CITY UNIVERSITY
LONDON

Object Detection and Segmentation for Autonomous Cars

Master of Science in Artificial Intelligence

Khalil Adib

Supervised by:

Alex Ter-Sarkisov

October 1, 2022

Declaration

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the assessment instructions and any other relevant programme and module documentation. In submitting this work I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Signed: Khalil Adib

Abstract

In the last decade, the field of deep learning has seen a lot of research and a lot of those research helped in solving real world problem. Until recently, the self-driving cars were just an ambition idea, today, it is our reality, tomorrow they're our future. In this research study, I have tested and evaluated several object detection algorithms to see whether the current state of the art algorithms is good enough to be used in self-driving industry. I did the experiments on two different GPU hardware. Between all the models I have tested, the new State-of-the–art model Yolov7 model outperformed all the other models. I took into consideration the speed and the accuracy of the models. On the other hand, I employed the transfer learning and I combined two different annotations to see whether we can run instance segmentation on BDD100K dataset using MaskRCNN, which nobody did before. The results were good as a starting point.

The source code of this projects with the detailed instruction can be found in the project's GitHub page:

https://github.com/Khaliladib11/MSc_Thesis

Keywords: Self-driving cars, Deep Learning, Computer Vision, Object Detection, Segmentation.

Acknowledgement

First, I would like to thank my supervisor Dr. Alex Ter-Sarkisov for his valuable insights and guidance for the duration of this project and for introducing me to the magnificent world of computer vision. I wish to acknowledge all my City University lecturers who have contributed to my current level of knowledge and who made my year of study a very interesting experience. Further, I would like to express my appreciation to people providing free online study materials and courses that helped me to understand many topics. My grateful thanks also extend to people who made their research open source to the rest of the world so everyone can benefit from it.

I wish to express my gratitude to my family and friends for their endless support and encouragement. Finally, a special thanks to my uncle Alex and his wife Agnieszka, also to my uncle Mike and his wife Maha.

Table of Contents

List of Tables

List of Figures

List of Equations

1. Introduction

1.1 Problem and Motivation

In recent years, automation industry has expanded and we can see its applications in every corner in our lives. Autonomous vehicle is one of these applications, which is the major topic of my dissertation. Self-driving cars (also known as autonomous cars or driverless cars) is a vehicle that uses multiple sensors, LIDARs and cameras to autonomously and intelligently travel from source to destination without human interference.

The Society of Automotive Engineer (SAE) defines six levels of driving automation [1], ranging from 0 (fully manual) to level 5 (fully autonomous) (See Figure 1). Autonomous cars heavily rely on sensors, actuators, algorithms, machine learning systems, and huge computing power to execute software.

Many companies have already started to create self-driving cars, maybe the most famous one is Tesla. Currently these vehicles are under test, we haven't achieved the full autonomous driving level yet.

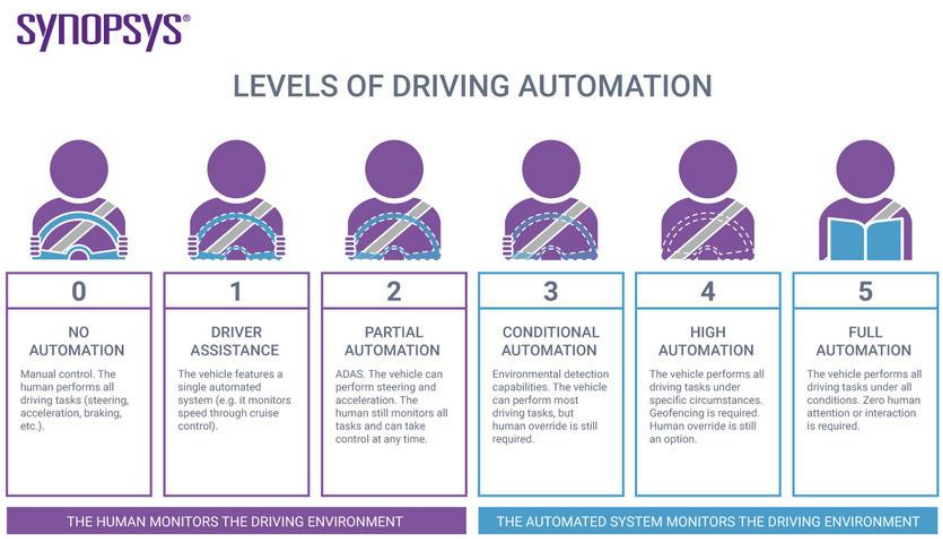


Figure 1 The six levels of driving automation

Computational power has risen dramatically over the last several decades (due to GPUs and TPUs), and massive datasets are now available to the public. These advancements catapulted artificial intelligence into the automotive sector, hastening the development of level 4 and 5 of autonomous vehicles. Indeed, many in the transportation industry have already recognized AI's enormous potential, with the worldwide market expected to reach $4 billion by 2026 [2].

Such spending will help the academia and industry to leverage advanced techniques in machine learning and computer vision especially, to shape the future of transportation so that passenger safety increases, road accidents are lessened, and traffic congestion is reduced.

The notion of self-driving cars is not new; it was first introduced by General Motors in 1939. However, we may apply Deep Learning to develop intelligent systems that can make sense of their surroundings by employing computer vision techniques such as object detection and segmentation.

There are a few companies that use only vision-based approach in their cars, Tesla being one of these companies. When we say vision-based that means they are using data coming from images only without any additional sensors like LIDARS. Using these images, we can train a model to perform object detection and segmentation tasks to help the car to navigate. From technical point of view this approach is more complicated than the others, nevertheless, by attaining full automation using just vision, we will have a generic vision system that can be deployed everywhere.

The primary research question for this study is:

"Are the current object detection algorithms efficient enough to be used in self driving cars?"

1.2 Objectives, Project Product, and Beneficiaries

My objective in this project is to research, analyze, and evaluate several computer vision algorithms that might be employed in self-driving vehicles, such as object detection and segmentation. As a result, the project will produce a fully functional framework for many computer vision tasks, particularly in the self-driving car industry.

Thus, the project deliverables are as follows:

- A detailed comparison between several object detection algorithms for multi-class detection and classification.
- Results of running Mask RCNN model to detect objects and segments the roads.

This work can benefit practitioners in a variety of sectors as well as researchers. People and researchers in the self-driving vehicle sector can employ this project to implement current state-of-the-art models in their self-driving car prototypes. Engineers in the robotics sector can use this project to include it into their robots. This project can help anyone with a fundamental foundation in artificial intelligence and machine learning who are interested in computer vision to expand their expertise in the field.

