جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

المدرسة الوطنية العليا للفنون و المهن
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS

Project report:

ADAS system with PPP functionality

Realized by:

THABIT Ayoub

TAOURARTI Imane

Supervised by:

Pr. MASROUR Tawfik

Pr. HAJJI

05/07/2020

جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

المدرسة الوطنية العليا للفنون و المهن
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS

Acknowledgment

We would like to express our deep gratitude to Professor **Masrour Tawfik** and Professor HAJJI, our project supervisors, for their patient guidance, enthusiastic encouragement and useful critiques of this research and technical work. We would also like to thank Professor HASSANI Ibtissam, for her advice and assistance in keeping our progress on schedule. We have taken efforts in this project. However, it would not have been possible without the kind support and help of the mentioned Professors, especially in this period that know the outbreak of a major pandemic in the whole world. We would like to extend our sincere thanks to all of them.

Finally, we wish to thank our parents for their support and encouragement throughout our studies.

Abstract

Vision-based driver assistance systems are designed, and implemented in modern vehicles, for improving safety and better comfort. This report reviews areas of research on vision-based driver assistance systems and covers our work on systems of object detection and tracking of cars and pedestrian with much more other optional functionalities, using YoloV3, Keras, OpenCV... As we're in the first phase of a big project, this report also provides an extensive bibliography for the discussed subjects

Introduction

Most road accidents occur due to human error; Advanced driver-assistance systems are systems developed to automate, adapt, and enhance vehicle systems for safety and better driving. The automated system which is provided by ADAS to the vehicle is proven to reduce road fatalities, by minimizing human error. Safety features are designed to avoid collisions and accidents by offering technologies that alert the driver to potential problems or to avoid collisions by implementing safeguards and taking over control of the vehicle.

Assignment and Project Approach:

It is in this context that we are working in this business project on the PPP functionality. the project is in its infancy, and it already leads us to master many essential concepts. To take the first steps towards artificial intelligence and in turn, we are not just trying to implement some algorithms and get some results. Instead, we are trying to write a report that would be used as a reference for our fellow students. A reference that groups theory and practice. We will try to make things look as easy as possible.

Report Structure:

This report is divided into five parts.

- Part 1: this part will be dedicated to present specifications and goals of this project.
- Part 2: We will present our learning plan that we stuck to during the first period of this project.
  - Chapter 1: will present the major courses we followed and that we assume will be essential for everyone wanting to continue working on this big project.
  - Chapter 2: contains the most important notion which is convolutional neural networks.
  - Chapter 3: will present a quick overview about ADAS.
- Part 3: will contain a benchmarking of different possible solutions and reasons why we chose the actual solution for our project.
- Part 4: will contain the actual scripts that we're using and try to explain each part in detail. this part contains the script of object detector using yoloV3 along with the functionality of determining the distance from our car and their directions. Then we will add plate numbers detection, and finally, we will present curved lane detection scripts and how it works.
- Part 5: will be dedicated to applying our work on the scenarios generated by our colleges on Carla.

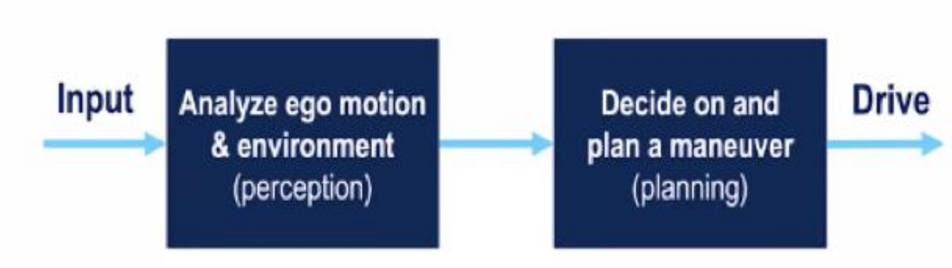
Finally, a general conclusion will wrap our work.

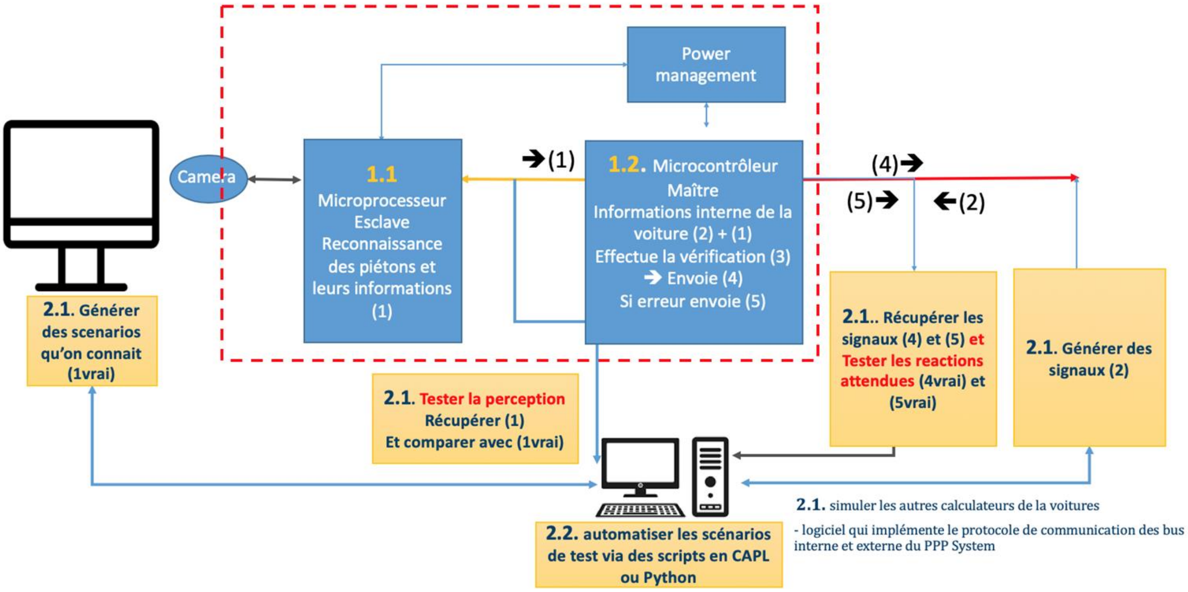
List of figures:

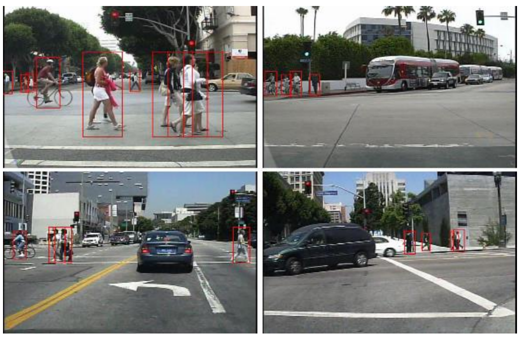
Content table

I. Specifications on the PPP system:

Our goal for this project is to make a system that take the input information from the camera and analyze the environment, putting the first and most important bloc of the perception functionality of ADAS, which is the first P of the PPP system these results that we have to generate will be combined with ego motion analysis to create and input to the future made bloc “Planning” which will have to decide and plan based on our result a maneuver and a make a way to its destination. Then control the hardware to apply instructions from the PPP system.







The system we’re making is an object detector which detect objects on the road and sideways, especially pedestrian and other cars as well as measuring the distance between those objects and the ego car.

Finally, we added some additional functions that we judged may be interesting for future extensions.

II. A project where learning is the ultimate goal:

1. Major courses we enrolled in:

From the get-go, we realized that this project required essential knowledge in various fields of machine learning as deep learning, convolutional neural network, and machine vision. So, we started searching for the most requested courses in this field. In the next pages, we will cite the most important and basic courses we enrolled.

P.S: We should note that we didn't fully complete these courses, instead, we took what's essential for our project

A. *Machine Learning from Andrew Ng.*

One of the top recommendations was the Coursera "Machine Learning" course, from Stanford University presented by the co-founder of Coursera Professor Andrew Ng. We got to apply some of the most advanced machine learning algorithms to such problems as anti-spam, image recognition, clustering, and many other problems. For this course, we used Octave software which, generally, helps to solve linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with MATLAB. It covered linear regression with one or multiple variables, presenting notions of a cost function, gradient descent, regularization, overfitting, and underfitting problems. After these essential notions, we learned how to train neural networks. For that purpose, the course explained in detail backpropagation, bias, and variance. Then, we moved on to support vector machine algorithms as they are the most powerful black box learning algorithms.

B. *Deep Learning Specialization*

This specialization course was created by professor Andrew Ng, Teaching Assistant Younes Bensouda Mourri, and Head Teaching Assistant - Kian Katanforoosh. Being composed of 5 courses, it helped us learn the foundations of deep learning, understand how to build neural networks, and learn how to lead successful machine learning projects. You will learn about Convolutional networks and RNNs.

Introduction to Deep learning:

In the first course, we:

- ✓ Understood the major technology trends driving Deep Learning
- ✓ were able to build, train and apply fully connected deep neural networks - Knew how to implement efficient (vectorized) neural networks
- ✓ Understood the key parameters in a neural network's architecture.

This course also taught us how Deep Learning works, rather than presenting only a cursory or surface-level description. So, after completing it we were able to apply deep learning to our applications.

Convolutional Neural Network:

On the other side, the fourth course entitled ***" Convolutional Neural Network"*** was a chance to learn how to build convolutional neural networks and apply them to image data. Thanks to this course we were able to:

- ✓ Understand how to build a convolutional neural network, including recent variations such as residual networks.
- ✓ Know how to apply convolutional networks to visual detection and recognition tasks.
- ✓ Know to use neural style transfer to generate art.
- ✓ Be able to apply these algorithms to a variety of images, videos, and other 2D or 3D data.

C. *Self-Driving cars Specialization:*

One more essential course we had to enroll in is "self-driving cars" specialization from the University of Toronto it contained four courses from which we followed two primordial plans.

Introduction to self-Driving cars:

The parts we enrolled in this course introduced us to the terminology, design considerations, and safety assessment of self-driving cars. Thanks to this course, we were able to:

- ✓ Understand commonly used hardware used for self-driving cars

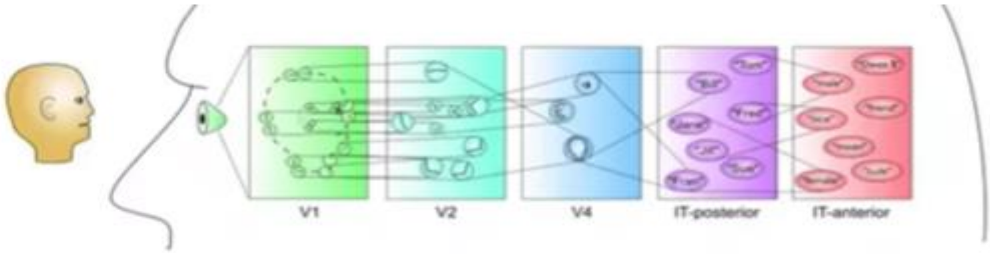
- ✓ Identify the main components of the self-driving software stack
- ✓ Program vehicle modeling and control
- ✓ Analyze the safety frameworks and current industry practices for vehicle development.

Visual Perception for self-driving cars:

the first weeks of this course introduced us to the main perception tasks in autonomous driving, static and dynamic object detection, and surveyed common computer vision methods for robotic perception. we looked into visual odometry, object detection and tracking, and semantic segmentation for drivable surface estimation. These techniques represent the main building blocks of the perception system for self-driving cars.

2. Convolutional Neural Network:

In this chapter, we choose to dissect in detail the convolutional neural networks. CNN is the basis of the algorithm chosen for the detection of pedestrians and the classification of images in general. it is therefore essential to understand the principle in order to be able to choose its parameters and easily use the models already developed.



Nature is the key! Mammals visually perceive the world around them using a layered architecture of neurons in the brain. Within the visual cortex, complex functional responses generated by “complex cells” are constructed from more simplistic responses from “simple cells” that would respond to oriented edges, while complex cells will also respond to oriented edges but with a degree of spatial invariance.

