on a very large dataset with target object class instances. The new dataset training set was not enough to increase the accuracy, and precision. The state-of-the-art datasets such as Microsoft COCO provided better results with YOLOv2 high speed. YOLOv2 with Microsoft COCO dataset had %48,74 confidence, the new dataset decreased the confidence to under %20 threshold. Faster R-CNN was slow but a confident model with a much higher accuracy, and precision with pre-trained weights from Microsoft COCO. In addition, we could increase the confidence of the model by using the new dataset which is more adaptive to the UAV's perspective with the scenes we captured. The total confidence of the model with Microsoft COCO dataset was %87,71 on our test data, we could increase the confidence to %98,81 with training data collected from actual scenes. The performance through 3 GPUs was always better with more powerful GPUs such as NVIDIA GTX 1060. Besides, the GPUs used %100 memory, which means if it is aimed to process such models, a higher GPU memory is a need. In our case, we used 6 gigabytes of GPU memory during the process. The localization error which occurred as detecting an object on a place where there was no object instance. It was decreased by increasing the training data of the object class in similar scenes. The summary of the results is given in Table 3.1. The training time per model, and tool is shown in Table 3.2. The performance summary is given in Table 3.3.

Table 3.1. Summary of the results of YOLOv2 and Faster R-CNN object detection models with two objectives.

Object Detection Model and Objectives	Total Unclassified Objects	Precision (%)	Accuracy (%)	Total Localization Error
YOLOv2 trained on MS COCO	93	69,10	53,16	23
YOLOv2 trained on new dataset	235	21,92	19,78<20,00	52
Faster R-CNN trained on MS COCO	29	91,60	87,71	**21**
Faster R-CNN trained on new dataset	**24**	**92,34**	**98,81**	30

Table 3.2. The training duration per model, and tool.

Training Model + Tool	Training Time
YOLOv2 model with Darkflow API	13 hours 34 minutes
Faster R-CNN with Inception v2 module with TFNet	23 hours 33 minutes

Table 3.3. Real-Time operation test of the models with each GPUs.

Object Detection Models	Real-Time Performance of the Models per GPU		
	NVIDIA GT750M	NVIDIA GTX 1060	NVIDIA Jetson TX2
YOLOv2	1 fps	**24 fps**	6 fps
Faster R-CNN	0,08 fps	**2,64 fps**	0,47 fps

The trade-off between performance-confidence was shown in Table 3.4. The highest speed of the GPUs, and the average confidence scores were taken into consideration.

Table 3.4.The performance-confidence table.

Object Detection Model and Objectives	Precision (%)	Accuracy (%)	Performance (fps)	Total Localization Errors
YOLOv2 trained on MS COCO	69,10	48,74	**24**	23
YOLOv2 trained on new dataset	21,92	19,78<20,00	**24**	52
Faster R-CNN trained on MS COCO	91,60	87,71	2,64	**21**
Faster R-CNN trained on new dataset	**92,34**	**98,81**	2,64	30

The Faster R-CNN is useful in operations in which confidence is critical, and more important than the performance as the single frames from UAV from the exact scene during the application. YOLOv2 is more suitable for applications such as the aerial movement of the UAVs since the movement is fast, and the camera output is a fluent frame sequence that requires a real-time operation. Both models are easy to implement using APIs. Both models required large datasets according to application scenes. But, YOLOv2 had a better trade-off.

4. CONCLUSION AND FUTURE WORK

In this study, the test images were captured using a UAV from actual scenes consisting of the target object classes (person, car, and motorbike) for object detection. The state-of-the-art object detection models, YOLOv2, and Faster R-CNN were implemented for two objectives. Firstly, the models were tested with pre-trained weights, which were trained on Microsoft COCO: Common Objects in Context. Secondly, the models were tested on a new dataset created by the images taken from the UAV. The results were compared. YOLOv2 was a fast and suitable real-time object detection algorithm for UAV's movement when it was trained on a large dataset. The objects were detected with a %69,10 precision, %53.16 accuracy, and 23 localization error at 24 fps speed when the YOLOv2 model was trained on Microsoft COCO. When YOLOv2 was trained on the new dataset, the results were not improved. The precision decreased to %21,92, the accuracy went down to 19,78, and the localization error increased to 52. The threshold accuracy was %20, and YOLOv2 had a lower accuracy when it was trained on a smaller dataset. The model suffered from under-sampling. Faster R-CNN was more accurate, but slower algorithm than the YOLOv2. The Faster R-CNN should be used in cases in which UAV has a mission to detect the target objects on a certain scene, and the accuracy is critical. Instead of using the algorithm on a video stream, using the method on well-caught image frames. During the tests, the Faster R-CNN reached %91,60 precision, %87,71 accuracy, and 21 localization errors at 2,64 fps when it was trained on Microsoft COCO. When the Faster R-CNN algorithm was trained on the new dataset, output images were more adaptive to object classes, so the results were improved. The precision increased to %92,34, the accuracy increased to%98,81. On the other hand, the localization error also increased to 30 due to a lower number of image samples. This situation may be solved by adding more object class image samples from all directions. As future work, optimization algorithms such as particle swarm algorithm (Kennedy, et al., 1995) may be used to

utilize the parameters, and to reach higher accuracy, and precision rates. The dataset may be enriched by capturing more image samples to make the models more confident.

REFERENCES

Abadi, Martin, et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2015.

