Intelligent Traffic Management Systems

AN INTERNSHIP REPORT

SÜLEYMAN CEVDET ÖZBEY

Student No: 16290617

INTEGRATED SYSTEMS & SYSTEM DESIGN

Halıcı Yazılımevi No:33 ZK:2/A ODTÜ Teknokent 06800 Ankara, Türkiye

July 2020

ANKARA

CERTIFICATE

Certified that the internship report ***"Optical Character Recognition"*** and ***"Vehicle Counting with Deep Learning"*** is the bonafide work of "**Süleyman Cevdet ÖZBEY,** Student No:16290617" 4rd Year student of Electrical&Electronics Engineering Department, Ankara University, carried out under my supervision during 01.07.2017 to 14.08.2017

14.08.2020

Betül KARAMAN

Computer Engineer

ABSTRACT

Working in the ISSD at internship was educationally good. Firstly, I want to thank my supervisor Betül KARAMAN: thanks to the way my supervisor has shown in the problem identification and solution, I have succesfully complated my Internship by improving myself.

Image processing has become a part of our lives today, thanks to advancing technologies. We actively use image processing in the field of security, face recognition systems, text reading and many places. During my internship, we did two projects with my supervisor. The first one is optical character identification (OCR) with MATLAB and the second is vehicle counting with deep learning.

OCR is a feature that makes our daily life much easier. Normally, it is very difficult to digitize any text we have. Even the fastest writer can write an average of 500 words per minute, and if we wanted to digitize a long text like a book, it would take a long time. In addition, digitalizing text in a normal way can cause many errors. OCR, on the other hand, can transfer the text we want to transfer into digital media within seconds, and the more data we have, the less chance of making mistakes.

Deep learning has become very popular lately. It has been a method that makes software developers very comfortable in many areas and facilitates their work. It has a place in our lives in many places such as face and voice recognition systems, autopilot alarm systems in vehicles and healthcare sectors. We use deep learning on image processing in the project we do here. We need to determine the number of vehicles in order to determine the intersections with high traffic and adjust the light wave accordingly. With deep learning, we can do this easily.

CONTENTS

İçindekiler Tablosu

LIST OF FİGURES

1. INTRODUCTION

1.1 Who is Integradet System&System Design (ISSD)?

Established in 2009, ISSD operates in system design and integration, digital signal processing, software development and electronic design. The entire process, which starts with the design of the required digital circuits and the development of application-specific algorithms, and ends with the transfer of all kinds of developed software to the embedded platform, can be carried out within ISSD.

ISSD, which continues its activities in its offices in METU Technopolis, produces solutions in traffic management, dynamic intersection control system, license plate recognition and electronic control and develops products that are leaders in the market. Dynamic Intersection Management System developed by our company CHAOS, Turkey is actively used in about 750 locations across. Traffic control systems, alternative data collection systems, traffic simulation software, intersection improvement and road study studies are among ISSD's products and services that make a difference.

2. IMAGE PROCESSING

People need to transfer images to a computer environment to make things easier, or for human eyes to find it difficult to perceive.

To describe image processing simply: It is a method that can be identified with different techniques in order to obtain useful information according to the related needs through the images that have been digitized. The image processing method is used to alienate or improve certain recorded images by manipulating and modifying existing images and graphics. For example, it is possible to see the quality degradation experienced when scanning photos and documents to digital media. It is during this quality reduction that image processing comes into play. We use the image processing method to minimize the degraded image quality and visual distortions. This and many other places that can be used in image processing is among the rapidly developing technologies.

Objectives of image processing:

• Image enhancement

• Image repair

• Image compression

• Image analysis

• Image recognition

So where is image processing used;

- Face recognition and security systems.

- Demographic information analysis.

- Traffic, astronomy, radar and photo industry applications.

- Determination of population density by using application images.

- Radiology field (Tomography, Ultrasound, etc.).

- Understanding of both underwater and satellite images in the field of Military Industry.

- The field of medicine to distinguish various organ structures.

We can apply image processing in 3 steps;

• Transfer an image to a computer: At first, we need to transfer the image to the computer. There are various applications for transferring the image to a computer, such as MATLAB, Phtyon, C ++ and so on. These are applications. In addition, we can transfer real-time images to the computer.

• Analyze the image: After transferring the image to the computer, we can do the operations we want. For example, we can examine the alignment of the face on which we transfer a face picture, or we can identify the model, license plate, color of a car that violates the rule.

• Output: After making the analysis we want from the last image we can print and use. If this is a real-time image, we can quickly process and print it out on the computer.

General concepts in image processing;

Pixel: It is the combination of the words "Picture element, it is the unit element of the image.

Brightness: Shows the brightness value of a pixel in the x and y coordinates.

Discrimination: In order to be able to express the analog image in the digital system, it must first be divided into a finite number of spatial dimensions (sampling, sampling), and then the analog brightness value in each part must be expressed in one of a number of discrete digital levels (quantization, quantization).

Resolution: The number of pixels per inch or cm. (1 inch. = 2.54 cm) Indicates how many pixels the image is divided, ie how many pixels it is represented. The higher the resolution, the higher the frequency the sample is sampled, and the details in the image become clearer.

Spatial Frequencies: Spatial frequencies are the frequency of change in brightness value over a distance.

Input image: is considered a two-dimensional, MxN-length matrix, and the pixel value (1,1) in the upper left corner is numbered as the starting point.

Spatial Frequencies (fx, fy): Typically, the sampling period is assumed to be 1 pixel and spatial frequencies and periods are expressed in pixels.

Image Types:

Binary image; It consists of "0" and "1" s. These values correspond to "black" and "white. 1 pixel, 1-bit footprint

Gray scale; It takes values between "0-255. It contains shades of gray. It gets black as it gets closer to 0 and it gets whitened as it gets closer to 1. 8 pixels are usually reserved for 1 pixel.

Color Image; It is a picture of shades of red, green and blue. 8 bits are allocated for 1 pixel.

2.1. Optical Character Recognition (OCR)

Optical character recognition or optical character reader (OCR) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example: from a television broadcast).

Widely used as a form of data entry from printed paper data records – whether passport documents, invoices, bank statements, computerized receipts, business cards, mail, printouts of static-data, or any suitable documentation – it is a common method of digitizing printed texts so that they can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as cognitive computing, machine translation, (extracted) text-to-speech, key data and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision.

Early versions needed to be trained with images of each character, and worked on one font at a time. Advanced systems capable of producing a high degree of recognition accuracy for most fonts are now common, and with support for a variety of digital image file format inputs. Some systems are capable of reproducing formatted output that closely approximates the original page including images, columns, and other non-textual components.

2.2. Project

This project was done in MATLAB environment. There are many techniques for OCR, and in this project, we do the letter recognition process by correlating the text with the data set we have. The project consists of 4 stages; main part, data set, separating function, clipping function.

- **Main part:** where the image is uploaded and processed. It prints the letters to the txt file after it goes through.
- **Data set:** It is the part where we store the data we have.
- **Line split function:** Separates the written text in the image into lines.
- **Crop function:** Crop the places we want in the Picture.

2.2.1. Main Part:

Before explaining how the main function works, let's talk about some important functions.

Önemli fonksiyonlar:

- **fopen/fprintf/fclose:**

We open a new file with fopen. Its usage is in the form of fopen (filename) or fopen (filename, permission). The opened file is opened to the folder where MATLAB is located, with the name we specified. The permission part specifies the operations we will do on the file. For example;

'r'	Open file for reading.
'w'	Open or create new file for writing. Discard existing contents, if any.