The architecture of deep convolutional neural networks is inspired then by:

- Local connections
- Layering
- Spatial invariance; we learn abstraction.

The computer sees an input image as an array of pixels. Based on the image resolution, it will see h*w*d

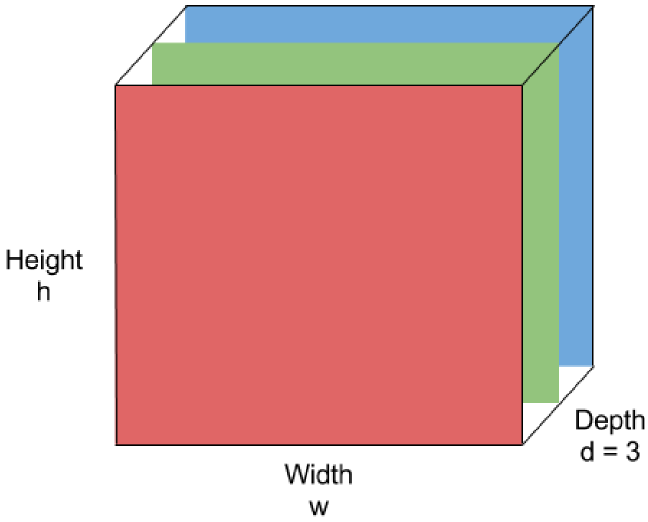


Figure 1: The input image

A. Convolution layer

Convolution is a first layer to extract features from an input image; edge detection, blur, and sharpen.

It's a mathematical operation that takes two inputs:

-An image matrix (h * w * d)

-A filter (fh * fw * d)

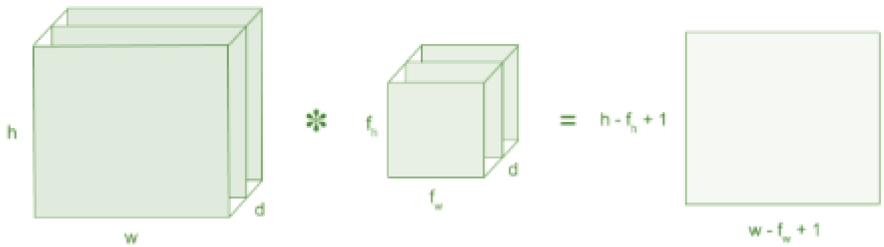


Figure 2: Image matrix multiplies filter matrix

It preserves locality; The information that makes up a human includes the relative position of its eyes, nose, and mouth; A single number in the output is representative of when the filter is at the top left of the image.

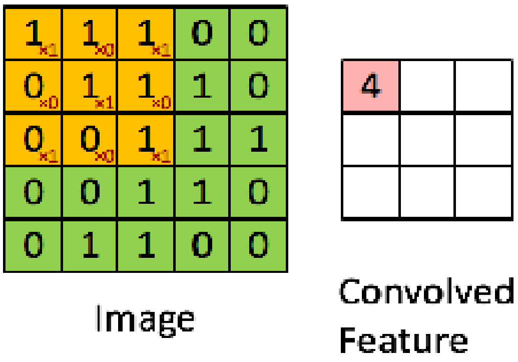


Figure 3: The convolution operation with a stride of 1.

It reduces the number of parameters we need to learn; it captures the local dependencies in the original image; We can say that it's gathering and processing the information on one part of the image and sum them up into a single, meaningful value.

B. Pooling layer

After the convolutional operation, we perform the pooling on the output to reduce the size but still retaining the most important information. It can be a Max, Average, Sum pooling.

We are always looking for operations that reduce computation and still keep the core information. Pooling makes the network invariant to small transformations in the input

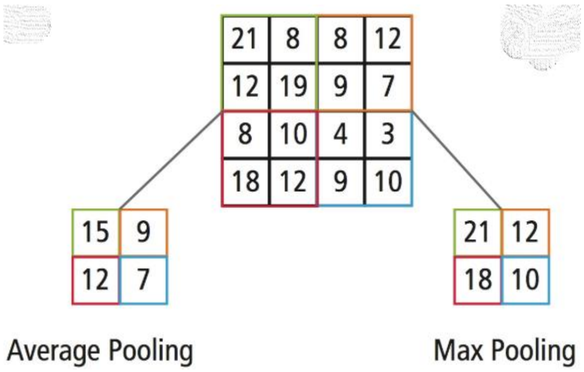


Figure 4: Max and Average pooling

C. Fully connected layer

The output from the convolutional and pooling layers represents high-level features of the input image. At this level, we need to use these features to classify the input image into various classes. The fully-connected layer is a cheap way of learning non-linear combinations of these features.

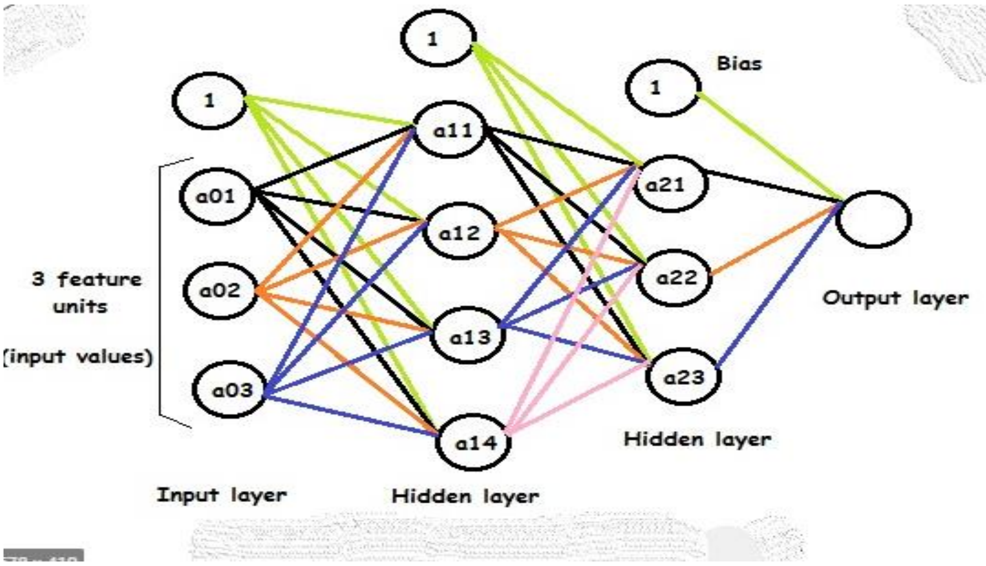


Figure 5: A neural network with fully connected layers.

For the deep neural network, we chose to append the complete code after having followed a course in Coursera. It highlights the activation function, the forward and backward propagation and many other algorithms.

❖ ***Conclusion:***

In this chapter, we discussed CNN with all the layers that make it up. However, Python has many libraries with which it is enough to design the architecture of the network without really worrying about the theory behind it.

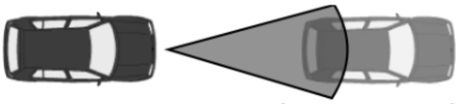
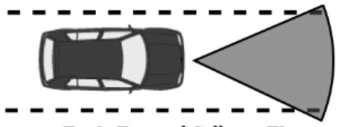
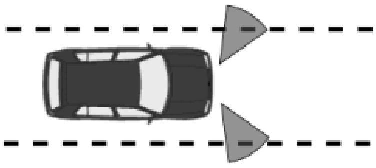
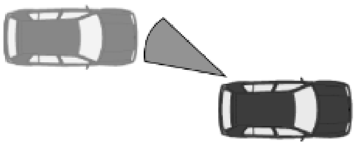
In a model for the calculation of the distance separating the vehicle from a pedestrian, we will process the image with KERAS. In that chapter, we will study in detail the code used and how to combine theory with practice.

3. Advanced drive-assistance systems:

Advanced Driver Assistance Systems are intelligent systems that reside inside the vehicle and assist the main driver in a variety of ways. These systems may be used to provide vital information about traffic, closure and blockage of roads ahead, congestion levels, suggested routes to avoid congestion etc.

A. ADAS TECHNOLOGY OVERVIEW

To give an impression of what ADAS means to end users, an overview of existing ADAS technology is presented. For convenience, they’ve been divided into subcategories. This short overview of existing ADAS technology only highlights the more 'common' types of ADAS

	ADAS		*Description*
Longitudinal	ACC	Adaptive Cruise Control	ACC is becoming a more and more common accessory in modern cars. Basically, this technology keeps a safe distance between the driver's car and vehicles ahead. The driver can adjust the distance, and the system makes sure it's maintained, using throttle and brake control. Most ACC systems have influence on the driving task (they control brake and throttle), but still allow user take-overs.  *Fig 1: Adaptive Cruise Control*
	FCW	Forward Collision Warning	Like the ACC, this system detects vehicles in front of the driver's car. Obviously, it can be integrated with ACC. However, current systems still have problems distinguishing cars from trees, bridges from road signs, etc.  *Fig 2: Forward Collision Warning*
	ISA	Intelligent Speed Assistance	ISA influences the speed at which a car is driving. The maximum speed can be pre-set, or acquired from GPS data. Interfacing with the driver is done via the acceleration pedal, or by using visual or audio warnings.
Lateral Support	LDW	Lane Departure Warning	The main task of Lane Departure Warning is to make sure a car is driving safely between road marks (i.e. in a lane). LDW uses cameras and computer systems to detect and process roadsides and lane markings, and warn the driver if necessary. Acceptance of LDW is expected to be a problem because control of the car is given to the computer, and chances of false alarms are still present.
	LKS	Lane Keeping System	An extended version of the LDW system is the Lane Keeping System. Instead of warning the driver about the unintended lane departure, LKS intervenes with the driving task by using steering wheel actuators. LKS can completely take over the steering task of the driver.  *Fig 3: Lane Keeping System*
	LCA	Lane Change Assistance	LCA is a collection of technologies taking care of blind spots and rear-view problems. It uses sensors to detect objects and vehicles which normally can't be seen by the driver because of obstructed view. Also, approaching vehicles from behind can be detected in time, and the driver can be informed of this.  *Fig 4: Lane Change Assistance*

Miscellaneous		Night Vision Systems	These systems provide the driver with an enhanced view of the outside world. It's meant to be used during bad weather or night time. Though already implemented in several car models, the system still has a problem with its interface: how to present the enhanced image to the user. Current solutions consist of displaying the image on a monitor on the dashboard.
		Parking Assistance	The Parking Assistance system looks like Lane Change Assistance, but is meant for low speed and short distance, for example when parking a car. Using sensors a car can measure available space, and show this information to the driver. Current systems have limited use because of the low range these sensors operate with. Future developments will let the system take over control of the car during parking, letting the car park itself.
		Fuel Economy Devices	With Fuel Economy Devices the fuel flow and usage can be monitored and analysed per car. A system can intervene by informing the driver about the fuel usage, or by actively intervening, using an active gas pedal or other active systems.

III. Benchmark and Adopted Solution:

To minimize human error, the actual ADAS system is helping the conductor by recognizing pedestrians and bicycles in an automobile's path to take action for safety. It detects pedestrians, calculates the distance to them, and display direction, detect lanes on the road…

In the first part, we justify the choice of the model chosen for the detection of pedestrians, then we explain in detail the code used, how to train, and test it.

[1] Benchmark

A. R-FCN

Region-based Fully Convolutional networks, closely resemble the architecture of Faster R-CNN, but instead of cropping features from the same layer where region proposals (RoI) are predicted, crops are taken from the last layer of features before prediction.