1.3 Structure of the Report

In this section, I introduced my project and stated why I decided and what motivates me to work on this subject. I also outlined my goals and the projects' objectives, as well as the parties that may profit from the work.

In section 2, I will start with a board introduction to the field of machine learning, state what are the types of machine learning, then talk about the limitation of machine learning. Following that, I'll cover deep learning, introduce the area of computer vision, and go over the relevant algorithms.

In section 3, I will talk about the method and the methodology I followed. From data collection, to EDA phase, moving to models selection and evaluation metric.

In section 4 and 5, the results of the experiments are presented using figures, diagrams and graphs, followed by a discussion to interpret the results.

Finally in section 6, I will assess my performance and go through the research question and the project's objectives. After that, I will list some ideas to extend this work. Then I will finish with a small reflection and conclusion.

2. Literature Review

2.1 Machine Learning

2.1.1 Definition

Deep learning is the deepest part of machine learning. To fully comprehend deep learning, we must first grasp the fundamental principle of machine learning.

Machine learning algorithm is a tool to convert information into knowledge, in other words, it's an algorithm that can learn from data. But what exactly do we mean by **"learning"?** Machine learning algorithms are capable to discover the hidden patterns in structured and unstructured data. Using these hidden patterns, we can generate some predictions and perform some kind of decision making.

Mitchell's (1997) description in his book Machine Learning [3] may be the most formal definition so far:

*"A computer program is said to learn from experience **E** with respect to some class **T** and performance **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**."*

At first glance, this definition can be confusing, but it's not! In order to build a machine learning algorithm, we must define the three main components which are the task **T**, the experience **E** and the performance measure **P**.

To begin with, the main goal of machine learning algorithms is to perform tasks that cannot be explicitly program or designed by a human. These algorithms are now employed in almost every aspect of our life. For example, song streaming applications like Spotify recommend new songs based on our previous songs and other people's favorite songs; similarly, online shopping stores suggest things that we are more likely to acquire. Furthermore, text translation is an example of an intelligent system that can recognize semantics in a text and translate it to another language. Moreover, self-driving cars employ machine learning to make sense of their surroundings and drive safely. These are some examples of tasks that machine learning can perform.

On the other hand, the second component of a machine learning system is the experience **E**, which can be described by a list of many examples called dataset. Without dataset, it is nearly impossible to make any machine learn anything. The size of the dataset depends of the type of task. One of the oldest datasets studied by statisticians and researcher is the Iris Dataset [4]. It is a collection of 150 row data, each row has four features and one label, the label can be one of three different classes. Another example of a dataset is MSCOCO [5] dataset which is a collection of images companied with annotations that can be used for different tasks in vision domain.

Finally, the performance measure P is the final component of the machine learning algorithm. To assess an algorithm, we must first develop a mathematical function that will provide us with an indicator of how good or bad the algorithm is. There is no such universal performance measure that can be used for all tasks, usually this measure is dependent on the task. For instance, in a classification problem sometimes the accuracy metric is a good indicator, now the accuracy metric is just a percentage of example of which the algorithm predicts the correct output. Another example of a performance measure is the mean squared error (MSE) that is used when the output is continues values, not discrete and predefined. Usually we are interested in how well the algorithm performs on a data that has not seen before, in other words on test set.

2.1.2 Machine Learning Approaches

Moving to talk about the types or approaches of machine learning. Usually any machine learning model is listed under of the following types:

- Supervised learning,
- Unsupervised learning,
- Semi-supervised learning,
- Reinforcement learning.

According to the **no free lunch Theorem** for machine learning [6], there is no machine learning algorithm that is universally any better than the other. Therefore, there are algorithms to suit some specific problems.

Let's start with **supervised learning**, in this type of learning the goal is to learn the relation between an input **X** called features to an output **Y** called label or target**.** The term supervised comes from the idea that the data must be fully labeled, in other words, the **Y** values must be provided by a human. For instance, the Iris dataset example that I mentioned earlier in section 2.1.1, each row data is provided with four features and one output which is the type of the flower. In this type of learning, the machine is taught by examples and a lot of examples. This is the most common type of machine learning algorithms and most of the real world applications nowadays are categorized as supervised learning.

In addition to that, the supervised learning can be divided into two kinds, classification and regression. The difference between those two kinds is in classification the model is trying to predict a class from a predefined discrete set of classes. On the other hand, in regression the model is trying to predict a continues number (i.e. predict the price of a house).

One of the most famous and powerful supervised learning algorithms is the **support vector machine (SVM)** [7]. This algorithm can be used for classification and regression. Other examples of supervised learning algorithms are Decision tree, Logistic and Linear Regression, Naïve Bayes, etc…

On the other hand, in **unsupervised learning** approach the features **X** is the only input for the algorithm, there are no labels or targets associated with it. Unsupervised learning tries to extract information from a distribution without the help of a human. One of the algorithm in unsupervised learning is the **Principle component analysis (PCA)** [8]**,** that learns a kind of linear projection. The most used algorithm in unsupervised learning is the **Clustering**. Basically, clustering attends to find subgroups within a dataset.

One may ask, where can we use unsupervised learning? Well, we can use it to segment a customer based on their preferences. Another application is anomaly detection where we attend to identify the unusual events. Finally, one of its most famous application is dimensionally reduction to find the most important features in a dataset.

Moving on to **semi-supervised learning**, which is mix between supervised and unsupervised learning. In this kind of learning we provide the algorithm with a small labeled dataset and a lot of unlabeled data.

Finally, the last approach is the **reinforcement learning** where we attend to train an agent to learn through trials and errors. It learns from past experiences. However, this type needs a lot of experiments in order to give a decent result. Here is a fun fact: the **Alpha Go agent** [9] who defeated the best Go player in the world played more games than the whole Go players in the world.

2.1.3 Limitation of Machine Learning

The machine learning algorithms that I have mentioned above are essential and solve many problems. However, they didn't succeed in much complex and more exiting problems in AI such as image classification, object detection, speech recognition, etc...

Some of the main challenges that faced the traditional machine learning algorithm is the **curse of dimensionality** [10]. This phenomenon appears when the number of dimensions in the data is high. For instance, let's say we want to classify a 250×250 image, then the input features will be 62500 which is very big number for traditional machine learning algorithm. However, 250×250 image is very poor resolution. Also, if we want to classify a video with 10 FPS, so the number of input features will be 10×250×250.

Another problem in the traditional learning algorithm is that they were linear functions, so they faced a lot of problems dealing with non-linear functions like **XOR**.

In the end, all these failures motived the development of Deep Learning to create more intelligence and generalize AI systems.