Basavaprasad, B and Ravi, M. 2014. A Study on the Importance of Image Processing and Its Applications. [ed.] NCRIET-2014. s.l. : International Journal of Research in Engineering and Technology, 2014. pp. 155-160. Vol. 03.

Chorowski, Jan, et al. 2015. Attention-Based Models for Speech Recognition. s.l. : Neural Information Processing Systems (NIPS), 2015.

Dawson-Howe, Kenneth. 2014. A Practical Introduction to Computer Vision with OpenCV. [ed.] WILEY. 2014.

Eriş, Halit and Çevik, Ulus. 2019. Implementation of Target Tracking Methods on Images Taken from Unmanned Aerial Vehicles. [ed.] SAMI 2019 • IEEE 17th World Symposium on Applied Machine Intelligence and Informatics. s.l. : IEEE, 2019. pp. 311-316.

Fisher, Ronald. 1936. Iris Data Set. basım yeri bilinmiyor : UCı, Machine Learning Repository, 1936.

Girshick, R., et al. 2014. Rich Feature Hierarchies for Accurate Object Detection and Sementic Segmentation. s.l. : IEEE Conference on Computer Vision and Pattern Recognition, 2014.

Girshick, Ross. 2015. Fast R-CNN. s.l. : IEEE, 2015.

Goodfellow, Ian, Bengio, Yoshua and Courville, Aaron. 2016. Deep Learning. s.l. : MIT Press, 2016. pp. 30-50; 80-224; 330-371; 443-478.

Goodfellow, Ian, Bengion, Yoshua and Courville, Aaron. 2017. Deep Learning. s.l. : MIT Press, 2017. Vol. 521.

Hinton, G., Osindero, S. and Teh, Y. 2006. A Fast Learning Algorithm for Deep Belief. 2006. pp. 1527-1554. Vol. 18.

Kaeppler, Sebastian, et al. 2016. Shading Correction for Grating-based Differential Phase Contrast X-ray Imaging. [ed.] IEEE. s.l. : 2014 IEEE Nuclear Science Symposium and Medical Imaging Conference, NSS/MIC 2014, 2016. pp. 1-3.

Kayalı, Devrim and Çevik, Ulus. 2018. Detection And 3D Modeling Of Brain Tumors Using Image Segmentation Methods And Volume Rendering. [ed.] ISSN:2458-8989. s.l. : NESciences, 2018, 2018. pp. 79-86.

Kennedy, James and Eberhart, Russell. 1995. Particle Swarm Optimization. s.l. : IEEE, 1995.

Kuruvilla, Jiss, et al. 2016. A Review on Image Processing and Image Segmentation. [ed.] SAPIENCE 2016. s.l. : Proceedings of 2016 International Conference on Data Mining and Advanced Computing, 2016. pp. 198-203.

Laencina, Pedro J. Garcia and Sancho-Gomez, Jose Luis. 2007. Multi-task Neural Networks for Dealing with Missing Inputs. 2007.

Lin, Tsung-Yi, et al. 2015. Microsoft COCO: Common Objects in Context. 2015.

McCulloch, W.S and Pitts, W. 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity. [ed.] The University of Chicago Press 1943. s.l. : Kluwer Academic Publishers, 1943.

Mor-Yosef, S., et al. 1990. Ranking the Risk Factors for Cesarean: Logistic Regression Analysis of a Nationwide Study. s.l. : Obstet Gynecol, 1990. pp. 944-947. Vol. 75.

Pulli, Kari, et al. 2012. Real-Time Computer Vision with OpenCV. s.l. : ACM, 2012. Vol. 55.

Redmon, Joseph and Farhadi, Ali. 2018. YOLOv3: An Incremental Improvement. 2018.

Redmon, Joseph. 2016. Darknet: Open Source Neural Networks in C. 2016.

Rumelhart, D., Hinton, G. and Williams, R. 1986. Learning Representations by Back-Propagating Errors. s.l. : Nature, 1986. pp. 533-536.

Russakovsky, Olga, et al. 2015. ImageNet Large Scale Visual Recognition Challenge. s.l. : Springer Science+Business Media New York 2015, 2015.

Shaoqing, Ren, et al. 2016. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. s.l. : IEEE Transactions on Pattern Analysis and Machine Intelligence, 2016. pp. 1137-1149. Vol. 39.

Sokolov, Rostislav I. 2016. Theoretical Investigation of Gaussian and non-Gaussian Noise Masking Properties. [ed.] IEEE. s.l. : 2016 2nd International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), 2016. pp. 1-4.

Szeliski, Richard. 2010. Computer Vision: Algorithms and Applications. London : Springer, 2010.

Yadav, Garima, Maheshwari, Saurabh and Agarwal, Anjali. 2014. Contrast Limited Adaptive Histogram Equalization Based Enhancement For Real Time Video System. s.l. : Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2014, 2014. pp. 2392-2397.

CURRICULUM VITAE

Halit ERİŞ was born in Vienna, Austria in 1993. He received his Bachelor of Science (B.S) degree in Electrical-Electronics Engineering Department from Çukurova University in 2016. After the graduation from his B.S education, he had have worked in “Güney Çelik Hasır ve Demir Mam. San. Tic. A.Ş” for 1 year as an Electrical-Electronical Engineer. He started Master of Science (MSc) education in Electrical-Electronics Engineering Department of Çukurova University. He has been working as a Research Assistant in Electical-Electronics Engineering Department of Çukurova University since 2018. His research areas are Machine Learning, Computer Vision, Image Processing, Digital Design, and Embedded Systems.