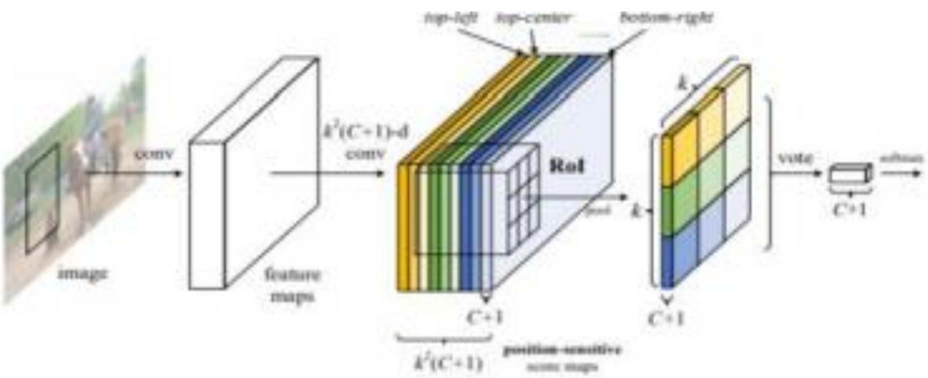


Figure 6: Architecture of R-FCN.

As shown in this paper [4], we can summarize the approach as follows:

1. Generate region proposal
2. Make classification from ROIs

3. Localization predictions from ROIs

We create 9 region-based feature maps each detecting the top-left, top-middle, top-right, middle-left, … or bottom-right area of an object. By combing the votes from these feature maps, we determine the class and the location of the objects.

B. SSD – Single Shot Multibox Detector

By using SSD, we only need to take one single shot to detect multiple objects within the image, while regional proposal network (RPN) based approaches such as R-CNN series that need two shots, one for generating region proposals, one for detecting the object of each proposal. Thus, SSD is much faster compared with two-shot RPN-based approaches.

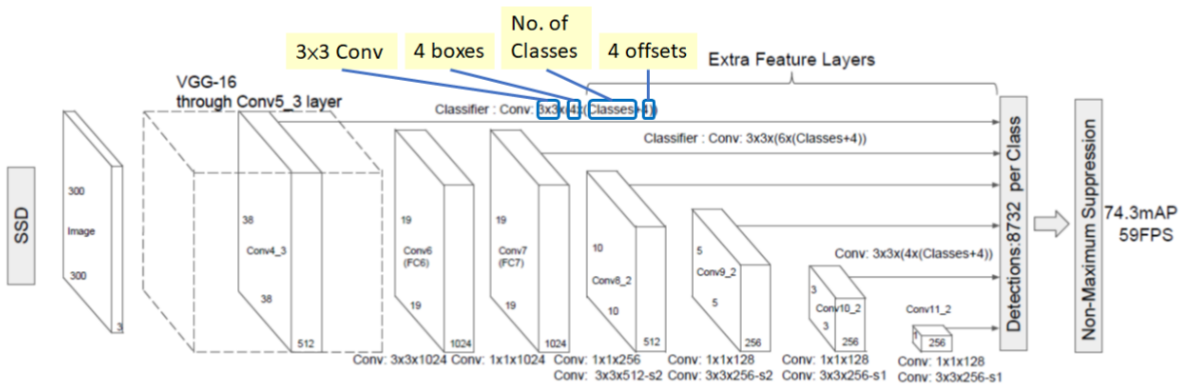


Figure 7: SSD network architecture.

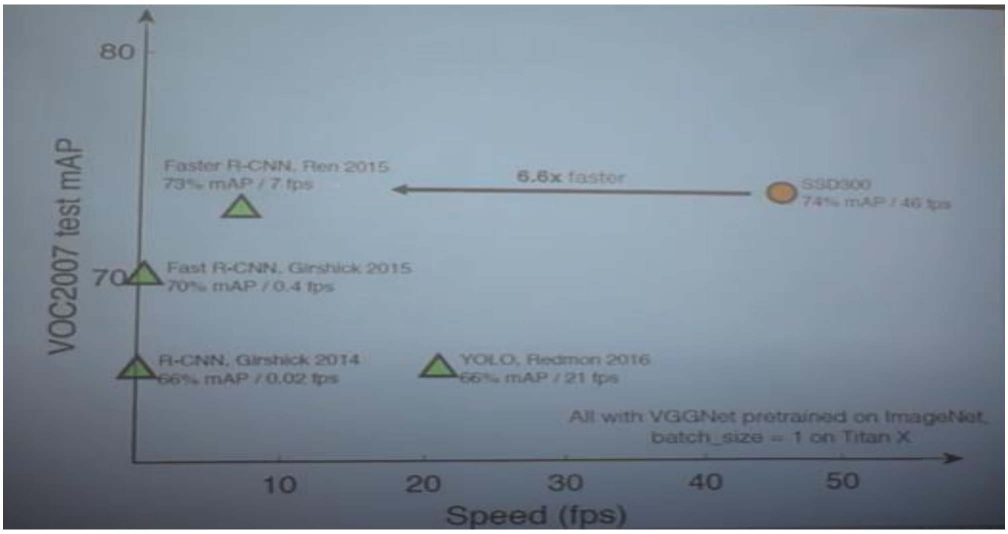


Figure 8: SSD compared to other models.

SSD compares favorably to its state-of-the-art object detector counterparts in terms of both accuracy and speed.

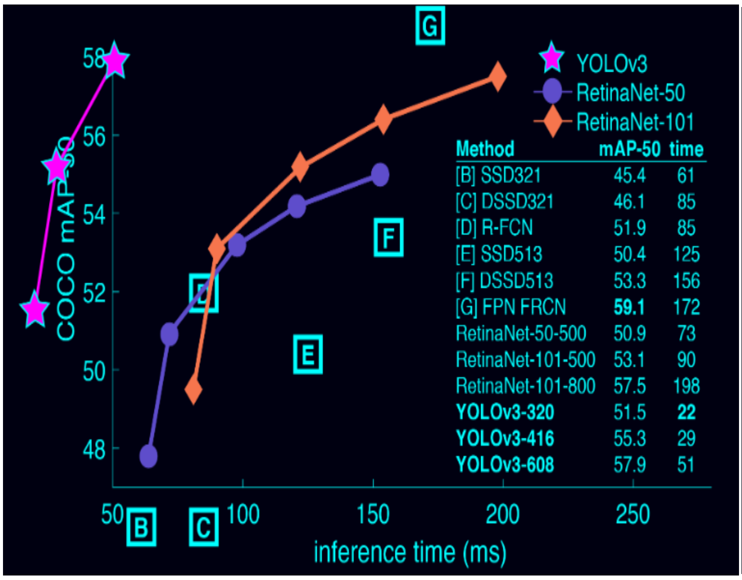
In addition to the two models mentioned, there are many others such as Fast RCC, Mask R-CNN ...

However, the masterpiece and state-of-the-art at the time of writing this report is YOLOV3. In the next section, we discuss the different versions, the frameworks, the database used to train and test the model, and the code used.

[2] YOLO V3

You Only Look Once is a state of the art, real-time object detection system by the University of Washington.

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human vision system is fast and quite accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate, algorithms for object detection would allow computers to drive cars in any weather without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems. YOLO enables cars to do so!



Model	Train	Test	mAP	FLOPS	FPS
SSD300	COCO trainval	test-dev	41.2	-	46
SSD500	COCO trainval	test-dev	46.5	-	19
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244
SSD321	COCO trainval	test-dev	45.4	-	16
DSSD321	COCO trainval	test-dev	46.1	-	12
R-FCN	COCO trainval	test-dev	51.9	-	12
SSD513	COCO trainval	test-dev	50.4	-	8
DSSD513	COCO trainval	test-dev	53.3	-	6
FPN FRCN	COCO trainval	test-dev	59.1	-	6
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20

YOLOv3 is extremely fast and accurate!

As shown above, compared with RetinaNet, YOLOv3 got comparable mAP@0.5 with much faster inference time.

i. What is YOLO v3?

Yolov3 is one of the most popular deep learning algorithms, it's fast, accurate, and different! It uses a different approach! Yolo can detect pedestrians in only one pass; This algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the network to make predictions.

Figure 9: Comparison to other detectors

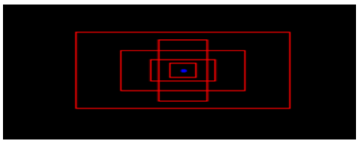
Well, we are trying in this section to explain the YOLOv3 algorithm as we understand it from the paper [7]. The deal is Detection, Classification, and Localization (Bounding box)!

Figure 10: Human detection.

- ✓ Yolo divides the input image into an S * S grids. Each grid predicts only one object.

*Figure 11: 3 * 3 grids*

- ✓ The system predicts bounding boxes using dimension clusters as anchor boxes. Anchor boxes are hand-picked boxes of different height/width ratios (for 2-dimensional boxes) designed to match the relative ratios of the object classes being detected.



- ✓ The network predicts 4 coordinates for each bounding box tx, ty, tw, th.

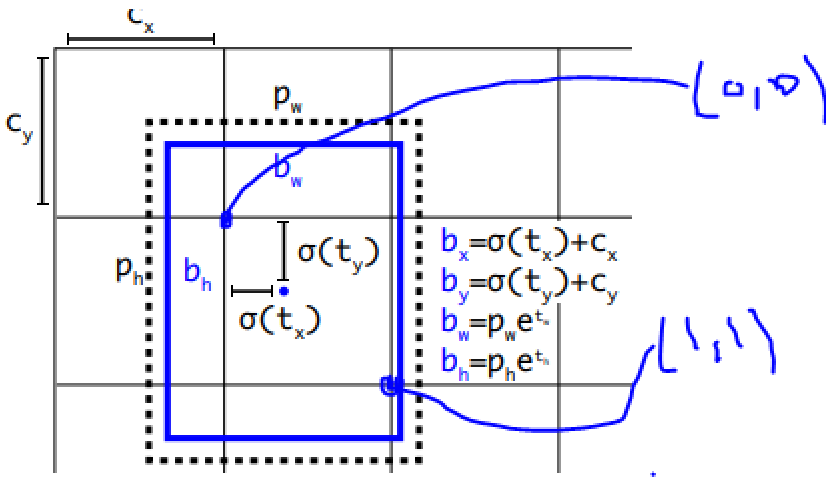


Figure 12: Bounding boxes with dimension priors and location prediction.

A sigmoid function is used to make sure the coordinates do not exceed (1,1).

- ✓ Each box predicts the classes the bounding box may contain using multilabel classification.
- ✓ The resulting output is

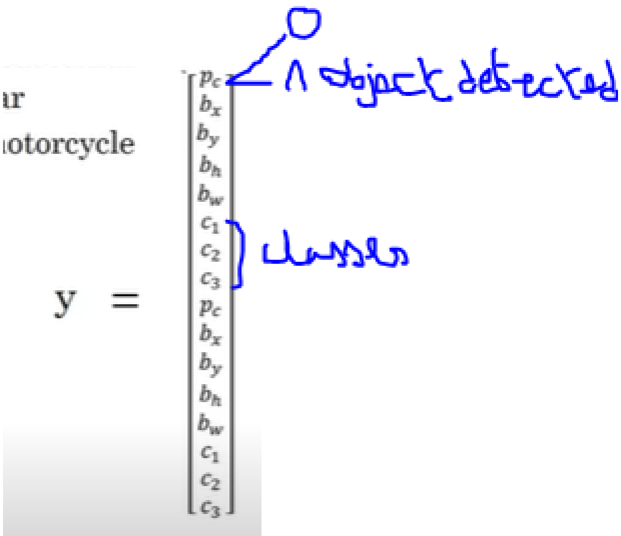


Figure 13: The output of the system.

- ✓ The system use k-means clustering to determine the bounding box priors. We just sort of chose 9 clusters and 3 scales arbitrarily and then divide up the clusters evenly across scales. On the COCO dataset the 9 clusters were: (10×13),(16×30),(33×23),(30×61),(62×45),(59× 119),(116 × 90),(156 × 198), (373 × 326).
- ✓ The feature extractor has 53 convolutional layers.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 14: Darknet-53.

There are some specifications that we will see while explaining the code. So, now that we explained the YOLO algorithm. We need to discuss the need for frameworks.

Yolo is the deep learning algorithm, and to work, it needs a framework:

Darknet: Yolo came out in 2016 with the darknet framework which was built for YOLO. Darknet is an open-source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation. But it works only on Linux. Here is a link to install and run it [9].