2.2 Deep Learning

Deep learning is a subfield of machine learning, where we use deep neural networks to perform very complex tasks. The term neural network isn't new at all, in 1958, an American psychologist called **Frank Rosenblatt** built a machine that can sense, recognize, remember and respond like a human, this machine called **Perceptron** [11].

The perceptron is a linear classifier consists of single layer. It takes as input a metric of values, and give us a binary output (0 or 1) (see Figure 2). The main motivation behind perceptron were the real neurons in our brain, however, that doesn't mean they work the same way.

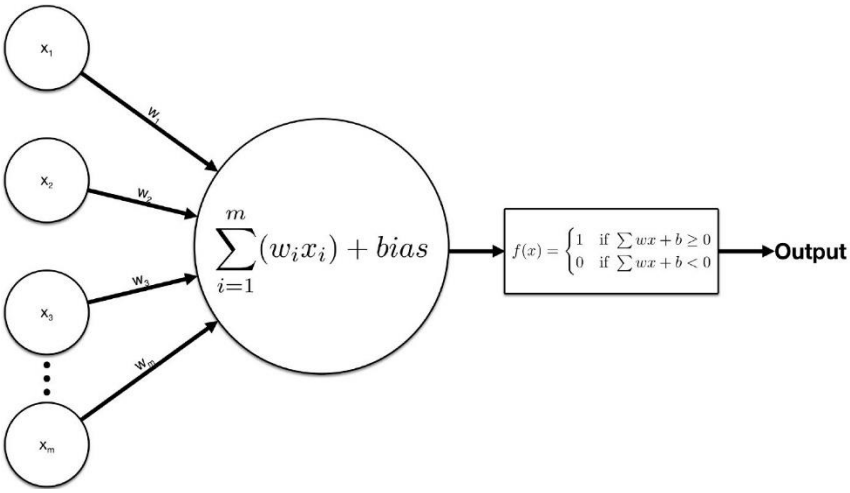


Figure 2 Perceptron

2.2.1. Feedforward Network

Feedforward networks [12] also known as multilayer perceptron are the heart of deep learning. Essentially, the goal of these networks is to approximate some function $\boldsymbol{f}^*$. The feedforward networks try to learn a non-linear and complex relation between an input **x** and output **y,** by defined a mapping $\boldsymbol{y = f(x, \theta)}$ where **θ** is the value of parameter that represents the best approximation.

Feedforward networks called **networks** because we are using several layers chained and connected with each other. The length of the layers represents the **depth** of the network, and from this point the name **Deep** comes to the title. Moreover, the word **neural** comes from the inspiration from the brain and neuroscience as I mentioned above.

The deep neural network consists of three main parts. First of all, the input layer, where we input a vector of points to the network. The second part is the hidden layers, each layer of the network consists of several nodes called **units** that work in parallel, the dimensionality

of these hidden layers represents the **width** of the model. Lastly, the last layer or the output layer returns the results. The length of input and output vectors are fixed, while the depth and width of the hidden layers can vary. Figure 3 illustrates a deep neural network with three hidden layers.Figure 3 Deep Neural Network

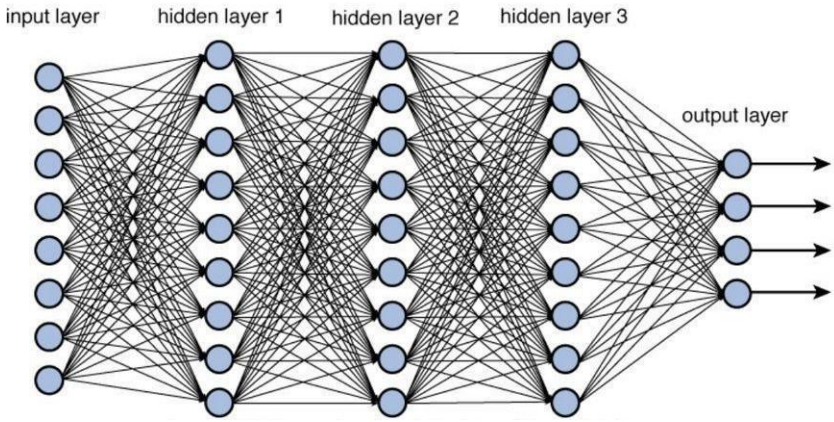


Figure 3 Deep Neural Network

If we want to take a closer look at what is happening inside the neural network, we find that at each layer, we are multiplying the vector from the previous layer with the weights that connect both layers, then we add a small amount called bias. After that we apply a mathematical function called activation function. To simplify things, the output of each layer is as follows: $\boldsymbol{a = g\ (x.w^T + b).}$

The role of activation function is to introduce non-linearity and to learn the complex patterns from the data. The default recommendation for activation function inside the hidden unit is the rectified linear unit (**ReLU**) [13] activation function. Typically, **ReLU** is defined by **g(z) = max{0, z}** as we can see in Figure 4**.**

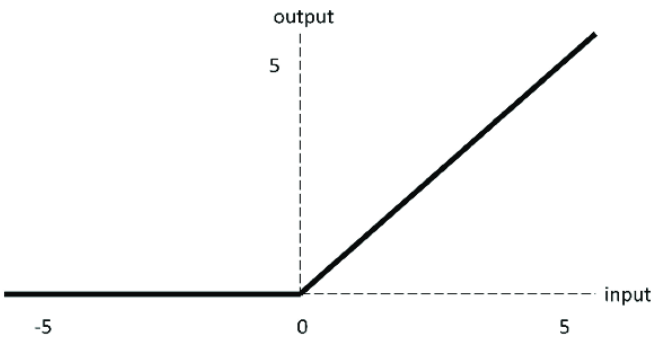


Figure 4 Rectified Linear Unit (ReLU) activation function

At the output layer, usually we use different type of activation function. One kind of these function is called **Sigmoid** [14] activation function, it returns a set a values, each of them

represents a probability distribution and the one with the highest probability is the class the network predicted. On the other hand, there is **Softmax** [15], which returns a set a probability distribution as well, however, the sum of all probability here will be equal to one.

2.2.2. Backpropagation

After passing the value through feedforward pass and calculating the cost, it's time to go backwards now, using an algorithm called **backpropagation** [16]. Backpropagation is simply computing the derivatives of the gradients using the chain rule, as simple as that.

By calculating the derivative of each gradient, we can now update the weights of the network in such a way to minimize the cost function.

Figure 5 illusrates the whole process of training a standar neural network. We start by passing the input to the network in the forward pass. Then we calculate the cost function. Finally we go backward and update the weights using backpropagation algorithm. Repeat this process until convergence.