'a'	Open or create new file for writing. Append data to the end of the file.
'r+'	Open file for reading and writing.
'w+'	Open or create new file for reading and writing. Discard existing contents, if any.
'a+'	Open or create new file for reading and writing. Append data to the end of the file.
'A'	Open file for appending without automatic flushing of the current output buffer.
'W'	Open file for writing without automatic flushing of the current output buffer.

fprintf allows us to write text on the file.

fclose closes the file we opened.

- **corr2()**

It correlates 2 matrices and specifies the ratio in double data type. Algorithm;

$$r = \frac{\sum_m \sum_n (A_{mn} - \bar{A})(B_{mn} - \bar{B})}{\sqrt{(\sum_m \sum_n (A_{mn} - \bar{A})^2)(\sum_m \sum_n (B_{mn} - \bar{B})^2)}}$$

Where $\bar{A} = mean2(A)$ and $\bar{B} = mean2(B)$.

- **imfill()**

Fills regions or holes in an image. We can fill the regions we want with the imfill (image, locations) command. It fills all the gaps it detects in the image with the command imfill (image, 'holes'). Below is an example.

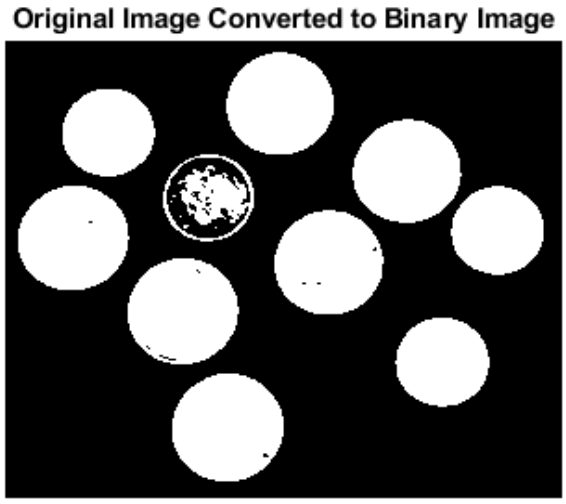

Figure 2.1.1. Original Binary Image

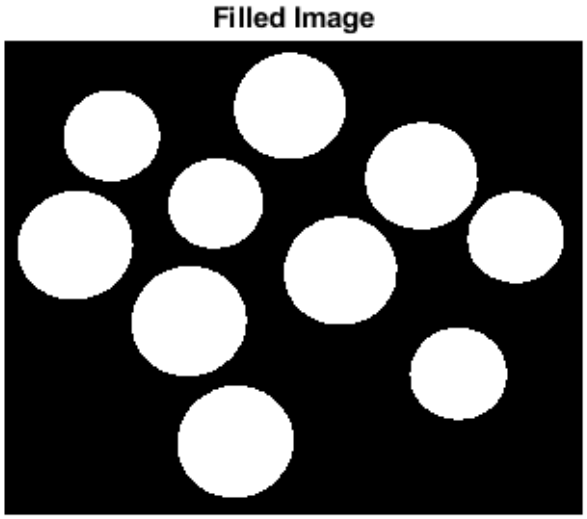

Figure 2.1.2. Filled Image

- **bwperim()**

Finds around objects in binary image. The perimeter of a pixel controls the pixels, if a pixel is worth one and neighbors with at least one pixel of zero value, it is part of the perimeter.

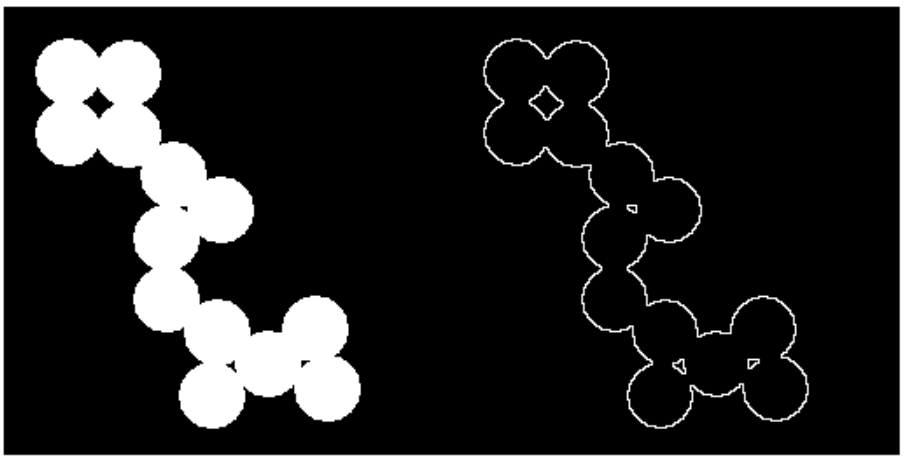

Figure 2.1.3. Around Objects in Binary Image

Expression through code:

```
str={'a', 'b', 'c', 'ç', 'd', 'e', 'f', 'g', 'h', 'k', 'l', 'm', 'n', 'o', 'r', 's', 'ş', 't', 'u', 'v', 'y', 'z' 'B' 'E' 'F' 'P' 'Y'};

word=[ ]; word2=[]; comp=[]; empty=[]; leng=[]; grtleng=[];
```

In this section, we create a variable named "str" and define the letters we will use later in the variable. We will use these letters later to write in the .txt file.
We will explain the usage purposes of other empty variables that we created later.

```
platon=imread('Platon.jpeg');
gray=rgb2gray(platon);
thresh=graythresh(gray);
BW=~im2bw(gray, thresh);
bw=bwareaopen(BW,40);
```

We assign the picture of our article named "Platon.jpeg" to the "platon" variable. Since the picture is in RGB color space, we first convert it to gray space and then to blak-white space. We apply thresholding while converting to black-white space. If we do not apply thresholding, MATLAB automatically takes the value to be converted as 0.5 (white above 122, black below). Thresholding takes the highest and lowest value of the picture in the gray space and calculates its average. For example, in the project I use, the thresholding value is 0.5255. We explained the bwareaopen function earlier. The reason we chose 40 here is to remove noise and punctuation marks.

```
re=bw;
```

We put the size of our image into the re variable.

```
fid=fopen('text.txt', 'wt');
load templates
global templates
```

We create a .txt file named "text" and specify that we will write over it, also add the letter "t" to the w argument to open the file in text mode.

Then we load it into the main function to reach the cell named templates that we created in our data.

After that, we open a while loop and analyze the picture in this loop. The reason we use while is that when we finish the operation on the image, we can return "false" and terminate the loop.

```
[fl, re]=lines(re);
imgn=fl;
cc=bwconncomp(imgn,4);
```

The lines () function takes the information from the re variable and assigns the first row to the fl variable, trimming the remaining part and assigning it back to the variable re. Thanks to the lines () function, we can examine our picture line by line. We will talk about how the function works in future topics.
We throw the information of the fl variable that comes out in the lines () function to the variable named imgn. Then, we apply the bwconncomp function to the imgn variable and pass the information to the cc variable. There are four pieces of information in the cc variable;

- Number of connections
- size of the picture
- Number of objects
- Location of pixels with objects

Here, we open a for loop and rotate the number of objects, we reach the number of objects in the cc variable by typing cc.NumObjects. Thanks to the for loop, we can examine each object separately.

```
grain = false(size(imgn));
grain(cc.PixelIdxList{ob}) = true;
```

We explained the grain variable earlier while describing the bwconncomp function. Thanks to this variable, we can reach each object and examine it.

```
[fr,fc] = find(grain==1);
n1=grain(min(fr):max(fr),min(fc):max(fc));
img_r=imresize(n1,[87 50]);
```

In this section, we bring them to the same size to compare our object with other objects in our data. [fr, fc] save the top, left, bottom and right parts of the selected object and then trim from those parts. Then we resize it to be the same size as the objects in our data.

```
for ii=1:278
   sem=corr2(img_r,templates{1,ii});
   comp=[comp sem];
end
```