TensorFlow + Darkflow: DarkFlow is a python implementation of YOLO using TensorFlow.

TensorFlow is a software library or framework, designed by the Google team to implement machine learning and deep learning concepts. It can train and run deep neural networks for handwritten digit classification, image recognition, word embedding, and creation of various sequence models.

PyTorch: is an open-source machine learning library for Python and is completely based on Torch. It is primarily used for applications such as natural language processing. PyTorch is developed by Facebook's artificial-intelligence research group along with Uber's "Pyro" software for the concept of in-built probabilistic programming.

OpenCV: with at least the version 3.4.2. The Open Source Computer Vision Library, extensive documentation, and sample code for real-time computer vision.

IV. Our achievements:

[1] Object detection using YOLOV3:

We chose to run YOLOv3 with Python3.7 and OpenCV. They both work on Windows, easy to install and require no other installations.

After installing Anaconda, we create a new environment using this command

```
>conda create -n detection python=3.7
```

In this new environment, we install OpenCV

```
>conda install -c conda-forge opencv
```

And launch SPYDER, we got easily familiar with it.

The full script:

STEP 1: We need to download three files containing the configuration, the pre-trained weight file and the classes from the COCO dataset;

yolov3.cfg	24/06/2020 23:58	Fichier CFG	9 Ko
yolov3.weights	25/06/2020 00:07	Fichier WEIGHTS	242 195 Ko
coco.names	28/06/2020 17:56	Fichier NAMES	1 Ko

- ✓ COCO is free large-scale object detection, segmentation, and captioning dataset.

STEP 2: We need only 2 libraries to install: OpenCV and Numpy

STEP 3: Here is the complete code with all the necessary clarifications as comments.

```python
# Here we import OpenCV
import cv2 as cv
# We import Numpy; The fundamental package for scientific computing with Python.
import numpy as np


#Name the files downloaded.
modelConfig = 'yolov3.cfg'
modelWeights = 'yolov3.weights'
classesFile = "coco.names"

# We load the classes from the COCO file
classes = None
with open(classesFile, 'rt') as f:
    classes = f.read().rstrip('\n').split('\n')
# We only need to detect pedestrians.
# print(classes) ==> ['person'].


# We load the model
net = cv.dnn.readNetFromDarknet(modelConfig, modelWeights)

#Backend refers to the implementation
#backends = (cv.dnn.DNN_BACKEND_DEFAULT, cv.dnn.DNN_BACKEND_HALIDE, cv.dnn.DNN_BACKEND_INFERENCE_ENGINE, cv.dnn.DNN
net.setPreferableBackend(cv.dnn.DNN_BACKEND_OPENCV)

#Target refers to the processor
#DNN_TARGET_CPU, DNN_TARGET_OPENCL,  DNN_TARGET_OPENCL_FP16, DNN_TARGET_MYRIAD,  DNN_TARGET_FPGA
net.setPreferableTarget(cv.dnn.DNN_TARGET_CPU)


inpWidth = 416  # Width of networks input image
inpHeight = 416  # Height of networks input image
```

```python

inpWidth = 416  # Width of networks input image
inpHeight = 416  # Height of networks input image

confThreshold = 0.25
# nms; non-maximum suppression threshold; It is a class of algorithms to select one entity
#(e.g. bounding boxes) out of many overlapping entities
nmsThreshold = 0.40


# Get the names of the output layers
def getOutputsNames(net):

    # Get the names of all the layers in the network
    layersNames = net.getLayerNames()

    # Get the names of the output layers, i.e. the layers with unconnected outputs
    return [layersNames[i[0] - 1] for i in net.getUnconnectedOutLayers()]


# Create a VideoCapture object and read from input file
# If the input is the camera, pass 0 instead of the video file name
cap = cv.VideoCapture('detect.mp4')

#We create a window where we will display the detection, we name and resize it
winName = 'YOLOv3 with OpenCV'
cv.namedWindow(winName, cv.WINDOW_NORMAL)
cv.resizeWindow(winName, 1000, 1000)

```

```python

# The network outputs bounding boxes are each represented by a vector of number of classes + 5 elements.
# Remove the bounding boxes with low confidence using non-maxima suppression
def postprocess(frame, outs):

    # we import the current frame to calculate the coordinates of the objects detected
    frameHeight = frame.shape[0]
    frameWidth = frame.shape[1]

    classIDs = []
    confidences = []
    boxes = []

    # Scan through all the bounding boxes output and keep only the ones with high confidence scores.
    for out in outs:

        for detection in out:

            print(len(detection))
            scores = detection[5:]
            # numpy.argmax Returns the indices of the maximum values along an axis
            #Here it determines the classID
            classID = np.argmax(scores)
            confidence = scores[classID]

            #The bounding boxes kept are determinated by conThreshold = 0.25
            if confidence > confThreshold:

                centerX = int(detection[0]* frameWidth)
                centerY = int(detection[1]* frameHeight)

                width = int(detection[2]* frameWidth)
                height = int(detection[2]* frameWidth)

                left = int(centerX - width/2)
                top = int(centerY - height/2)
```

```
                classIDs.append(classID)
                confidences.append(float(confidence))
                boxes.append([left, top, width, height])


    #Perform non maximum suppression to eliminate redundant overlapping boxes with lower confidences.
    indices = cv.dnn.NMSBoxes (boxes, confidences, confThreshold, nmsThreshold )


    for i in indices:

        i = i[0]
        box = boxes[i]
        left = box[0]
        top = box[1]
        width = box[2]
        height = box[3]
        predicted_class = classes[classID]

        drawPred(classIDs[i], confidences[i], left, top, left + width, top + height, center)

#Finally, we draw the boxes that were filtered through the non maximum suppression
def drawPred(classId, conf, left, top, right, bottom, center):

    #We draw the rectangle; the bounding box
    cv.rectangle(frame, (left, top), (right, bottom), (255, 178, 50))

    label = '%.2f' % conf

    # Get the label for the class name and its confidence
    if classes:

        #Test if the condition returns True
        assert(classId < len(classes))

        label = '%s:%s' % (classes[classId], label)

        #Display the class and the confidence at the top left of the bounding box
        cv.putText(frame, label, (left, top), cv.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,255))



while cv.waitKey(1) < 0:

    hasFrame, frame = cap.read()
    if not hasFrame:
        break

    # Create a 4D blob from a frame.
    blob = cv.dnn.blobFromImage(frame, 1/255, (inpWidth, inpHeight), [0,0,0], 1, crop=False)

    #Set the input to the image
    net.setInput(blob)

    # Runs the forward pass to get output of the output layers
    outs = net.forward(getOutputsNames(net))

```

So until now, we used YOLOv3 as a pedestrian detection with OpenCV as framework and CPU as a processing unit. However, wa can use GPU to emphasis on high throughput.

We set the Backend and the target as following:

```
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
```

[2] Distance estimation from the monocular camera; DisNet

With level 3, the ADAS system not only perceives the world around. But also needs to make decisions, anticipate collisions, and avoid them. In this sense, we need to estimate the distance separating the car from the pedestrian from the image obtained from a monocular camera.

Here we use a distance estimation system based on a Multi Hidden-Layer Neural Network, named DisNet, which is used to learn and predict the distance between the object and the camera sensor.

The DisNet was trained using a supervised learning technique, where the input features were manually calculated parameters of the object bounding boxes resulted from the YOLO object classifier. And outputs were the accurate 3D laser scanner measurements of the distances to objects in the recorded scene; 2000 input feature vectors are used to train the model. The presented DisNet-based distance estimation system was evaluated on the images of railway scenes as well as on the images of a road scene.

Outputs from YOLOv3 are bounding boxes of detected objects in the image and labels of the classes detected objects belong to. The objects' bounding boxes resulted from the YOLO object classification are then processed to calculate the features, bounding boxes parameters. Based on the input features, the trained DisNet gives as outputs the estimated distance of the object to the camera sensor.

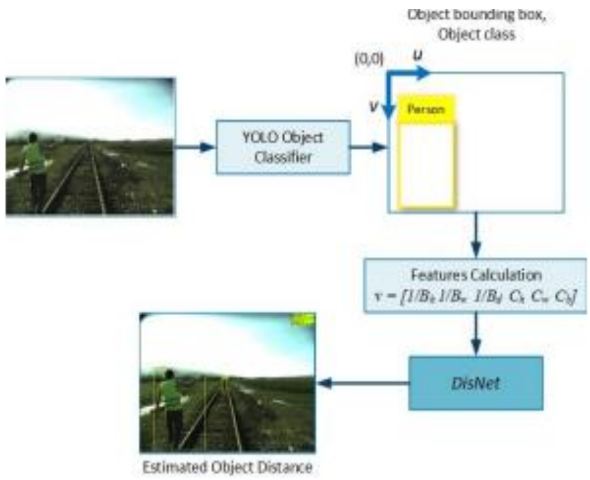


Figure 15: The DisNet-based system used for object distance estimation from a monocular camera

For each extracted object bounding box, a six-dimensional feature vector v was calculated:

$$v = [1/Bh \quad 1/Bw \quad 1/Bd \quad Ch \quad Cw \quad Cb]$$

Height, Bh = (height of the object bounding box in pixels/image height in pixels)

Width, Bw = (width of the object bounding box in pixels/image width in pixels)

Diagonal, Bd = (diagonal of the object bounding box in pixels/image diagonal in pixels).

Ch, Cw, and **Cb** are the values of average height, width, and breadth of an object of the particular class

After calculating the vector v, we use it as an input to the following model:

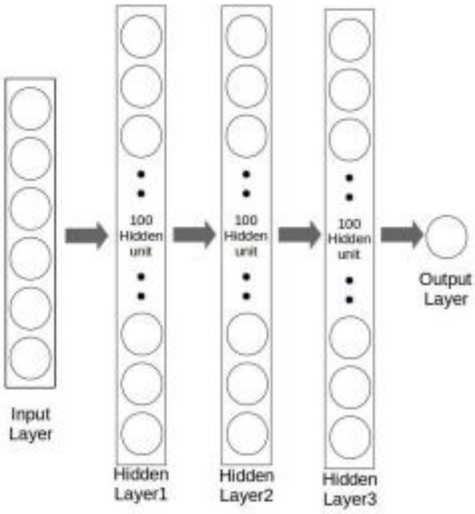


Figure 16: The structure of DisNet.

For more pieces of information, here is the link to the original paper [12].

To detect pedestrians OpenCV is used as a framework. Now to estimate the distance, the architecture of DisNet is implemented with KERAS.

What is KERAS?

KERAS is an open Source python based deep learning framework with a high performing API used to specify and train differentiable programs. KERAS has three types of models:

-The Sequential model: a simple list of layers.

-The functional API supports arbitrary model architectures.

-Model subclassing: where everything has to be implemented from scratch.