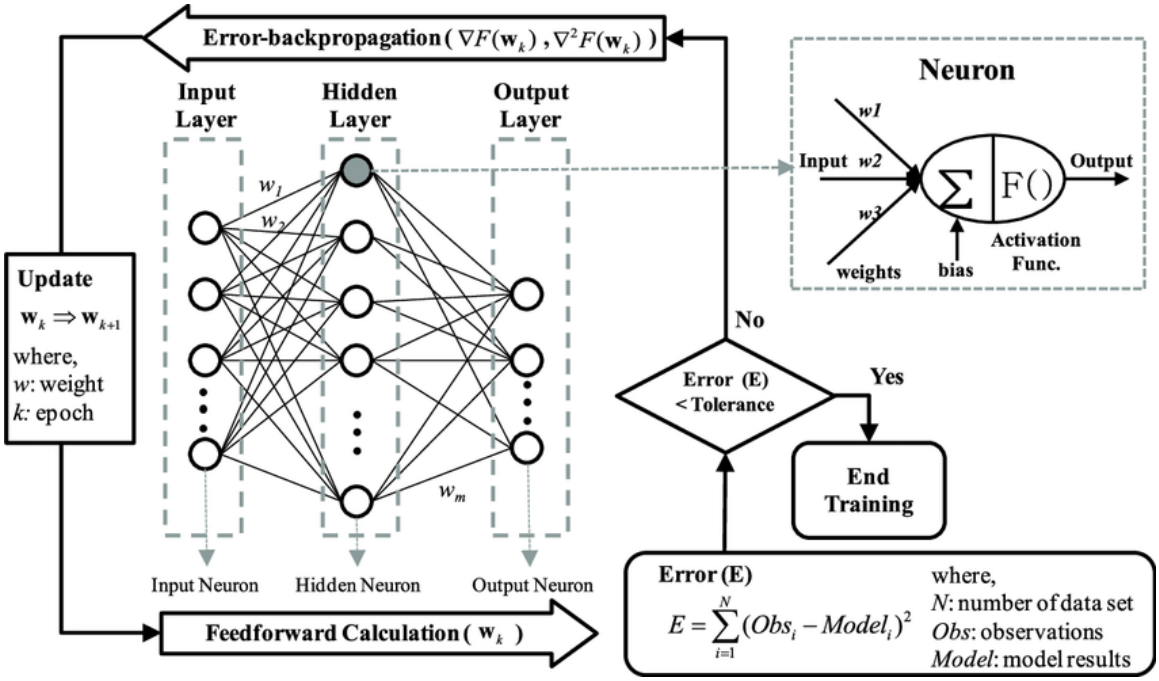


Figure 5 Training Process of a Neural Network

2.2.3. Regularization

One of the most famous problems in machine learning and deep learning is overfitting. Overfitting occurs when the model performs exceptionally well on training set but poorly on test set. This can happen when the model starts to learn the actual training data instead of generalize. On the other hand, there is underfitting, where the model doesn't perform well on both sides.

When training any model, the goal is always to generalize, and to give the best result on testing set. To do that, regularization techniques are usually used.

One of the well-known regularization method is to add a new term to the cost function that penalizing the weight like we can see in Equation 1.

$$J(\theta) = \frac{1}{n}\sum_{i=0}^{n} Loss(y_i, \hat{y}_i) + \frac{\lambda}{2n}\|\theta^2\|$$

Equation 1 Loss Function with Regularization Term

We see how the regularization parameter **λ** is used to control the effects of weight on the final objective function. We should choose the value of λ carefully because sometimes if we chose the wrong value we can end up in the underfitting region.

Another technique that can be used to avoid overfitting is **dropout** [17]. The main idea of dropout is to randomly dropping some units by ignoring them in the feedforward and backward propagation. This can help in forcing the units to not rely on any specific units (see Figure 6).

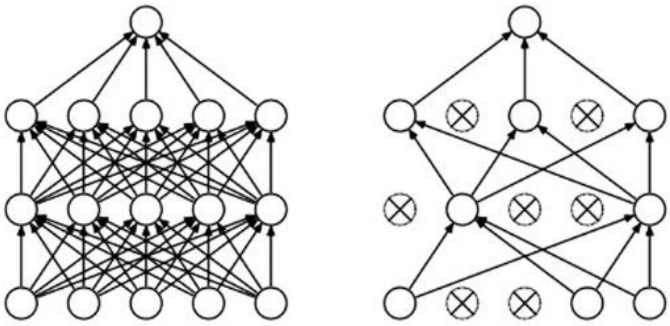


Figure 6 Example of Dropout

Additionally, data augmentation is one of the most wildly used techniques to avoid overfitting. As we know, the more data we have the better. We can apply some techniques to generate more data from the data available with us, because the data is usually very limited. For instance, if we want to work on a simple image classification problem, we can apply methods like rotation, cropping, padding, adding some noise, etc… (see Figure 7 for example). In this way we can help the model to generalize by expose him to different scenarios.

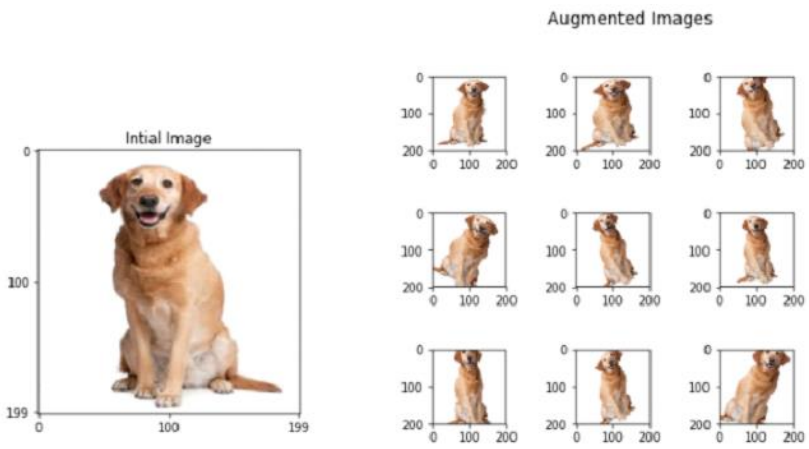


Figure 7 Example of Data Augmentation

2.2.4. Optimization

The goal of any deep learning model is to generalize well and achieve the lowest loss possible while avoiding overfitting. Through the years, the researchers worked hard to make the neural network robust, they spent too much time on optimizing the algorithms that we are using in neural networks.

When we talk about optimization, the main goal is to find the right parameter **θ** in order to reduce the cost function or the loss.

Gradient Descent [18] was one of the first optimizers and the most basic one. It is a first order optimizer which means it depend on the first order derivative of the loss function. Gradient Descent uses the backpropagation algorithm to update the gradients of the network.