We open a new for loop and return it for the number of data in our data. We correlate the object we have with the ones in our data and assign its value into the sem variable. Then we drop this information into our vector called comp.

```
for iii=2:length(comp)
   for jjj=1:iii-1
      if comp(jjj)==comp(iii)
         if comp(jjj)<=0 || comp(iii)<=0
            break
         end
         comp(iii)=-1;
      end
   end
end
```

The purpose of the loop here is to destroy elements of the same value in the comp vector. We'll explain why in the next step.

```
empty=[empty find(comp==max(comp))];
comp=[];
```

Here we find the highest value in the comp vector with the find function and put it into the empty vector. This method does not stop after finding the maximum number, if there is more than one equal maximum number at the same time, it throws all of them into the empty vector. This is the purpose of eliminating other equal values in the previous section.
With these commands, the for loop that we opened to examine each object ends.

```
new_img=imfill(imgn, 'holes');
```

Here we fill in the gaps in all objects in imgn. We'll explain our purpose of doing this in the next step.

```
bwprm=bwperim(new_img);
```

We explained the bwperim command before. bwperim draws the perimeter of the objects. If we do not fill the inside of the object completely with the imfill function, bwperim also draws the inside environment and this is not what we want.

```
stats = struct2table(regionprops(bwprm,'PixelList'));
```

We assign the pixel information of the objects in the picture that we apply bwperim to the stats variable. The reason we do this is to easily measure the distance between two objects. We will explain the purpose of measuring the distance between objects in later stages.

```
for no=1:height(stats)-1
    dist = pdist2(stats.PixelList{no},stats.PixelList{no+1});
    [dmin,idx] = min(dist(:));
    leng=[leng dmin];
end
```
We open a loop the length of the stats variable. Here we measure between both objects. In these measurements, we get the minimum length, the distance between the far right part of the first object and the far left part of the second object. Then we throw these distances into a vector called leng.

```
for number=1:length(leng)
    if leng(number)>11
        grtleng=[grtleng number];
    end
end
```
We open another loop the size of the leng vector. In this loop, we put the values greater than 11 in the leng vector into the vector named grtleng.

After these operations, we open a loop the size of the empty vector. In this loop, we examine the elements in the empty vector respectively. We assign a letter from the srt variable to the word vector according to the value of the element in the empty vector.

```
for num=1:length(word)
    word2=[word2 word(num)];

    for num2=1:length(grtleng)

        if num==grtleng(num2)
        word2=[word2 ' '];
        end
    end
end
```
The purpose of this loop is to make up the space between words. We had previously detected gaps between objects and recorded a certain size of gaps in the grtleng vector. The values in the grtleng vector set the number of letters to leave a space after. We throw all the letters in the word vector to the word2 vector, leaving a space.

```
fprintf(fid,'%s\n',word2);
```
We print the words in the word2 vector to the .txt file and then leave a space.

```
empty=[]; word=[]; leng=[]; grtleng=[]; word2=[];
```
We're resetting our vectors as we'll use them again in the next loop.

```
if isempty(re)
  break
end
```
If the variable re is completely empty, we end the while loop. If not empty, the same actions will occur for the next line..

```
fclose(fid);
winopen('text.txt')
```

We close the .txt file with the fclose command and open the text.txt file in windosw with the winopen command.

2.2.2. Data Set:

It is the function that we hold the data to identify the letters on the picture. We transfer the letter data in our computer to the MATLAB environment via this function. You can see some of the letters in our dataset below;

Figure 2.2.2.1 Some Examples of the Letter a

Figure 2.2.2.2 Some Examples of the Letter f

The examples we see above are taken directly from the texts we use. The reason we get the data in our data set directly from the text we use is to increase the accuracy as much as possible. Below you can see an example of our data set consisting of Calibri letters that we use in the Word environment;

Figure 2.2.2.3 Example of the Letter a

Figure 2.2.2.4 Example of the Letter f

As can be seen in the examples given, although the letters are similar, there are visible differences. Due to these differences, some letters can be misidentified. In the following sections we will show the difference between the two with examples.

Important Function

Mat2cell:

Divides a given array into sub-arrays and keeps those sub-arrays in one cell.

C = mat2cell(A,dim1Dist,...,dimNDist)

Here we see the usage. First we define the array to be divided, then we specify how many vectors they will be and how large they will be.

```
A = reshape(1:20,5,4)'
A = 4×5

     1     2     3     4     5
     6     7     8     9    10
    11    12    13    14    15
    16    17    18    19    20
```

Divide A into two 2-by-3 and two 2-by-2 subarrays. Return the subarrays in a cell array.

```
C = mat2cell(A,[2 2],[3 2])
C=2×2 cell array
    {2x3 double}    {2x2 double}
    {2x3 double}    {2x2 double}
celldisp(C)

C{1,1} =

     1     2     3
     6     7     8

C{2,1} =

    11    12    13
    16    17    18

C{1,2} =

     4     5
     9    10

C{2,2} =

    14    15
    19    20
```

The above example is taken directly from MATLAB's own site. As we can see in the example, we divide a 4x5 matrix into sub-parts and assign it to the C variable. You can see the example visually in the picture below:

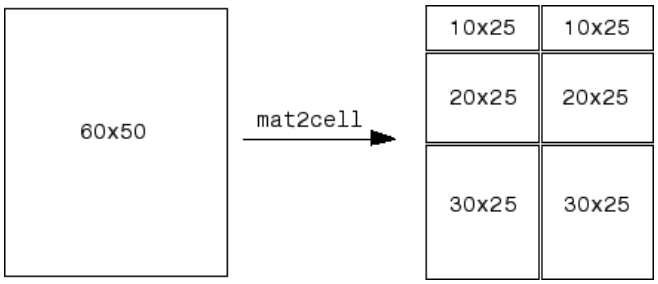

Figure 2.2.2.5 mat2cell Function

Function operation:

First of all, we load the data we have into the MATLAB environment via the imread function. As we mentioned before, the more data we have, the higher our accuracy rate. In the data set we made; There are 31 'a', 5 'b', 1 'B', 1 'c', 1 'ç', 11 'd', 37 'e', 1 'E', 3 "F", 1 "F", 14 "g", 1 "h", 10 "k", 21 "l", 17 "m", 28 n, 7 "o", 1 'P', 18 'r', 15 's', 6 'ş', 10 't', 24 'u', 5 'v', 6 'y', 1 "Y" and 3 "z" letters. The point we need to be careful about here is that all letters should be equal in size. In our example, all letters are 87x50 in size.

We assign the letters we have loaded to arrays by making groups within themselves. For example; We create an array called a_letter for the letters a and assign 31 data a to this array. We do this in the same way for all letters such as b_letter, c_letter… z_letter. Then we collect these strings we created in the array named character. The reason we collect them under a single array is to create a cell with the mat2cell function and thus reduce the processing load by reaching a single cell. After creating a cell named templates, we transfer the array named character into the templates cell and divide the whole array into 87x50 cells. Then we save to use this data.

2.4.3. Lines Funtion:

The purpose of this function is to split the text in the image we uploaded into lines. The reason we divide the text into lines is for easier processing on the text. When the text comes to this function, it separates the first line and the rest. It returns the first row for us to process, the remaining part is clipped again, and when we call the function again, this time it returns our second row.