We still work on the code explained above. We add to the functions some other specifications concerning the distance model:

- ✓ First, we Import the libraries that we need.

```
#We import load_model to load the model pre-trained
from keras.models import load_model
#container behaving like a list with additions and deletions
from collections import deque
```

- ✓ We need to download then upload the model pre-trained.

model_3.keras 26/06/2020 23:27 Fichier KERAS 155 Ko

```
#name the model's path
distance_model_path = "model_3.keras"
#load  the model
#boolean whether to compile the model after loading.
distance_model = load_model(distance_model_path, compile=True)
```

- ✓ Now, we need to calculate the vector v using this function

```
def load_dist_input( predict_box, classID, img_width, img_height):

        predict_class = classes[classID]

        cc = ['person']

        #Make sur its a human that is detected
        if str(predict_class) in cc:

            top, left, bottom, right = predict_box

            width = float(right - left) / img_width
            height = float(bottom - top) / img_height
            #calculate the diagonal
            diagonal = np.sqrt(np.square(width) + np.square(height))

            class_h, class_w, class_d = set_class_size[predict_class]

            #The input to the model
            dist_input = [1 / width, 1 / height, 1 / diagonal, class_h, class_w, class_d]

        else:

            dist_input = [0, 0, 0, 0, 0, 0]

        return np.array(dist_input)
```

```
# In case we need to determine other classes distance
classes = ['person', 'bus', 'truck', 'car', 'bicycle', 'motorbike', 'cat', 'dog', 'horse', 'sheep', 'cow']

class_shape = [[115, 45, 10], [300, 250, 1200], [400, 350, 1500], [160, 180, 400], [110, 50, 180],
               [110, 50, 180], [40, 20, 50], [50, 30, 60], [180, 60, 200], [130, 60, 150], [170, 70, 200]]

set_class_size = dict(zip(classes, class_shape))
#print("Load Class size!")
```

- ✓ After keeping the bounding boxes with high confidences and applying NMS Threshold we calculate the top, bottom, left, and right of the box and then, we estimate the distance.

```
            Top = max(0, np.floor(top + 0.5).astype('int32'))
            Left = max(0, np.floor(left + 0.5).astype('int32'))
            bottom = min(frameWidth, np.floor(Top+height + 0.5).astype('int32'))
            right = min(frameHeight, np.floor(Left+width + 0.5).astype('int32'))
            boxx = [Top, Left, bottom, right]


            distance_input = load_dist_input(boxx, classIDs[i], frameWidth, frameHeight)
            #Here we predict the distance to the object
            distance = distance_model.predict(np.array([distance_input]).reshape(-1, 6))

```

- ✓ We draw the prediction.

```
    # draw prediction
        drawPred( i, classIDs[i], confidences[i], left, top, left + width, top + height, center, distance
```

```
predict_class = classes[classId]
cc = ['person']
if str(predict_class) in cc:

    label_left = '{} {} Distance: {:.2f}M'.format(classes[classId], int(i+1), float(np.squeeze(distance)))

    cv.putText(frame , label_left, (50,50+i*20), cv.FONT_HERSHEY_SIMPLEX, 0.25, (0,0,255), 1)

```

We use DisNet model to estimate Distance but we should still note that many other algorithms approach distance. Here are the links for more information [13] [14].

A. *Crash Risk*

We have to take into account that even in some scenarios, the pedestrian is close to the car, it may not be in its path and therefore there is no risk of collision. In what follows we display a WARNING in the event of a probable collision.

```
# the risk of collision
    if distance <1.70 :
        mid_x = center[0]
        if mid_x >0.3 and mid_x <1:
            # the ridke of the collision
             cv.putText(frame, "Warniiiing!!!", (left + 20, top), cv.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,255))
```

B. *Center*

We want to add a circle at the center of the bounding boxes.

```
        #calculate center of object
          center = (round(left + (width/ 2)), round(top + (height / 2)))
```

We put the line of code that draws the circle in the function DrawPred

```
# draw prediction
    drawPred( i, classIDs[i], confidences[i], left, top, left + width, top + height, center, distance
```

```
        cv.circle(frame, center, 5, (0, 0, 255), 5)
```

C. Direction

Another feature that we add here is the pedestrians' direction.

- ✓ Here is the function of calculating the different directions.

```
def getDirection(image, pointList):

    dx = 0
    dy = 0

    for x in range(len(pointList)-1):

        cv.line(image, pointList[x], pointList[x+1], [0, 255, 0], 10)

        dx += pointList[x+1][0] - pointList[x][0]
        dy += pointList[x+1][1] - pointList[x][1]

    x = ""
    y = ""

    if(dx < 0):
        x = "right"

    if(dx > 0):
        x = "left"

    if(dy < 0):
        y = "down"

    if(dy > 0):
        y = "up"

    return (x,y)
```

- ✓ After determining the bounding boxes, we calculate the direction based on the center of the bounding box.

```

            nom = classes[classID]
            if nom in pointsDict:
                pointsDict[nom].appendleft(tuple(center))

            else:

                pointsDict[nom] = deque(maxlen=25)
                pointsDict[nom].appendleft(tuple(center))

            if len(pointsDict[nom]) > 6:

               xdir, ydir = getDirection(frame, pointsDict[nom])

            else:
               xdir = ".."
               ydir = ".."

```

- ✓ Now In the DrawPred function, we add the lines that enable the program to draw lines showing the movement of the pedestrians and put the text mentioning the direction.

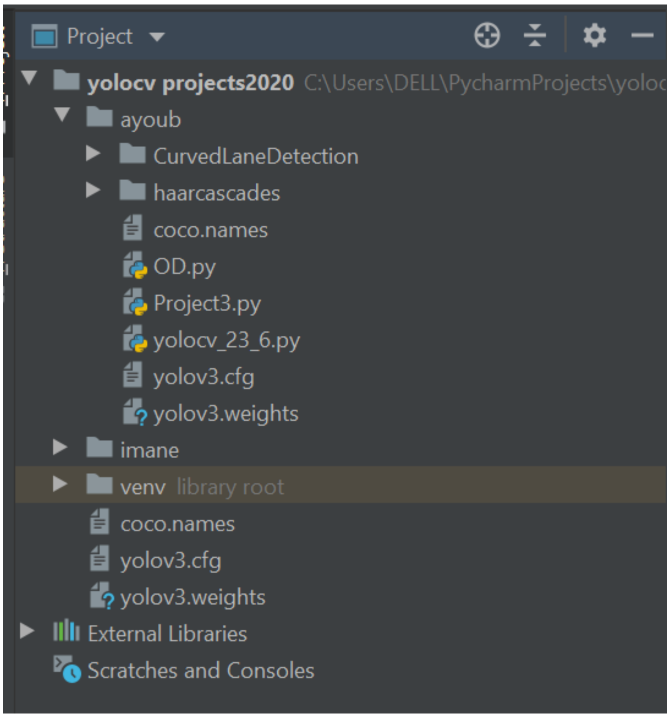
```
        cv.putText(frame, xdir  + " - " + ydir, (left, top - 20), cv.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,255), 1)
```

4. Number Plate detection:

A. Why do need to detect it?

In the case of a hit and run accident or some kind of crime or problem with other cars, the ego car might have to keep data on the car that was involved. For that, the most crucial information is the number plate and saving it to our database.

B. The script :



To detect Number plates, we're going to use a method proposed by Viola and Jones, this method was one of the earliest methods that allowed real-time object detection. If we want to detect number plates, we will collect a lot of positives images of them and also a lot of negatives which will contain anything but number plates. We will train and create a cascade file using these negative and positive data, this file will help us find faces in the images. In our case, we will not train the model from scratch, instead, we will use the pre-trained file tor number plates, which is provided by OpenCV. OpenCV cascades contain default ones that can detect different things such as number plates, eyes, faces, full-body, etc.

P.S: This functionality was added to the original script discussed above.

First, feed the function cv2.CascadeClassifier with the .xml file of number plates "haarcascade_russian_plate_number.xml". Then, we're converting our image to gray.

```python
minArea = 500
color= (255,0,100)
nPlateCascade = cv.CascadeClassifier("haarcascades/haarcascade_russian_plate_number.xml")
plateDetect = 1
with open(classesFile, 'rt') as f:
    classes = f.read().rstrip('\n').split('\n')
```

Inside our loop, we set condition to activate plate number detection if the variable plateDetect equal to 1. We detect the number Plates using the NP cascade, the output of detectMultiScale function is a list we named numberPlates containing the coordinates of the sources point of the bounding box (x,y) also the width and height of it (w,h).

```python
plate_cout = 0
while cv.waitKey(1) < 0:
    # get frame from video
    hasFrame, frame = cap.read()
    # Create a 4D blob from a frame
    blob = cv.dnn.blobFromImage(frame, 1 / 255, (inpWidth, inpHeight), [0, 0, 0], 1, crop=False)
    # Set the input the the net
    net.setInput(blob)
    outs = net.forward(getOutputsNames(net))
    if plateDetect == 1:
        imgGray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
        numberPlates = nPlateCascade.detectMultiScale(imgGray, 1.1, 4)
    postprocess(frame, outs)
```

We then apply our function postProcessPlate, we first add our filter which filters out the bounding boxes whose areas are less than the minimum area. To do that, we find the area of the bounding boxes. We already defined our minArea which is here equal 500. If the condition of the minimum area is satisfied, we draw a rectangle over it and label it putting a text saying “Number Plate” using the parameters needed for colors and fonts and size.

Since we got the x, y, width, and height of the bounding box we will use this information to crop our image and get the region of our interest which is the number plate. We store the output in the variable imgRoi. Whenever a number plate is detected we display the imgRoi.

```python
    if plateDetect == 1:
        postProcessPlate(frame,numberPlates)
    # show the image
    cv.imshow(winName, frame)
```

```
def postProcessPlate(img,numberPlates):
    for x, y, w, h in numberPlates:
        area = w*h
        if area > minArea:
            cv.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
            cv.putText(img,"number Plate",(x,y-5),
                        cv.FONT_HERSHEY_COMPLEX_SMALL,1,color,2)
            imgRoi = img[y:y+h,x:x+w]
            cv.imshow("PLate",imgRoi)
```

As we can see in the images below, we are getting our number plate detected on the display of our laptop camera feed, and extracting our region of interest.

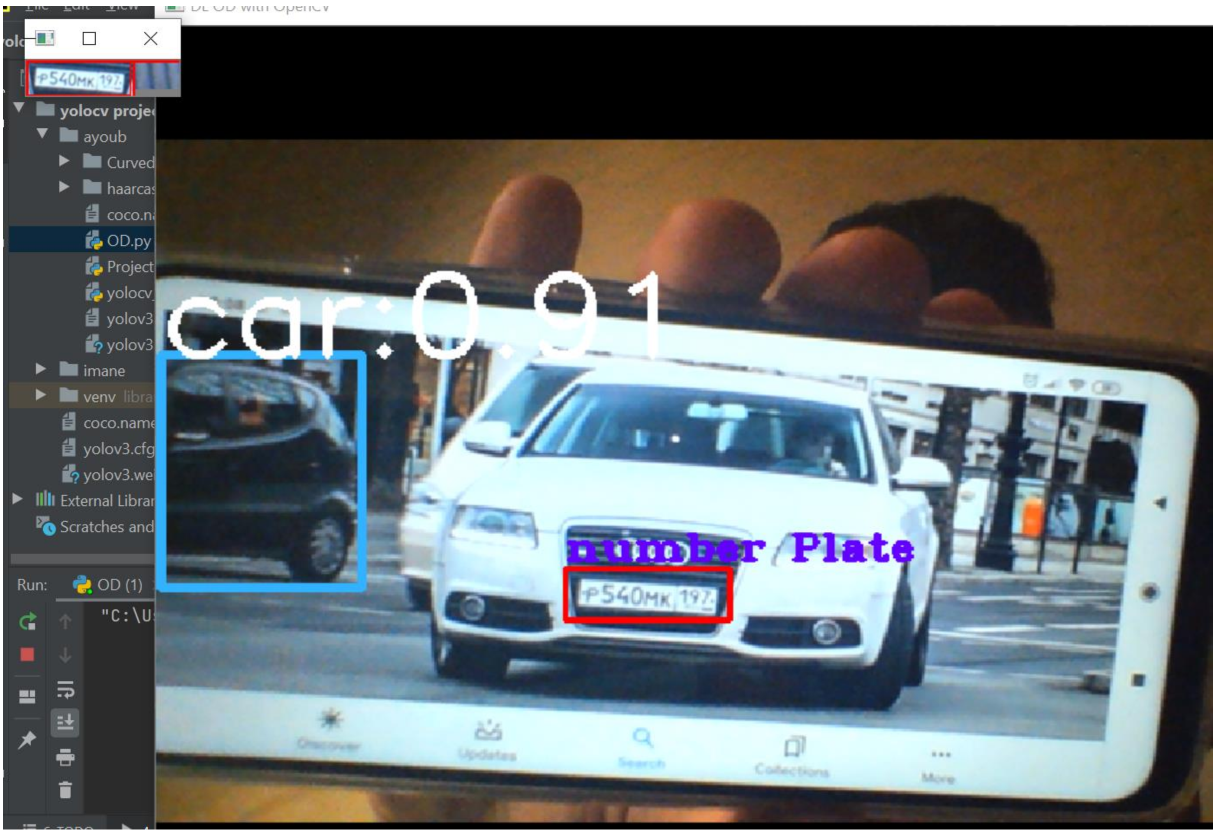
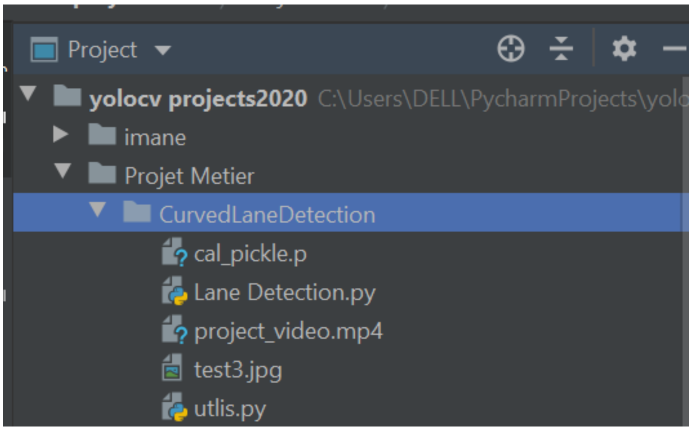


Figure 17: Plate number detection demo

We can add the possibility of saving the little cropped image if we want, but that's not our main focus in this project.