What gradient descent does is it feed the rows data to the network at once, calculate the loss, then update the gradients using the Equation 2. We should note here that $\boldsymbol{\alpha}$ is a hyperparameter called learning rate, which represents the size of the step at each iteration.

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta)$$

Equation 2 Gradient Descent update equation

Gradient Decent has some disadvantages, one of them is it requires a large memory to fit the whole data at once. In addition, it takes a lot of times to converges because It updates the gradient one time every epoch. Finally, it can be easily trap into local minima.

Sophisticated gradient Descent (SGD) [19] however, it takes one row of data at a time then update (see Equation 3). In other words, if we have 1000 row data, it updates the gradients 1000 times.

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta, x_i, y_i)$$

Equation 3 SGD update equation

SGD requires less memory obviously, however, it takes more time to converge!

That's why Min-batch [19] came into the picture, where we divide the dataset into group of batches and update the weights every batch (see Equation 4). For instance, if we have 1000 row data and we select a batch size of 200, then we update the weight only 5 times.

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta, B_i)$$

Equation 4 Mini-Batch update equation

In Figure 8, we can see the difference between those algorithms.

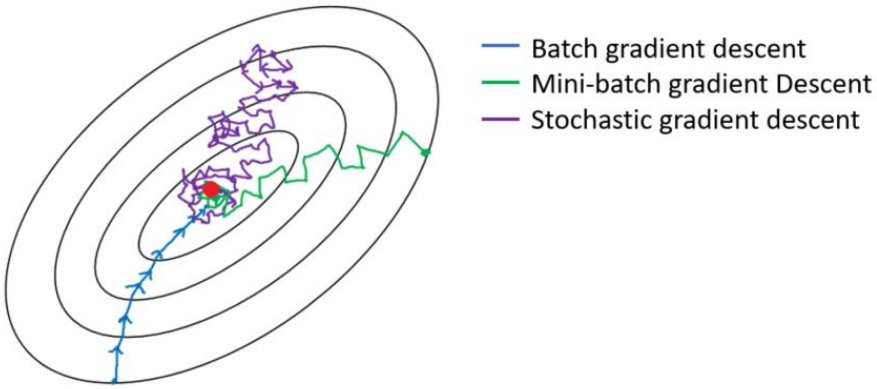


Figure 8 Comparison between Gradient descent, SGD and Mini-Batch algorithms

All of the above algorithms faced the same problem, which is the local minima. In addition, the learning rate is fixed in all these algorithms, which can cause a problem because sometimes we don't want to make changes at the same rate.

To solve these problems, several optimizers have been proposed. **Momentum** [20] reduces the high variance of SGD and speed up the conversance. **AdaGrad** [21] solved the problem of fixed learning rate. Finally, **Adam** [22] optimizer rectifies vanishing learning rate and high variance, while maintain fast convergence time.

In conclusion, Adam optimizer is the best optimizer so far as we can see in Figure 9.

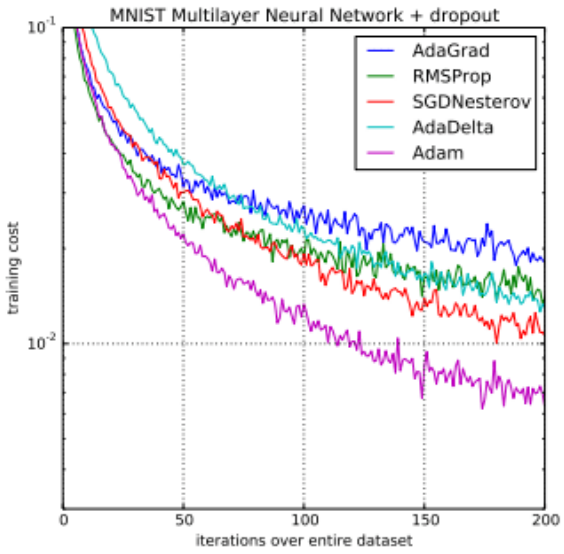


Figure 9 Comparison between different optimization algorithms

2.2.5. Transfer Learning

Deep neural networks take a lot of time to train, because they are so deep! So it is inconvenient to train the model from scratch even if we want to perform different tasks. In addition, deep neural networks usually require a lot of data to train on, the large dataset aren't usually available to public. To overcome these problems, **Transfer Learning** came in to facilitate our lives!

Basically, transfer learning is an optimization method where we use pre-trained weights on very large dataset as a starting point, instead of training from scratch. Moreover, by applying transfer learning, we can achieve higher accuracy than training only on the data that we have.

Nowadays it is very rare to train everything from scratch, researcher and scientist prefer to start from pre trained model.

Figure 10 shows the difference between training a traditional model vs using transfer learning. On the left hand side of the picture we can see how we are training each task from scratch separately. On the right hand side, we can see how in Transfer learning we use knowledge collected from another task to learn a new task.

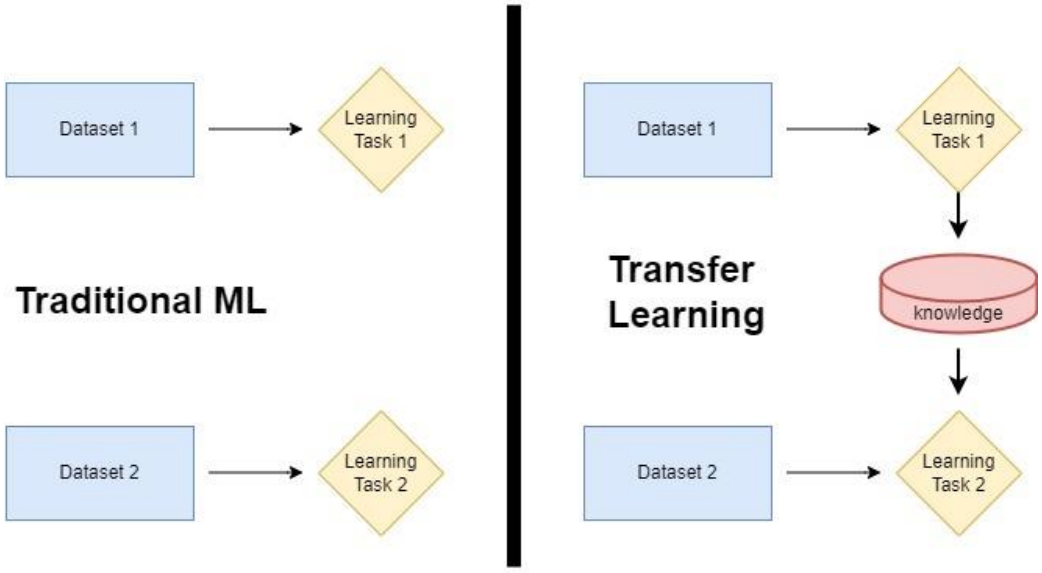


Figure 10 Difference between Traditional Machine Learning and Transfer Learning

In this project, I will be using transfer learning a lot.