Function Operation:

```
function [fl, re]=lines(im_texto)
```

In the main function, we assigned the size of the picture to the variable re. The other variable fl is completely empty. We will assign the first row to this variable later. im_texto is our picture.

```
im_texto=clip(im_texto);
num_filas=size(im_texto,1);
```

First, we clip the image in the im_texto variable with the clip function, we will explain how the clip function works in the following stages. Then we create a variable named num_filas and assign the horizontal length of the image to this variable.

```
for s=1:num_filas
   if sum(im_texto(s,:))==0
      nm=im_texto(1:s-1, :);
      rm=im_texto(s:end, :);
      fl = clip(nm);
      re=clip(rm);
      break
   else
      fl=im_texto;
      re=[ ];
   end
end
```

We return the size of the variable num_filas in our loop. Since our image is cropped, the first element always starts with a 1 and then goes down one by one after that. If it encounters 0, it

assigns the remaining part to the variable fl and the remaining part to the variable re in a cropped form.

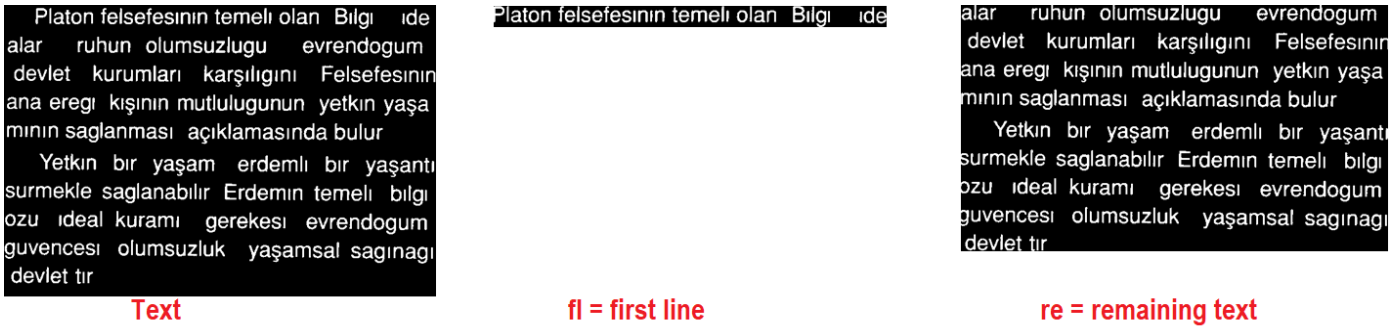

Figure 2.2.3 Example of Lines Funtion

2.2.4. Clip fonksiyonu:

Clip function helps to crop the picture we have.

```
function img_out=clip(img_in)
[f c]=find(img_in);
img_out=img_in(min(f):max(f),min(c):max(c));
```

It takes the image that comes to the function and assigns it to the img_in variable. It assigns the position of the places with "1" on the picture to variables f and c. It then crops from the leftmost 1 to the rightmost 1, from the highest 1 to the lowest 1 and assigns it to the img_out variable. Below is an example.

Figure 2.2.4.1 Example of Clip Function

2.2.5. The Output of Project

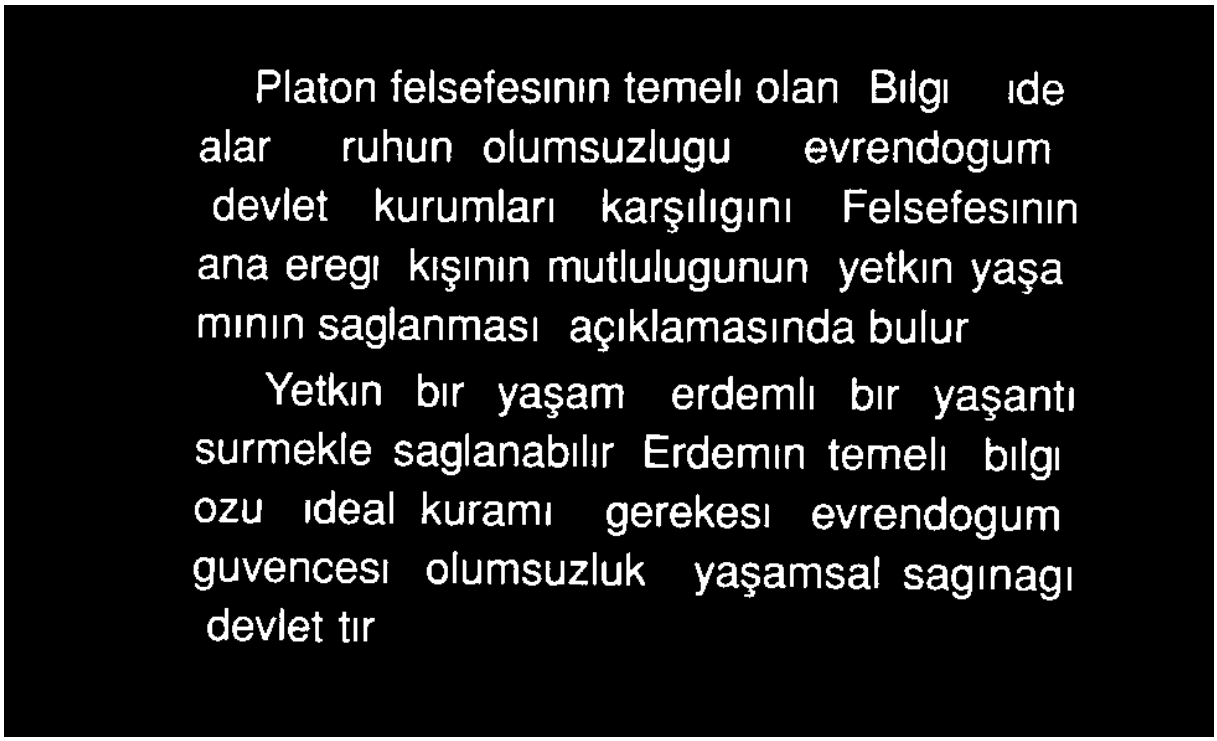

Figure 2.2.1 Text

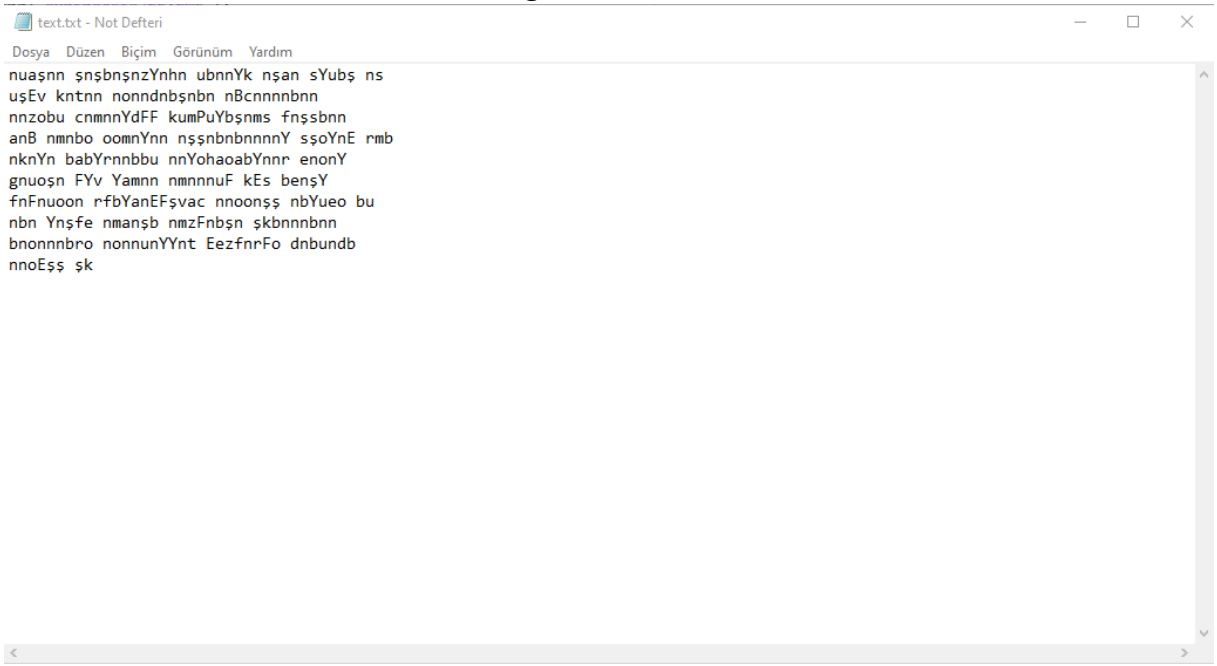

Figure 2.2.2 The Output with a Low Data

Figure 2.2.3 The Output with a High Data

When we examine the above two results, we understand how important the excess of the data we have is. There is only one sample of each letter in the first printout and we almost did not get an accurate result. As we saw in the second output, we have reached a more accurate conclusion with the more data we have.