5. Lane detection:



In this part, we will try to detect the lanes on the road as well as the curvature of it. The method we will use is proposed by Ross Kippenbrock. Initially, we will capture the image and remove the distortion caused by the camera lens. Then we will apply a yellow and white colors filters along with edge detector a combine both of them, to get a stable result. Next, with four selected points we will define the warp perspective inputs to get the bird-eye view of our desired lane.

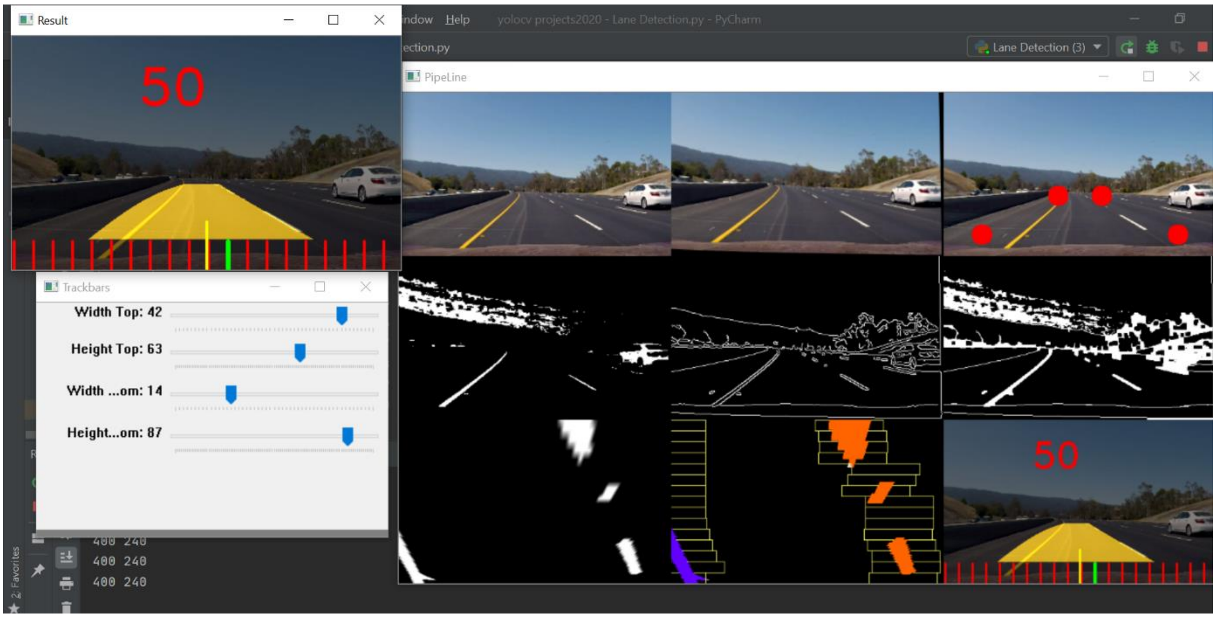


Figure 18: Lane Detection on a real world lanes

PS: The next scripts are separate from the project above, because the running time would be so big and we won't be able to all functions and purposes fulfilled at once with the current laptop we own for the time being. But we can always fuse these to project whenever we have better equipment.

In the code we have the lane detection script and utilities script.

First, we import our libraries "OpenCV" and "numpy" and "utlis" library that contains our important functions we wrote from CurvedLaneDetection folder.

```
Lane Detection.py
import numpy as np
import cv2
from ProjetMetier.CurvedLaneDetection import utlis
```

We define our parameters, if the camerafeed is true we will use the laptop camera, else we'll use the video path we're defining. Then we've got our camera parameters defined using the variables framewidth and frameheight.

```
cameraFeed= False
videoPath = 'project_video.mp4'
cameraNo= 0
frameWidth= 400
frameHeight = 240
if cameraFeed:intialTracbarVals = [24,55,12,100] #   #wT,hT,wB,hB
else:intialTracbarVals = [42,63,14,87]   #wT,hT,wB,hB
####################################################
if cameraFeed:
    cap = cv2.VideoCapture(cameraNo)
    cap.set(3, frameWidth)
    cap.set(4, frameHeight)
else:
    cap = cv2.VideoCapture(videoPath)
count=0
noOfArrayValues =10
global arrayCurve, arrayCounter
arrayCounter=0
arrayCurve = np.zeros([noOfArrayValues])
myVals=[]
utlis.initializeTrackbars(intialTracbarVals)
```

Inside the while loop, the code itself is quite short as we used the predefined functions from "utlis" script which contains the real code.

First, we get our image and make few copies of that we'll use later. We pass the original image through utlis.undistort function.

```
while True:

    success, img = cap.read()
    #img = cv2.imread('test3.jpg')
    if cameraFeed == False:
        img = cv2.resize(img, (frameWidth, frameHeight), None)
    imgWarpPoints = img.copy()
    imgFinal = img.copy()
    imgCanny = img.copy()

    imgUndis = utlis.undistort(img)
```

Let’s see how does it work. Basically, it uses the calibration pickle file, which was already created manually, to undistort the image.

```
def undistort(img, cal_dir='cal_pickle.p'):
    with open(cal_dir, mode='rb') as f:
        file = pickle.load(f)
    mtx = file['mtx']
    dist = file['dist']
    dst = cv2.undistort(img, mtx, dist, None, mtx)
    return dst
```

Then we go through the thresholding.

Inside this function, we are converting the image to gray, adding some blur and then canny filter which helps us to detect edges. After that, we apply dilation then erosion functions to improve our preprocessing. We get then the imgColor which is the output of the colorFilter function.

```
imgThres,imgCanny,imgColor = utlis.thresholding(imgUndis)
```

```python
def thresholding(img):
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    kernel = np.ones((5,5))
    imgBlur = cv2.GaussianBlur(imgGray,(5,5), 0)
    imgCanny = cv2.Canny(imgBlur, 50, 100)
    #imgClose = cv2.morphologyEx(imgCanny, cv2.MORPH_CLOSE, np.ones((10,10)))
    imgDial = cv2.dilate(imgCanny,kernel,iterations=1)
    imgErode = cv2.erode(imgDial,kernel,iterations=1)

    imgColor = colorFilter(img)
    combinedImage = cv2.bitwise_or(imgColor, imgErode)

    return combinedImage,imgCanny,imgColor
```

In the colorFilter function, we convert the image to hsv so that we can use the color ranges we define bellow. We defined the upper and lower limit of the colors that we need to capture, here are yellow and white colors as the road lines are always drawn using these two. We then combine the two results of the function cv2.inRange "maskedWhite" and "maskedYellow" using cv2.bitwise_or function to get our output "combinedImage".

```python
def colorFilter(img):
    hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
    lowerYellow = np.array([18,94,140])
    upperYellow = np.array([48,255,255])
    lowerWhite = np.array([0, 0, 200])
    upperWhite = np.array([255, 255, 255])
    maskedWhite= cv2.inRange(hsv,lowerWhite,upperWhite)
    maskedYellow = cv2.inRange(hsv, lowerYellow, upperYellow)
    combinedImage = cv2.bitwise_or(maskedWhite,maskedYellow)
    return combinedImage
```

We then return to thresholding function and perform cv2.bitwise_or once again to combine imgColor and imgErode.

The second step in our LaneDetection script is perspective_warp. This function needs four points which we get from trackbars. The trackbar was initialized in the beginning of the code in the positions we can see in the image below. We can move these points around thanks to the track bars. The points are chosen as to keep it and the angle as close as possible to the lane lines.

These points are drawn using the utlis.drawPoints. it's basically just few circles we are drawing and filling with red color.

```python
    src = utlis.valTrackbars()
    imgWarp = utlis.perspective_warp(imgThres, dst_size=(frameWidth, frameHeight), src=src)
    imgWarpPoints = utlis.drawPoints(imgWarpPoints, src)
```

```python
def perspective_warp(img,
                     dst_size=(1280, 720),
                     src=np.float32([(0.43,0.65),(0.58,0.65),(0.1,1),(1,1)]),
                     dst=np.float32([(0,0), (1, 0), (0,1), (1,1)])):
    img_size = np.float32([(img.shape[1],img.shape[0])])
    src = src* img_size
    # For destination points, I'm arbitrarily choosing some points to be
    # a nice fit for displaying our warped result
    # again, not exact, but close enough for our purposes
    dst = dst * np.float32(dst_size)
    # Given src and dst points, calculate the perspective transform matrix
    M = cv2.getPerspectiveTransform(src, dst)
    # Warp the image using OpenCV warpPerspective()
    warped = cv2.warpPerspective(img, M, dst_size)

    return warped
```

```python
def drawPoints(img,src):
    img_size = np.float32([(img.shape[1],img.shape[0])])
    #src = np.float32([(0.43, 0.65), (0.58, 0.65), (0.1, 1), (1, 1)])
    src = src * img_size
    for x in range( 0,4):
        cv2.circle(img,(int(src[x][0]),int(src[x][1])),15,(0,0,255),cv2.FILLED)
    return img
```

The third part is the histogram peak detection. The reason we need these peaks is to get the starting points of our lane. For that we use the sliding_window function.

```python
    imgSliding, curves, lanes, ploty = utlis.sliding_window(imgWarp, draw_windows=True)
```

In the sliding_window we are getting our histograms. We get the peaks at the left and at the right point. This will give us the initial position of where we can place our first box. Once we place it, we will average the values in that box, and place the next one on top of it based of the result. Nwindows is the number of windows in one image which will determine the height while margin is the width of each box.

Then we are checking the windows one by one, and based on their mean position we are going to recenter it. This will help us get the center points of each box.