2.3 Computer Vision

Computer vision was and still one of the most active area of research for deep learning applications. Vision is a simple task for humans and most animals but it is very challenging task for computers.

Computer vision has a wide range of applications that are used and employed in the real world. Application of computer vision range from simple image classification to complex object detection and segmentation tasks. The ambition behind computer vision is to create vision intelligence software that can be accurate and fast as human vison, to replace human in some tasks.

Computer Vision can be categorized as a sub-field of artificial intelligence and machine learning. It is different from image processing which focus on image manipulation or enhancing visual information rather than focusing on the actual content of the images and videos.

The majority of computer vision tasks require a large amount of images and/or videos in order to give a decent result in real-time.

Application of computer vision include image classification [23], object detection [24], semantic segmentation [25], instance segmentation [26], object tracking [27], 3D object rendering [28], etc...

2.4 Convolutional Neural Networks

The **convolution neural networks** [29] and also known as **CNNs** are the most powerful kind of neural network in analyzing and extracting information from images. Convolutional neural networks have been extremely successful in the computer vision field.

The CNNs are specialized in handling grid like topology such as images. An image is represented by a set of pixels arranged in a grid fashion, each pixel represents the brightness and the color of its position.

2.4.1 CNN Structure

The structure of a convolutional neural network typically consists of three main parts:

- Convolutional Layer.
- Pooling Layer.
- Fully Connected Layer.

The notion convolution comes from the special kind of linear operation called convolution as well, where we replace the stander metric multiplication in the neural network by a convolutional layer in at least one of their layers.

Mathematically, the convolution is defined as follows:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da$$

Equation 5 Mathematical formula of convolution

Equation 5 describe how the signal $x(a)$ is weighted with $w(t-a)$. The function $w(t-a)$ plays the role of a filter applied to the signal $x(a)$, this filter is learned during the training process.

Usually, when we want to apply Equation 5 on an image, the input is a 2D or 3D array so we can replace the integral with summation. Example of 2D array convolution:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(m,n)\ K(i-m, j-n)$$

Equation 6 Two-dimensional Kernel in convolution

Where *I* represent the image and *K* the filter or the kernel in Equation 6.

Moreover, Convolution is cumulative, resulting:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)\ K(m,n)$$

Equation 7 Commutative Convolution

Equation 7 can be described as a kernel applied at multiple regions in the image resulting a mapping from input image to output image, as we can see in Figure 11.

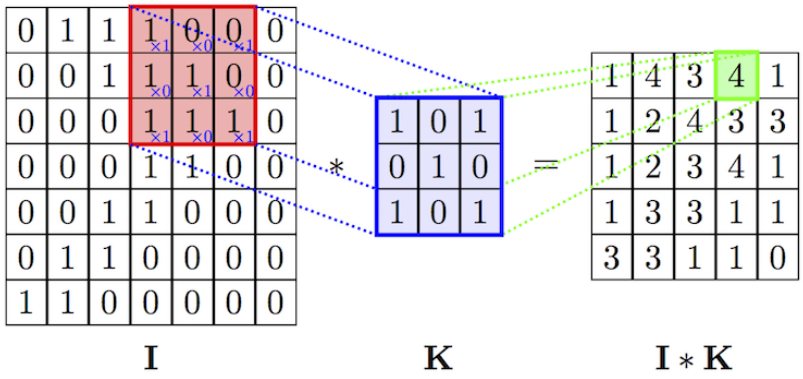


Figure 11 Illustration of the convolution operation between an image I and Kernel K

The size of the output image depends on the width and height of the kernel, as well as a parameter called stride which defined the number of pixel the kernel moves between each convolutional iteration. As a result, we will have the following equation:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i*s-m, j*s-n)\ K(m,n)$$

Equation 8 Convolution with Stride

Additionally, in order to maintain the image size, padding is used when the kernel is close to the borders of the image. There are few options to handle this case, one of them is to add zeros on the borders.

As a result, we can calculate the output size of the image using the following equation:

$$o = \left[\frac{i + 2p - k}{s}\right] + 1$$

Equation 9 Size of the output after convolution

Where ***o, i, k, p*** and ***s*** represent the output shape, input shape, kernel size, number of zeros padded and the stride step size respectively, as we can see in Equation 9.

Moving to talk about the pooling layer. The pooling layer is simply a mathematical function replace the output of the convolution layer by a summary statics of the nearby outputs in order to reduce the dimensions.

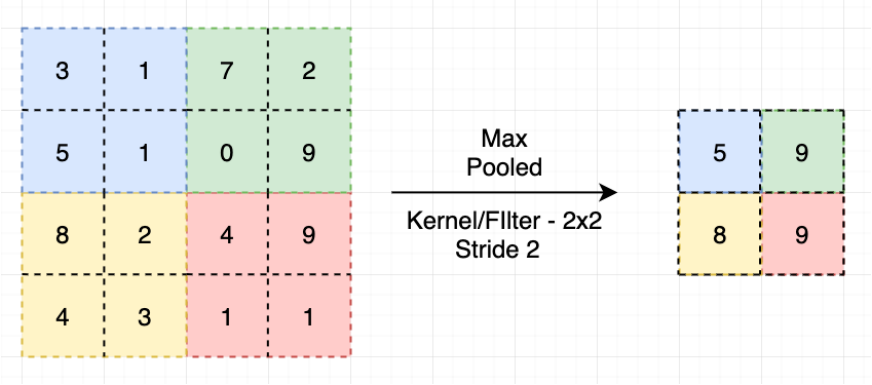


Figure 12 Example of Max-Pooling operation

There are multiple types of pooling can be applied. For instance, in Figure 12, max pooling is applied, where we take the maximum within a rectangular kernel. The output of the convolution and pooling layer usually called a feature map.

Similar to Equation 9, the number of output of a pooling layer can be calculated using the following equation:

$$o = \left[\frac{i-k}{s}\right] + 1$$

Equation 10 Output of a pooling layer

Finally, a fully connected layer is placed at the end of the network to give us the final output. Figure 13 illustrates in convolutional neural network architecture consists of two convolutional layer with max polling layers, followed by a fully connected layer to classify the number in the input image.