3. Vehicle Counting

In this project, we use object identification, tracking and counting processes, which are part of image processing. In a traffic image, we focus only on vehicles and try to follow them and count them. The purposes of determination and counting of these vehicles are as follows;

• Detecting intensity at instant intersections

• Calculation of rush hour in traffic

• Scheme of the traffic load

• Daily density of vehicles passing a certain road

There are more than one method of vehicle detection and in this project, it is done by using deep learning with python.

3.1. Python:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

3.2. Artificial Intelligence (AI):

Artificial intelligence was born in the 1950s, when a handful of pioneers from the nascent field of computer science started asking whether computers could be made to “think” -a question whose ramifications we’re still exploring today. A concise definition of the field would be as follows: the effort to automate intellectual tasks normally performed by humans. As such, AI is a general field that encompasses machine learning and deep learning, but that also includes many more approaches that don’t involve any learning. Early chess programs, for instance, only involved hardcoded rules crafted by programmers, and didn’t qualify as machine learning. For a fairly long time, many experts believed that human-level artificial intelligence could be achieved by having programmers handcraft a sufficiently large set of explicit rules for manipulating knowledge. This approach is known as symbolic AI, and it was the dominant paradigm in AI from the 1950s to the late 1980s. It reached its peak popularity during the expert systems boom of the 1980s.

Although symbolic AI proved suitable to solve well-defined, logical problems, such as playing chess, it turned out to be intractable to figure out explicit rules for solving more complex, fuzzy problems, such as image classification, speech recognition, and language translation. A new approach arose to take symbolic AI's place: machine learning.

3.3. Machine Learning:

A machine-learning system is trained rather than explicitly programmed. It's presented with many examples relevant to a task, and it finds statistical structure in these examples that eventually allows the system to come up with rules for automating the task. For instance, if you wished to automate the task of tagging your vacation pictures, you could present a machine-learning system with many examples of pictures already tagged by humans, and the system would learn statistical rules for associating specific pictures to specific tags.

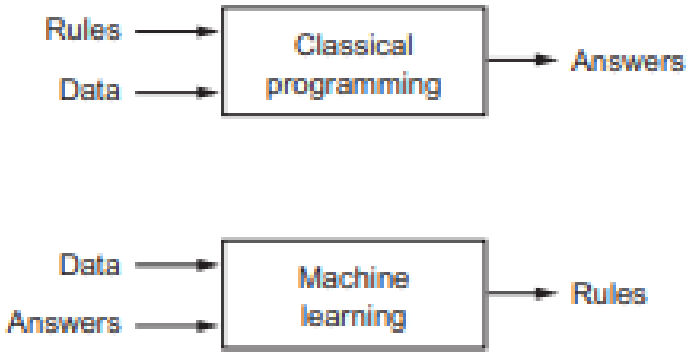

Figure 3.3.1 The Diagram of Machine Learning

3.4. Deep Learning

Computer vision as a field has a long history. With the emergence of deep learning, computer vision has proven to be useful for various applications. Deep learning is a collection of techniques from artificial neural network (ANN), which is a branch of machine learning. ANNs are modelled on the human brain; there are nodes linked to each other that pass information to each other.

How Deep Learning Works:

Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks.

The term "deep" usually refers to the number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150.

Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual feature extraction.

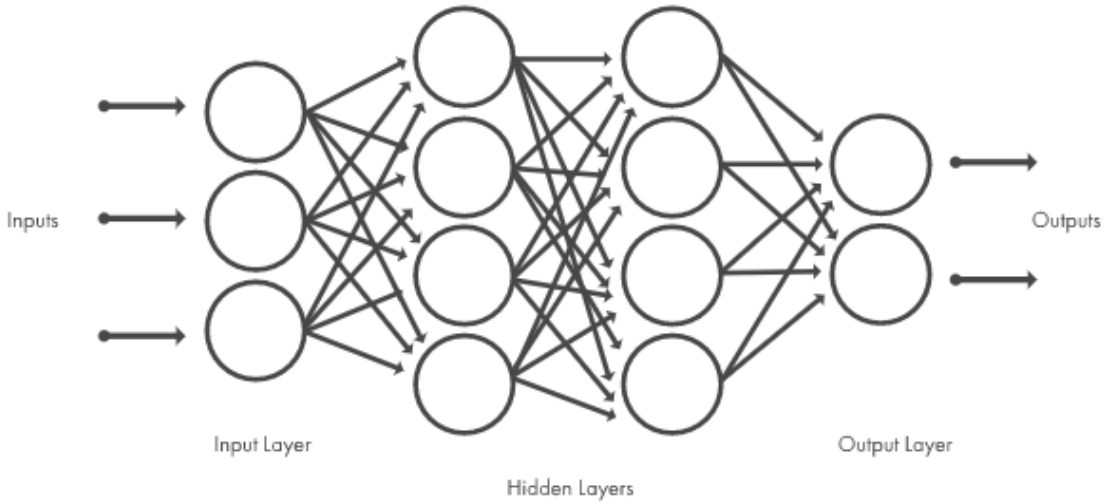

Figure 3.4.1. Neural Networks

We can use many methods using these neural networks, some of these methods are; cnn, ssd and yolo algorithms.

3.4.1. Convolutional Neural Networks (CNN):

It is a special type of linear process that applies convolution to neural networks. Since it is a linear process, neurons in the same layer do not share any connections. If we use normal neural networks for images, their size will be enormous because we need to use so many neurons. An image can be considered a volume with dimensions of height, width, and depth. Depth is the channel of an image, which is red, blue, and green. The neurons of a CNN are arranged in a volumetric fashion to take advantage of the volume. Each of the layers transforms the input volume to an output volume as shown in the following image:

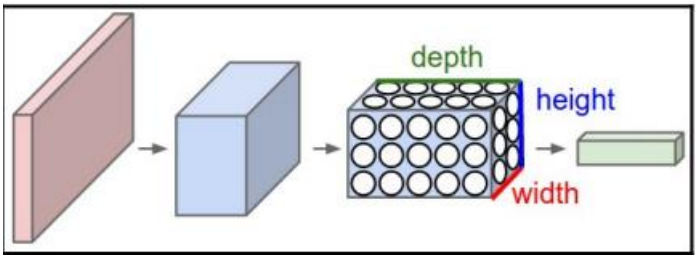

Figure 3.4.1.1 CNN Diagram

3.4.2. Recurrent Neural Networks (RNN):

Recurrent neural networks (RNN) can model sequential information. They do not assume that the data points are intensive. They perform the same task from the output of the previous data of a series of sequence data. This can also be thought of as memory. RNN cannot remember from longer sequences or time. It is unfolded during the training process, as shown in the following image:

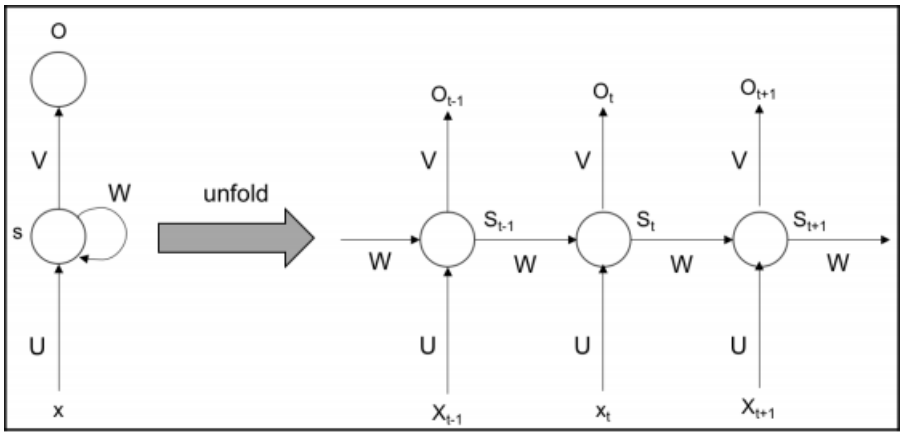

Figure 3.4.2.1 RNN Model

As shown in the preceding figure, the step is unfolded and trained each time. During backpropagation, the gradients can vanish over time. To overcome this problem, Long short-term memory can be used to remember over a longer time period.

3.4.3. Single Shot MultiBox Detector (SSD):

SSD is designed for object detection in real time. While it is considered the start-of-the-art in accuracy, the whole process runs at 7 frames per second. Far below what a real-time processing needs. SSD speeds up the process by eliminating the need of the region proposal network. To recover the drop in accuracy, SSD applies a few improvements including multi-scale features and default boxes. These improvements allow SSD to match the Faster R-CNN's accuracy using lower resolution images, which further pushes the speed higher. The SSD object detection composes of 2 parts:

- Extract feature maps, and
- Apply convolution filters to detect objects.

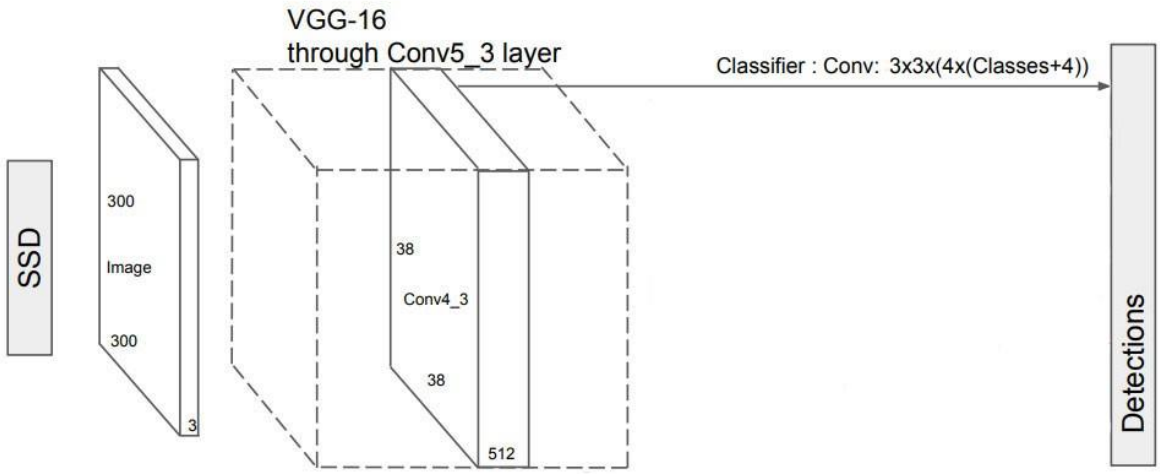

Figure 3.4.3.1 The Diagram of SSD

3.4.4. You Look Only Once (YOLO):

The biggest feature that distinguishes YOLO from other algorithms is real-time object detection. Except for YOLO, there were algorithms for real-time object detection, but the overall average precision (mAP) values were low.

Algorithms like CNN used region-based object detection. Although this method gave good results, there were two operations in one picture, causing them to be slow. The reason why the YOLO algorithm is so fast is that it can detect the class and coordinates of objects by passing the picture through the neural network in one go. To do this, it divides the picture into square grids, the size of these grids can be 3x3, 5x5, 19x19.

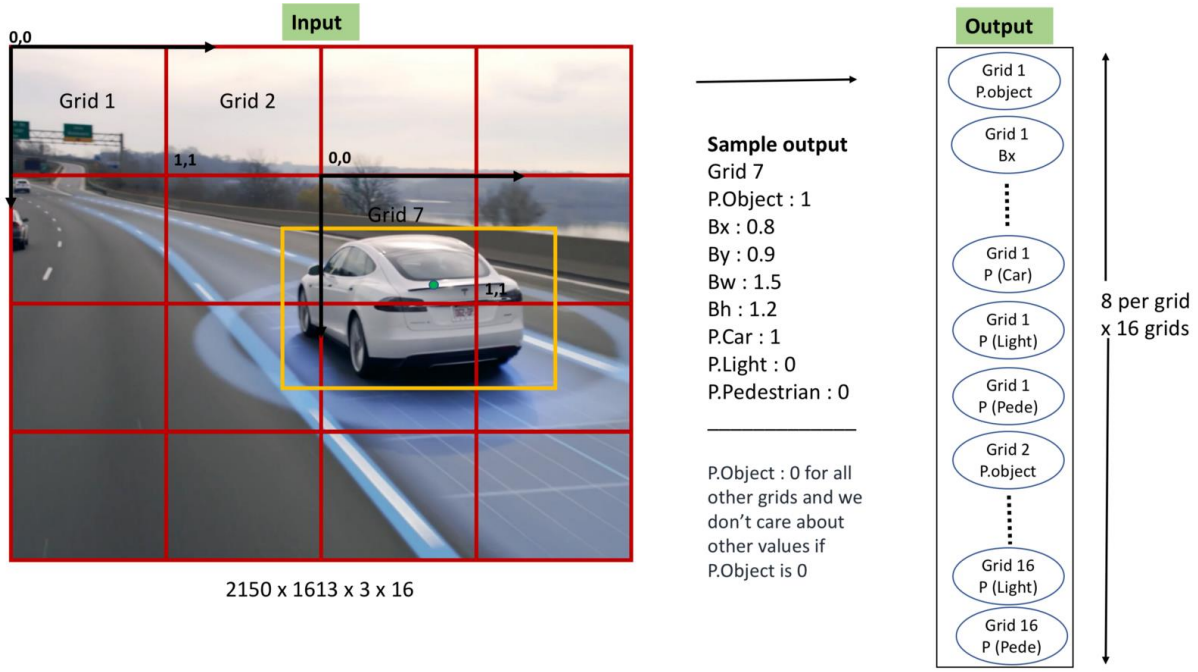

Figure 3.4.4.1 Sample YOLO Algorithm Output

Each grid in itself is responsible for finding out whether the object is in the field, if it is in its midpoint, if it is within its midpoint, its length, height and which class it belongs to. To put it more clearly, for example, in the above picture, since the middle point of the car corresponds to the 7th grid, that grid is responsible for detecting / drawing boxes around the car. Accordingly, YOLO creates a separate prediction vector for each grid. In each of these:

Güven skoru: The confidence score indicates whether there are any objects in the grid. If the score is zero, it does not exist, and if it is one, it certainly exists. If there is an object, it looks at which object it is and the coordinates of the box around it.