The next stage is fitting the curve. We’re going to use second order polynomial function which has three coefficients a, b, and c for each curved lane detected. We get these values by using the polyfit function, then we average the last 10 values to get better results.

```python
def sliding_window(img, nwindows=15, margin=50, minpix=1, draw_windows=True):
    global left_a, left_b, left_c, right_a, right_b, right_c
    left_fit_ = np.empty(3)
    right_fit_ = np.empty(3)
    out_img = np.dstack((img, img, img)) * 255

    histogram = get_hist(img)
    # find peaks of left and right halves
    midpoint = int(histogram.shape[0] / 2)
    leftx_base = np.argmax(histogram[:midpoint])
    rightx_base = np.argmax(histogram[midpoint:]) + midpoint

    # Set height of windows
    window_height = np.int(img.shape[0] / nwindows)
    # Identify the x and y positions of all nonzero pixels in the image
    nonzero = img.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])
    # Current positions to be updated for each window
    leftx_current = leftx_base
    rightx_current = rightx_base

```

```python
    # Create empty lists to receive left and right lane pixel indices
    left_lane_inds = []
    right_lane_inds = []

    # Step through the windows one by one
    for window in range(nwindows):
        # Identify window boundaries in x and y (and right and left)
        win_y_low = img.shape[0] - (window + 1) * window_height
        win_y_high = img.shape[0] - window * window_height
        win_xleft_low = leftx_current - margin
        win_xleft_high = leftx_current + margin
        win_xright_low = rightx_current - margin
        win_xright_high = rightx_current + margin
        # Draw the windows on the visualization image
        if draw_windows == True:
            cv2.rectangle(out_img, (win_xleft_low, win_y_low), (win_xleft_high, win_y_high),
                          (100, 255, 255), 1)
            cv2.rectangle(out_img, (win_xright_low, win_y_low), (win_xright_high, win_y_high),
                          (100, 255, 255), 1)
            # Identify the nonzero pixels in x and y within the window
        good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
                          (nonzerox >= win_xleft_low) & (nonzerox < win_xleft_high)).nonzero()[0]
```

```python
        good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
                          (nonzerox >= win_xleft_low) & (nonzerox < win_xleft_high)).nonzero()[0]
        good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
                           (nonzerox >= win_xright_low) & (nonzerox < win_xright_high)).nonzero()[0]
        # Append these indices to the lists
        left_lane_inds.append(good_left_inds)
        right_lane_inds.append(good_right_inds)
        # If you found > minpix pixels, recenter next window on their mean position
        if len(good_left_inds) > minpix:
            leftx_current = np.int(np.mean(nonzerox[good_left_inds]))
        if len(good_right_inds) > minpix:
            rightx_current = np.int(np.mean(nonzerox[good_right_inds]))
```

```python
    # Concatenate the arrays of indices
    left_lane_inds = np.concatenate(left_lane_inds)
    right_lane_inds = np.concatenate(right_lane_inds)

    # Extract left and right line pixel positions
    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]

    if leftx.size and rightx.size:
        # Fit a second order polynomial to each
        left_fit = np.polyfit(lefty, leftx, 2)
        right_fit = np.polyfit(righty, rightx, 2)

        left_a.append(left_fit[0])
        left_b.append(left_fit[1])
        left_c.append(left_fit[2])

        right_a.append(right_fit[0])
        right_b.append(right_fit[1])
        right_c.append(right_fit[2])
```

```
        left_fit_[0] = np.mean(left_a[-10:])
        left_fit_[1] = np.mean(left_b[-10:])
        left_fit_[2] = np.mean(left_c[-10:])

        right_fit_[0] = np.mean(right_a[-10:])
        right_fit_[1] = np.mean(right_b[-10:])
        right_fit_[2] = np.mean(right_c[-10:])

        # Generate x and y values for plotting
        ploty = np.linspace(0, img.shape[0] - 1, img.shape[0])

        left_fitx = left_fit_[0] * ploty ** 2 + left_fit_[1] * ploty + left_fit_[2]
        right_fitx = right_fit_[0] * ploty ** 2 + right_fit_[1] * ploty + right_fit_[2]

        out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [255, 0, 100]
        out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [0, 100, 255]

        return out_img, (left_fitx, right_fitx), (left_fit_, right_fit_), ploty
    else:
        return img,(0,0),(0,0),0
```

Next, we are going to take a look at get_curve function to find the value of the curve which basically the radius. Based on our previous a, b, and c coefficient we calculate the radii of the curve. Since we have different curvature in the left and right lane, we're getting the mean of the two values.

Once we are done with that; we are going to the curves on the original image. For that, we will use the draw_lanes function. Then we apply the inverse perspective, which is basically the reverse of what we've done above to get back to our original image.