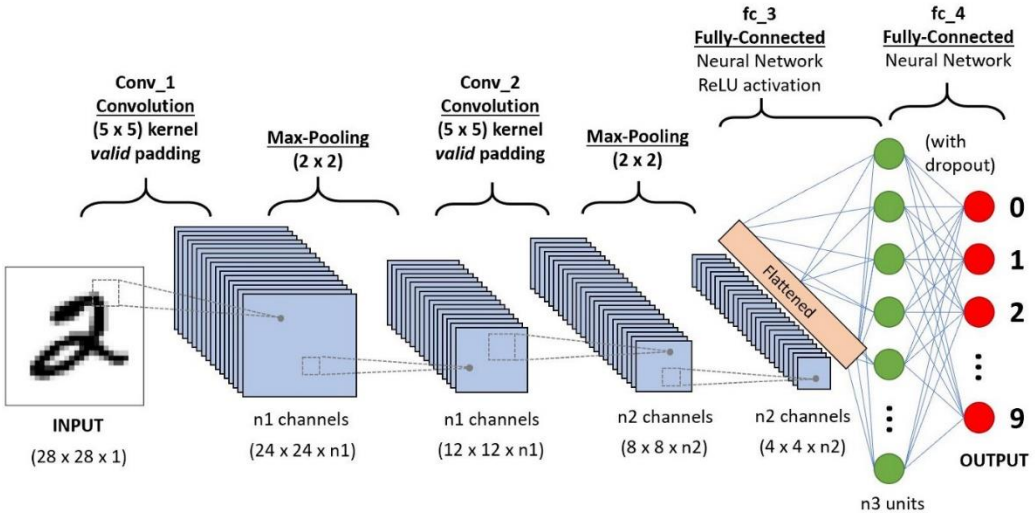


Figure 13 Example of a convolutional neural network

2.4.2 CNN Architecture

There are some famous CNN architectures used in the literature nowadays. Usually these architectures are used as a part of a bigger model and they are called feature extractors.

To begin with, **VGGNet** [30] is a type of convolutional networks that consists of several layers. There are various architectures from this network such as VGG-11, VGG-13, VGG-16 and VGG-19, where the number represents the number of convolutional and fully connected layers. For instance, VGG-16 has 13 convolutional layers and 3 fully connected layers resulting 16 layers, as we can see in Figure 14. The number of parameter in this network is proximally 135K parameters.

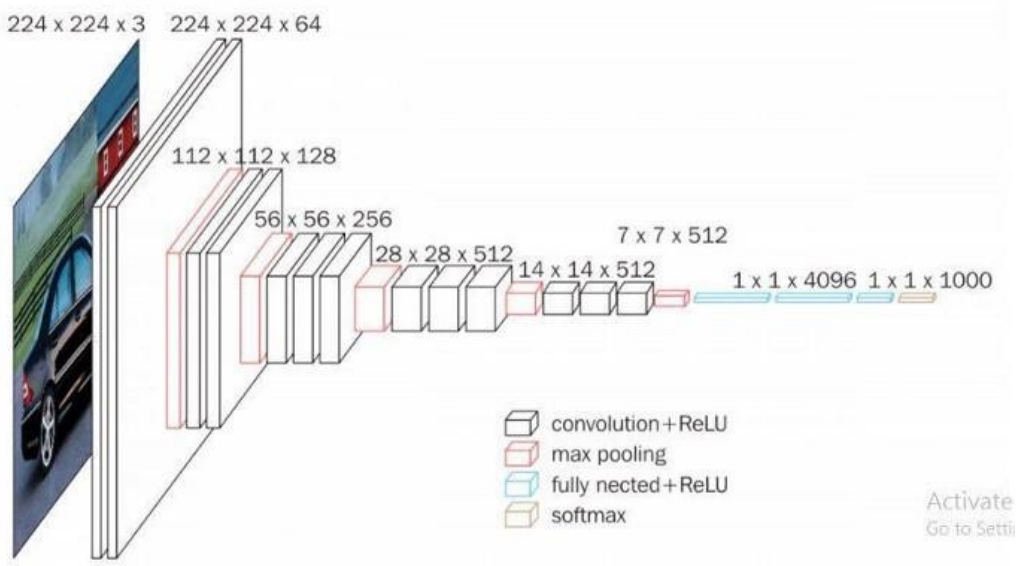


Figure 14 VGG16 Architecture

Another type of architectures called **ResNet** [31]**.** Researchers around the world agreed that the deeper the network the better, however they have noticed that the models started to lose the generalization capabilities as the network deepened. The motivation behind Resnet networks is to solve the problem of vanishing gradient. Basically, as the network deepened, the gradients start to shirk to zeros after several applications of the chain rules, leading to the situation where the weights stop updating, therefore, no learning occurs.

Now using Resnet network, the gradient can flows directly using the skip connections as illustrated in Figure 15.

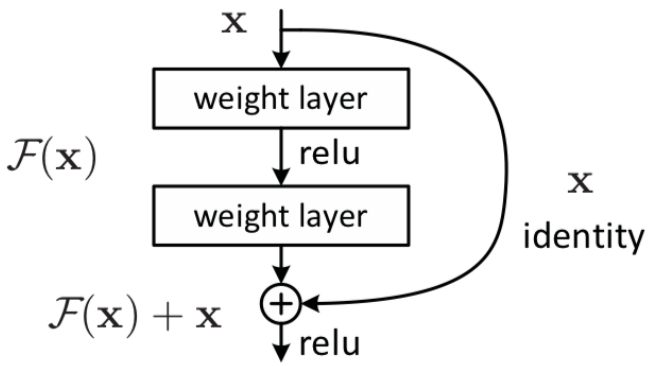


Figure 15 Residual Learning: a building block

Like in VGGNet, ResNet has various architecture as well. See Table 1 for more details.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1 Detailed architecture of the ResNet networks for ImageNet..

The last architecture I will talk about in this report is **MobileNet** [32]. MobileNet, as the name suggest, is designed to work on mobiles. MobileNet uses depthwise separable convolution to reduce the amount of parameters in the network when compared to a normal convolutional network with the same depth. This leads to a lightweight deep convolutional neural network.

The difference between stander convolution and depthwise separable convolution is in depthwise convolution we replace the 3x3 convolutional layer with 3x3 depthwise convolution followed by batchnorm, ReLU and 1x1 pointwise convolution as we can see in Figure 16.