- **Bx:** The x coordinate of the object's midpoint
- **By:** Y coordinate of the object's midpoint
- **Bw:** Object width
- **Bh:** Object height
- **Bağlı sınıf Olasılığı**: It contains the predictive value of how many different classes we have. When we look at the 7th grille in the picture above, we are sure that it is a car; Car = 1, Pedestrian = 0, Bicycle = 0.
- The formula for the trust score is; Box Trust Score x Linked Class Probability
- **Box Trust Score =** P(nesne) . IoU
- **P(**object**) =** The probability that the box covers the object. (So is there an object or not?)
- **IoU =** IoU value between ground truth and estimated box

For the background portion with no objects, the confidence score will be zero because the bound class probability value is zero.

Up until this part we had a general idea of how YOLO predicts for each grid. But while the algorithm is running, unnecessary boxes may appear. Assuming there is more than one grid center point for the same object, a problem will arise as follows;

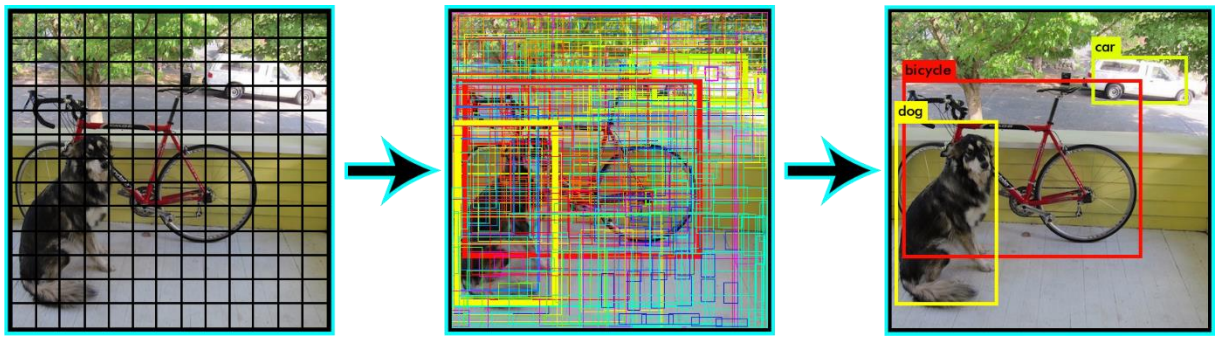

Figure 3.4.4.2 Sample of a Lot of Grid

1. This is where the Nonmax Suppression algorithm comes into play. Let's look at the algorithm if you want:

2. Skip all boxes with a confidence score below a certain level (eg 0.5)

   As long as the box remains:

   I. Select the box with the highest confidence score and output it. Let's call this box A.

   II. Discard A with all other boxes with IoU value greater than 0.5

As a result of this process, we have only one box for each object.

3.5. Project:

This project was made with the YOLO model. The important libraries used are OpenCV and numpy libraries.

OpenCV: Open Source Computer Vision Library is an open source library that runs on OpenCV Linux, Windows, Mac OS X, PSP operating systems, and most of its functions are platform independent. The areas where it is used are frequently used in image processing algorithms such as face recognition, sign language recognition, motion capture. In 1999, intel firm released its first version. SourceForge continues to be developed. Development can be done with C and C

++ programming languages. OpenCv is an image manipulation library and platform independent. *

NumPy (Numerical Python): It is a math library that allows us to do scientific calculations quickly. Numpy is based on numpy arrays. Numpy arrays are similar to python lists, but are more useful than python lists in terms of speed and functionality. Also, unlike python lists, Numpy arrays must be homogeneous, so all elements in the array must be of the same data type.

Now we can examine the codes in the project;

```
INPUT_FILE='arabakısa2.mp4'

OUTPUT_FILE='output.avi'

LABELS_FILE='coco.names'

CONFIG_FILE='yolov3.cfg'

WEIGHTS_FILE='yolov3.weights'

CONFIDENCE_THRESHOLD=0.7

H=None

W=None

fps = FPS().start()

fourcc = cv2.VideoWriter_fourcc(*"MJPG")

writer = cv2.VideoWriter(OUTPUT_FILE, fourcc, 30,

        (800, 600), True)

LABELS = open(LABELS_FILE).read().strip().split("\n")
```

```
np.random.seed(4)

COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),

        dtype="uint8")

net = cv2.dnn.readNetFromDarknet(CONFIG_FILE, WEIGHTS_FILE)

vs = cv2.VideoCapture(INPUT_FILE)
```

In this section, we first transfer our video and YOLO files to the python environment. We use our data set from COCO, the data set prepared by YOLO.

With VideoWriter, we determine the name, type, fps number and size of the output file. True, the encoder expect color frame, otherwise it works with grayscale frame.

Reads a network model stored in Darknet model files.

```
ln = net.getLayerNames()

ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]

cnt =0;

carr=0;
```

Gives a list of all layers used in a network, followed by the number of the last layers. After that, we open the while loop to examine the video screen by screen. Thanks to the loop we open, we can view the video as a normal single image.

```
   cnt+=1

   print ("Frame number", cnt)

   try:

      (grabbed, image) = vs.read()

   except:

         break

   ROI=image[0:800, 175:600]

   blob = cv2.dnn.blobFromImage(ROI, 1 / 255.0, (416, 416),

                        swapRB=True, crop=False)

   net.setInput(blob)

   if W is None or H is None:

      (H, W) = ROI.shape[:2]
```

```
layerOutputs = net.forward(ln)
```

In this section, we determine the area we will examine. After determining the area we will examine, we resize the image and layer it, and at the end of the process, we determine whether the resized image will return to its old size.

```
boxes = []
confidences = []
classIDs = []
for output in layerOutputs:
    for detection in output:
        scores = detection[5:]
        classID = np.argmax(scores)
        confidence = scores[classID]
```

In this section, we determine the class of objects in the score table within the specified layers.

```
if confidence > CONFIDENCE_THRESHOLD:
            box = detection[0:4] * np.array([W, H, W, H])
            (centerX, centerY, width, height) = box.astype("int")
```

If our score value is higher than the value we set, we determine the x and y coordinates of the specified object.

```
            x = int(centerX - (width / 2))
            y = int(centerY - (height / 2))
            boxes.append([x, y, int(width), int(height)])
            confidences.append(float(confidence))
            classIDs.append(classID)
```

Based on the coordinates we have determined, we put the object in a box.

```
idxs = cv2.dnn.NMSBoxes(boxes, confidences, CONFIDENCE_THRESHOLD,
                CONFIDENCE_THRESHOLD)
```

We assign the value and class number of the object inside the box to the variable idxs.

```
if len(idxs) > 0:
    for i in idxs.flatten():
        if classIDs[i]==2 or classIDs[i]==5 or classIDs[i]==7:
            carr=carr+1
```

If our class number corresponds to 2 or 5 or 7, we indicate it as a car. These number numbers are the numbers that correspond to the car in the COCO data set.

```
            (x, y) = (boxes[i][0], boxes[i][1])
            (w, h) = (boxes[i][2], boxes[i][3])
            color = [int(c) for c in COLORS[classIDs[i]]]
            cv2.rectangle(ROI, (x, y), (x + w, y + h), color, 2)
            text = "{}: {:.2f}".format(LABELS[classIDs[i]], confidences[i])
            cv2.putText(ROI, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
                    0.5, color, 2)
        else:
            break
    text2="{} = {}".format("car number", carr)
    cv2.putText(image, text2, (125,35), cv2.FONT_HERSHEY_SIMPLEX,
            0.6, (0,0,255), 2)
```