Finally, we will apply the cv2.putText to visualize the curve value and plot all of the image in one whole window using the stacking function.

```
    try:
        curverad =utlis.get_curve(imgFinal, curves[0], curves[1])
        lane_curve = np.mean([curverad[0], curverad[1]])
        imgFinal = utlis.draw_lanes(img, curves[0], curves[1],frameWidth,frameHeight,src=src)

        # ## Average
        currentCurve = lane_curve // 50
        if  int(np.sum(arrayCurve)) == 0:averageCurve = currentCurve
        else:
            averageCurve = np.sum(arrayCurve) // arrayCurve.shape[0]
        if abs(averageCurve-currentCurve) >200: arrayCurve[arrayCounter] = averageCurve
        else :arrayCurve[arrayCounter] = currentCurve
        arrayCounter +=1
        if arrayCounter >=noOfArrayValues : arrayCounter=0
        cv2.putText(imgFinal, str(int(averageCurve)), (frameWidth//2-70, 70), cv2.FONT_HERSHEY_DUPLEX, 1
    except:
        lane_curve=00
        pass
    imgFinal= utlis.drawLines(imgFinal,lane_curve)
    imgThres = cv2.cvtColor(imgThres,cv2.COLOR_GRAY2BGR)
```

```python
def get_curve(img, leftx, rightx):
    ploty = np.linspace(0, img.shape[0] - 1, img.shape[0])
    y_eval = np.max(ploty)
    ym_per_pix = 1 / img.shape[0]  # meters per pixel in y dimension
    xm_per_pix = 0.1 / img.shape[0]  # meters per pixel in x dimension

    # Fit new polynomials to x,y in world space
    left_fit_cr = np.polyfit(ploty * ym_per_pix, leftx * xm_per_pix, 2)
    right_fit_cr = np.polyfit(ploty * ym_per_pix, rightx * xm_per_pix, 2)
    # Calculate the new radii of curvature
    left_curverad = ((1 + (2 * left_fit_cr[0] * y_eval * ym_per_pix + left_fit_cr[1]) ** 2) ** 1.5) / np
        2 * left_fit_cr[0])
    right_curverad = ((1 + (2 * right_fit_cr[0] * y_eval * ym_per_pix + right_fit_cr[1]) ** 2) ** 1.5) /
        2 * right_fit_cr[0])

    car_pos = img.shape[1] / 2
    l_fit_x_int = left_fit_cr[0] * img.shape[0] ** 2 + left_fit_cr[1] * img.shape[0] + left_fit_cr[2]
    r_fit_x_int = right_fit_cr[0] * img.shape[0] ** 2 + right_fit_cr[1] * img.shape[0] + right_fit_cr[2]
    lane_center_position = (r_fit_x_int + l_fit_x_int) / 2
    center = (car_pos - lane_center_position) * xm_per_pix / 10
    # Now our radius of curvature is in meters

    return (l_fit_x_int, r_fit_x_int, center)
```

```python
def drawLines(img,lane_curve):
    myWidth = img.shape[1]
    myHeight = img.shape[0]
    print(myWidth,myHeight)
    for x in range(-30, 30):
        w = myWidth // 20
        cv2.line(img, (w * x + int(lane_curve // 100), myHeight - 30),
                 (w * x + int(lane_curve // 100), myHeight), (0, 0, 255), 2)
    cv2.line(img, (int(lane_curve // 100) + myWidth // 2, myHeight - 30),
             (int(lane_curve // 100) + myWidth // 2, myHeight), (0, 255, 0), 3)
    cv2.line(img, (myWidth // 2, myHeight - 50), (myWidth // 2, myHeight), (0, 255, 255), 2)

    return img
```

```python
def inv_perspective_warp(img,
                         dst_size=(1280,720),
                         src=np.float32([(0,0), (1, 0), (0,1), (1,1)]),
                         dst=np.float32([(0.43,0.65),(0.58,0.65),(0.1,1),(1,1)])):
    img_size = np.float32([(img.shape[1],img.shape[0])])
    src = src* img_size
    # For destination points, I'm arbitrarily choosing some points to be
    # a nice fit for displaying our warped result
    # again, not exact, but close enough for our purposes
    dst = dst * np.float32(dst_size)
    # Given src and dst points, calculate the perspective transform matrix
    M = cv2.getPerspectiveTransform(src, dst)
    # Warp the image using OpenCV warpPerspective()
    warped = cv2.warpPerspective(img, M, dst_size)
    return warped

def get_hist(img):
    hist = np.sum(img[img.shape[0]//2:,:], axis=0)
    return hist
```

```python
def textDisplay(curve,img):
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(img, str(curve), ((img.shape[1]//2)-30, 40), font, 1, (255, 255, 0), 2, cv2.LINE_AA)
    directionText=' No lane '
    if curve > 10:
        directionText='Right'
    elif curve < -10:
        directionText='Left'
    elif curve <10 and curve > -10:
        directionText='Straight'
    elif curve == -1000000:
        directionText = 'No Lane Found'
    cv2.putText(img, directionText, ((img.shape[1]//2)-35,(img.shape[0])-20 ), font, 1, (0, 200, 200), 2, cv2.LINE_AA)
```

```python
    imgThres = cv2.cvtColor(imgThres,cv2.COLOR_GRAY2BGR)
    imgBlank = np.zeros_like(img)
    imgStacked = utlis.stackImages(0.7, ([img,imgUndis,imgWarpPoints],
                                         [imgColor, imgCanny, imgThres],
                                         [imgWarp,imgSliding,imgFinal]
                                         ))

    cv2.imshow("PipeLine",imgStacked)
    cv2.imshow("Result", imgFinal)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

To sum up our work, lets take a look at our images stacked all together. The first one is the original image, then the undistorted one. We have the image with warp points that we can change using the trackbars.

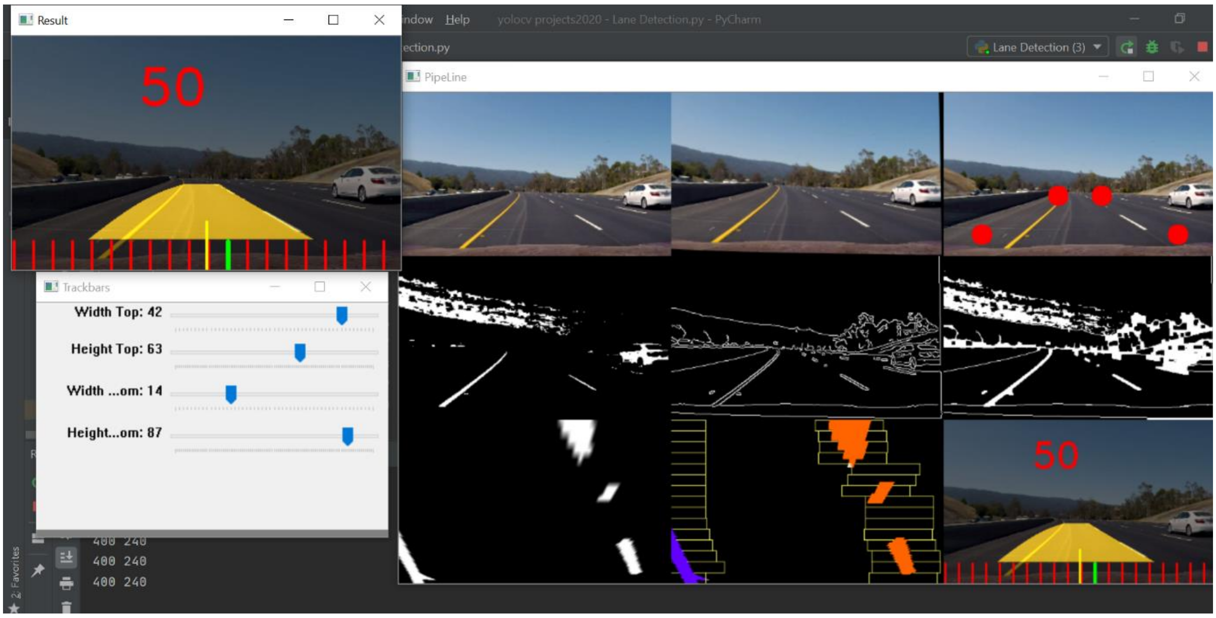


Figure 19: Lane Detection on a real world lanes

The second row contains the color and canny images then the combination of the two of them. We do then the warp perspective, from which we get the histogram initial boxes using the peak points. We draw on top of them boxes based on the mean position of the previous box, which will allow us to follow the curve. Then, we will take the center points to do a polyfit for a second-order polynomial from which we can derive the radii of the curvature these curves, then get the mean of our curvatures. Then we go back to our original image.

V. Demo on the generated scenarios:

1. Object detection and distance measurement:

You can find the demo in the link [30] to the playlist of demos, and we put here some screenshots from the videos:

Figure 20: Object detection on a generated scenario from CARLA

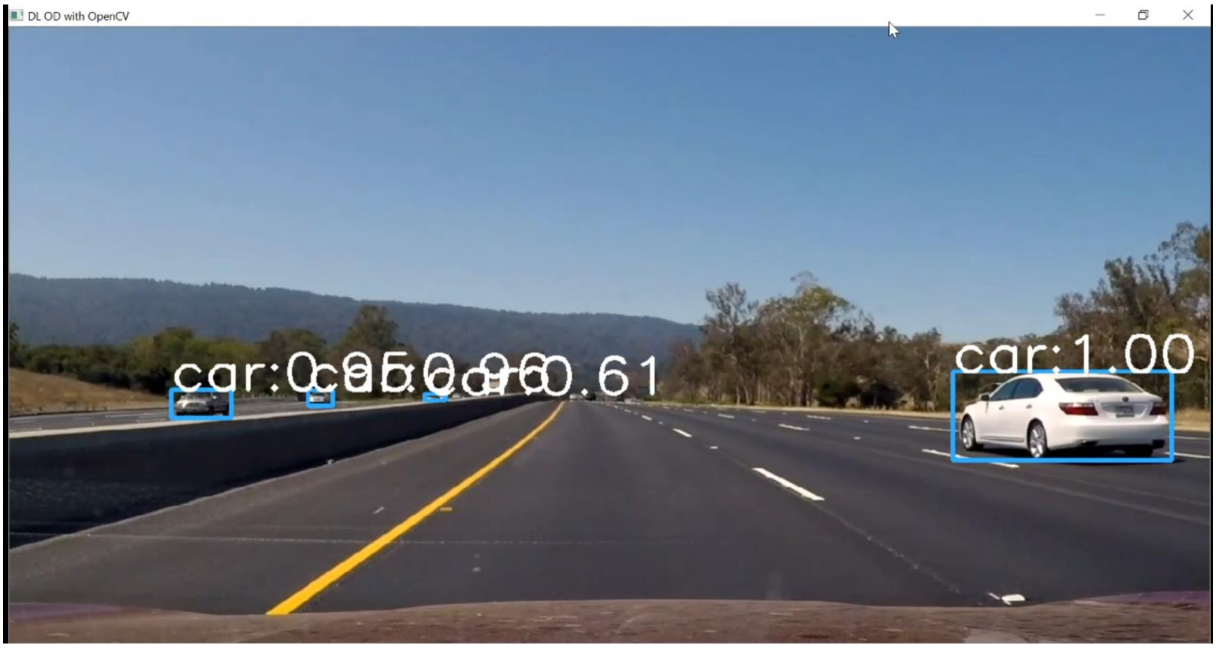


Figure 21: Object detection on a real world situation on highway

2. Plate detection:

We put here an image demonstrating how the laptop detect and image of a car with plate detection:

Figure 22: Number Plate detection Demo

2. Plate detection:

You can find the demo in the link [30] to the playlist of demos, and we put here some screenshots from the video:

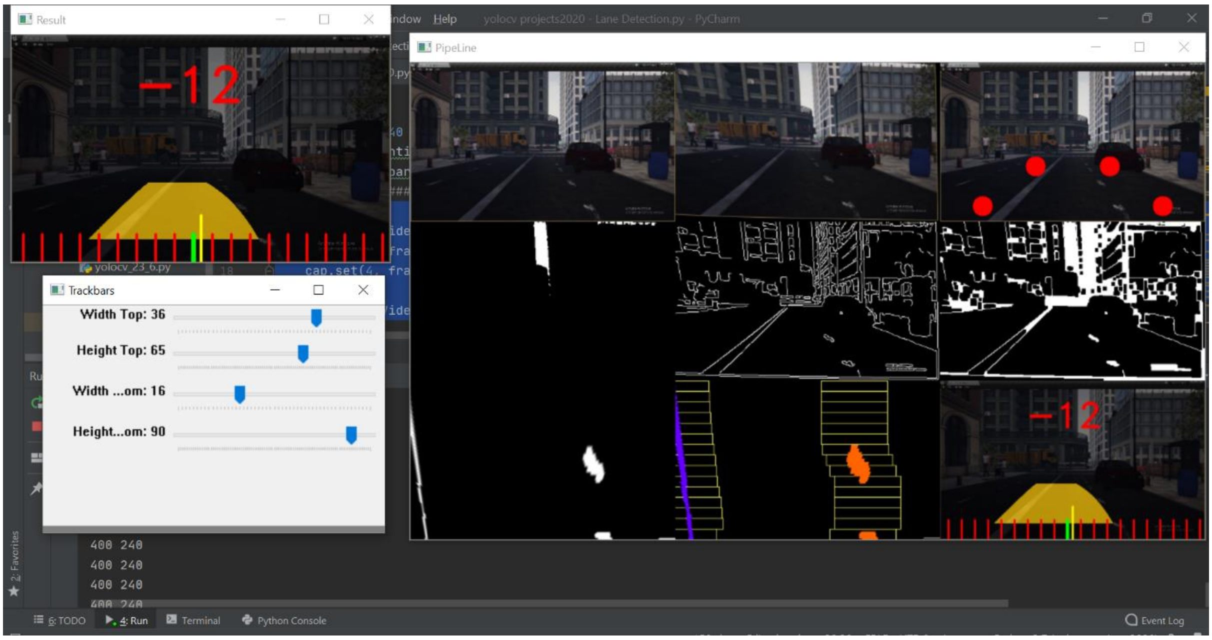


Figure 23: Lane Detection on a generated scenario on CARLA

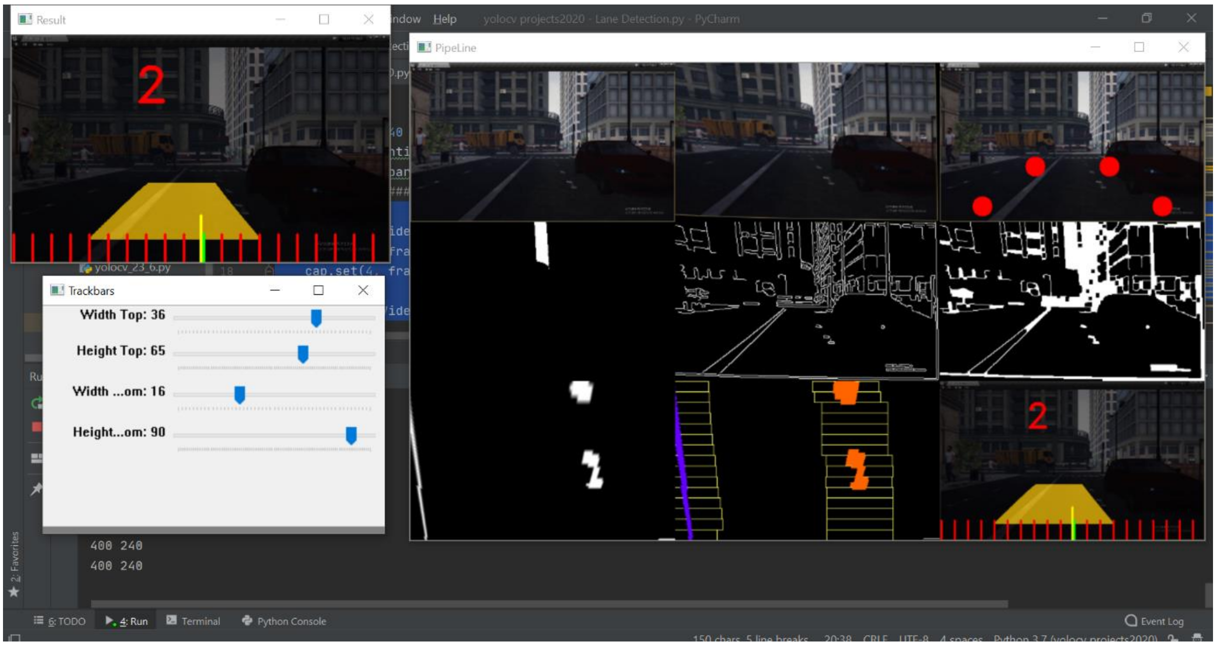


Figure 24: Lane Detection on a generated scenario on CARLA 2

Conclusion

The vehicle industry world-wide has assigned major research and development resources for offering competitive solutions for VB-DAS. Research at academic institutions needs to address future or fundamental tasks, challenges which are not of immediate interest for the vehicle industry, for being able to continue to contribute to this area. The report goes through multiple possible solutions to object on the road detection. We were able to make systems that detect more than 80 objects on the road, also keep track of desired objects such as vehicles other than the Ego-vehicle. In addition to this, we added optional functions to the original goal of the projects, such as number plate and curved lanes detection.

Computer vision can help to solve true problems in society or industry, thus contributing to the prevention of social harms or atrocities.

This project was a tremendously rich experience, where we learned much more than what we could expect. It will be a gate for us to the world of artificial intelligence and autonomous driving cars and related domains.

Computer vision inroad vehicles can play, for example, a major role in reducing casualties in traffic accidents which are counted by hundreds of thousands of people worldwide each year; it is a very satisfying task for a researcher to contribute to improv road safety. Autonomous driving is a realistic goal for some particular traffic situations in developed countries, but not yet expected as a general solution worldwide in the foreseeable future.

VI. References:

[1] https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[2] https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

[3] https://scholar.smu.edu/cgi/viewcontent.cgi?article=1075&context=datasciencereview

[4] https://arxiv.org/pdf/1605.06409.pdf

[5] https://arxiv.org/pdf/1512.02325.pdf%EF%BC%89

[6] https://www.youtube.com/watch?v=gKreZOUi-O0

[7] https://pjreddie.com/media/files/papers/YOLOv3.pdf

[8] https://openaccess.thecvf.com/content_cvpr_2017/papers/Redmon_YOLO9000_Better_Faster_CVPR_2017_paper.pdf

[9] https://pjreddie.com/darknet/

[10] https://cocodataset.org/#overview

[11] https://www.learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/#:~:text=In%20OpenCV%2C%20a%20video%20can,video%20file%20to%20be%20read.

[12] https://project.inria.fr/ppniv18/files/2018/10/paper22.pdf

[13]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6679565/

[14] https://zone.biblio.laurentian.ca/bitstream/10219/2458/1/Peyman%20Alizadeh%20MSc.%20Thesis%20Corrected_2_2.pdf

[15] https://medium.com/analytics-vidhya/custom-object-detection-with-yolov3-8f72fe8ced79

[16] https://medium.com/@duraklefkan/training-yolov3-object-detection-api-with-your-own-dataset-4dcfc7c1c34c

[17] https://towardsdatascience.com/dive-really-deep-into-yolo-v3-a-beginners-guide-9e3d2666280e

[18] https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

[19] https://www.pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/

[20] https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/

[21] https://www.pyimagesearch.com/2018/07/19/opencv-tutorial-a-guide-to-learn-opencv/

[22] https://www.pyimagesearch.com/2019/12/02/opencv-vehicle-detection-tracking-and-speed-estimation/

[23] https://www.researchgate.net/publication/272199860_Vision-based_Driver_Assistance_Systems

[24] https://www.coursera.org/learn/machine-learning

[25] http://neuralnetworksanddeeplearning.com/

[26] https://medium.com/intuitive-deep-learning

[27] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.161.3556&rep=rep1&type=pdf

[28] https://cs231n.github.io/

[29] http://colah.github.io/posts/2015-09-Visual-Information/

[30] https://www.youtube.com/playlist?list=PLqSusG62ek54bVmgER3384oBous6kn4bP