We use the algorithm above to write the number of vehicles we have detected on the picture. This is how we decide where the picture will be, how large, in what color, what type of font.

```
    cv2.imshow("output", cv2.resize(image,(800, 600)))
    carr=0;
    writer.write(cv2.resize(image,(800, 600)))
    fps.update()
    key = cv2.waitKey(1) & 0xFF
    if key == ord("q"):
        break
```

We resize the final picture as we want it and recombine each picture examined. This continues until the last screen and we end the loop. Then we close all the windows we have opened and run the video. You can see examples of the outputs we have received below;

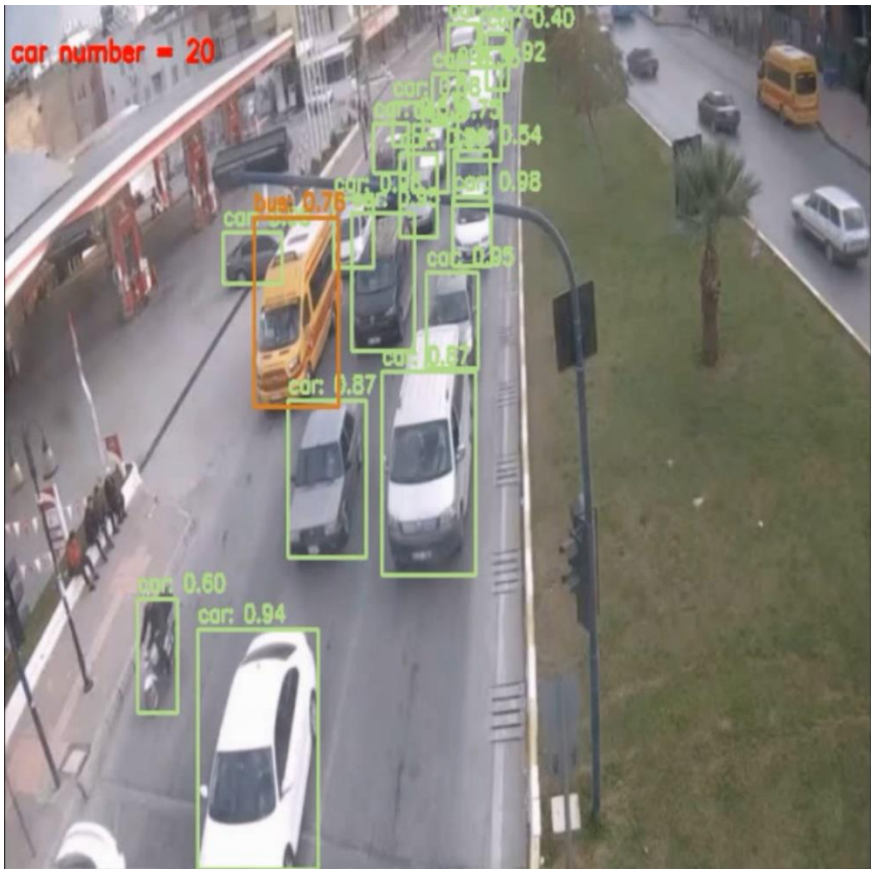

Figure 3.5.1 Sample of the Output

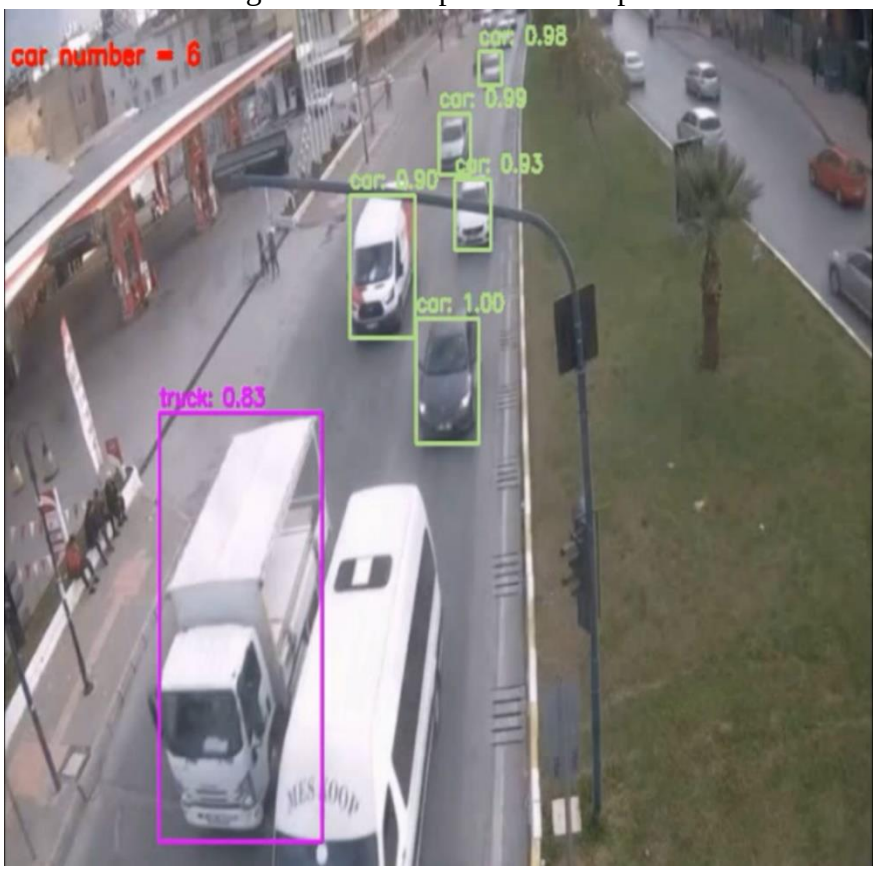

Figure 3.5.2 Sample of the Output

Figure 3.5.3 Sample of the Output

4. CONCLUSION

During my internship, I had the opportunity to learn a lot about image processing and deep learning. Using different languages for the two projects was a good experience. I learned what the changes that may occur instantly in business life are like and what I have to do against it.

When I was doing both projects, my supervisor helped me a lot and whenever I asked a question, he immediately responded. I realized how important the data set is in the OCR project. If we have little data, there are too many erroneous results. At first, I had a hard time doing the project, but I enjoyed it as I progressed and reached solutions. It was a nice experience for me to understand the working principles of the applications we see outside and to make them myself.

When it was about counting vehicles in my graduation project, I was more interested in the second project. We did it using MATLAB in the graduation project, but this time with the advice of my supervisor, I did this project with deep learning. Deep learning was a very broad subject and I still haven't been able to fully learn it no matter how many projects I have done. I want to do many more studies on deep learning and learn the subject better.

During my internship, I learned about the relationships with our superiors in business life, how to do the division of labor and how to make algorithms in software life. Using two languages gave me different experiences.

5. REFERENCES

I. Deep Learning with Python, François Chollet

II. Deep Learning for Computer Vision, Dr. Stephen Moore

III. https://en.wikipedia.org/wiki/Optical_character_recognition

IV. https://medium.com/bili%C5%9Fim-hareketi/veri-bilimi-i%CC%87%C3%A7in-temel-python-k%C3%BCt%C3%BCphaneleri-1-numpy-750429a0d8e5#:~:text=NumPy%20(Numerical%20Python)%20bilimsel%20hesaplamalar%C4%B1,yapmam%C4%B1z%C4%B1%20sa%C4%9Flayan%20bir%20matematik%20k%C3%BCt%C3%BCphanesidir.

V. https://www.pythontr.com/makale/opencv-nedir-271

VI. https://www.python.org/doc/essays/blurb/

VII. https://medium.com/@anyline_io/what-is-ocr-why-does-it-make-your-life-easier-209b9fcedec4

VIII. https://www.mathworks.com/