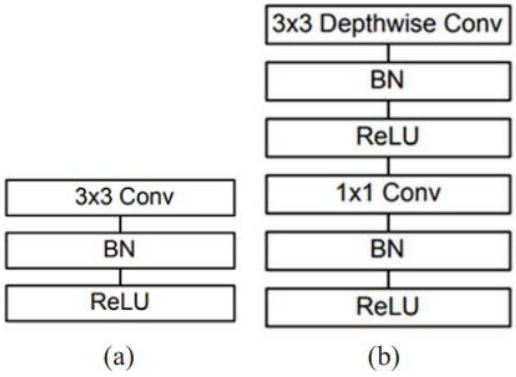


Figure 16 (a) Stander convolutional layer with batchnorm and ReLU activation. (b) Depthwise separable convolutions with depthwise and pointwise layers followed by batchnorm and ReLU activation.

2.5 Object Detection

One of the most exciting and challenging application in computer vision field is object detection. Object detection is the process of identifying and localizing one or more object in images. This can be achieved by drawing a bounding box around the object accompanied with a class label for the object, as we can see in Figure 17.

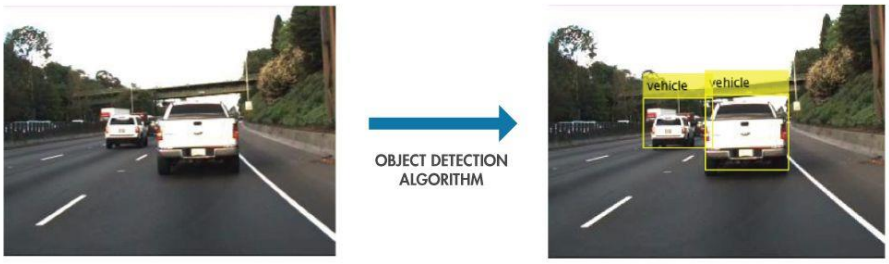


Figure 17 Example of Object Detection

We cannot solve the object detection problem by building a traditional convolutional neural network followed by fully connected layer simply because the output is not fixed. In other words, in each image the number of objects we are interested to detect is not fixed and can varies. Therefore, numerous algorithms using many different architectures have been proposed to solve this problem, and we can categorize them into two categories:

- Two-stage detectors or Region based Convolution (e.g. R-CNN family).
- One-stage detectors (e.g. YOLO, SSD).

In the following sections, I will explain a bit about those algorithms and how they work.

Before proceeding, most of the object detection algorithms consist of two main components. A backbone that plays the role of feature extractor to extract visual feature from images. A head responsible for detection operation, it can be dense prediction for one stage detectors and spare prediction for two stage detector. Some algorithms use a middle

network between the head and the backbone called neck, this helps to collect feature maps from different stages. Figure 18 shows it all.

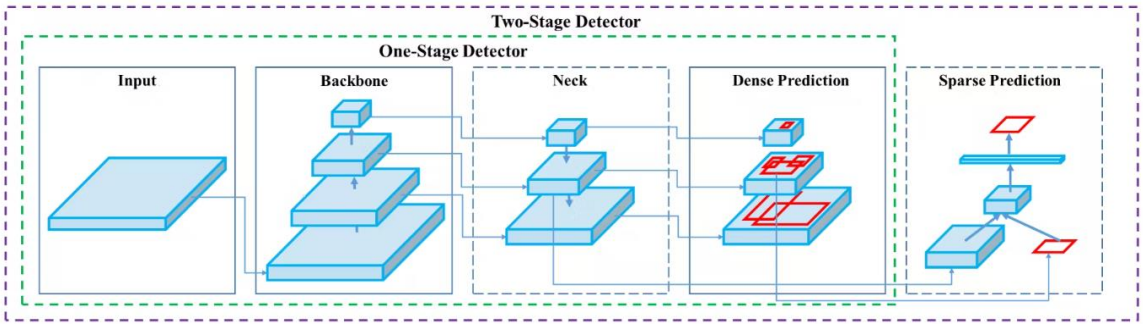


Figure 18 General architecture for object detection algorithms, taken from [33]

Note: In the following sections I will use terminologies such as IOU (intersection over union) and mAP (mean average precession). Those are some metrics used to evaluate object detection algorithms. I will explain more about these metrics in section 2.7. For now, you have to know that the higher those metrics are the better.

2.5.1 Region-Based Object Detectors

RCNN

The first version of these detectors called **R-CNN** [34]**,** which combines the CNN with region of interest. First, this algorithm starts extracting 2000 regions of interest (ROIs) from an image (where an object can potentially be present) using **selective search algorithm** (see stage 2 in Figure 19)**.** These 2000 region proposals are fed into a CNN network that plays the role of feature extractor to return extracted features (see stage 3 in Figure 19). After that, the extracted features are fed into an **SVM algorithm** to classify the presence on an object within the region proposal (see stage 4 in Figure 19).

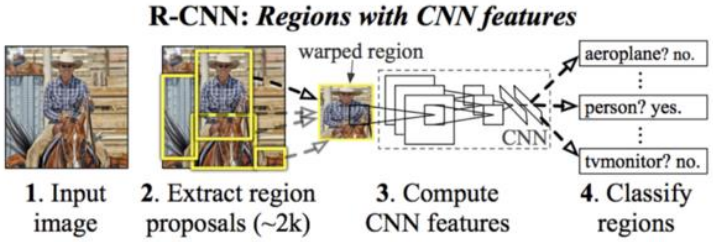


Figure 19 Illustration of R-CNN architecture, taken from [34]

R-CNN algorithm faced some disadvantages. During training, the algorithm uses 2000 ROIs per image, which requires a lot of time to train. Additionally, it is far from real time, as it takes around 47 seconds per image at inference. Moreover, there is no learning at extracting ROIs stage, since the selective search process is fixed, which leads to bad ROIs.

Fast RCNN

The same author of [34] came with some solutions for the above drawbacks and he published a new paper under the name of **Fast R-CNN** [35]**.** Fast R-CNN followed the same approach like in [34], but instead of feeding the ROIs to the CNN, we fed the image directly to the CNN to generate a feature map. From the feature map, we identify the region of interest then we pass them to a pooling layer. The role of ROI pooling layer is to reshape the ROI to a fixed size. After that, the output of the pooling layer is passed to a fully connected layers, then a softmax layer is responsible for predicting the class of the object and the bounding box regressor is used to return the offset values, as illustrated in Figure 20.

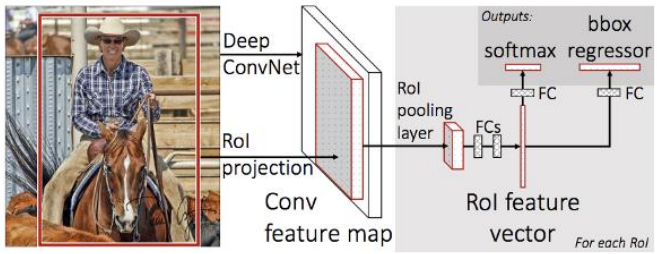


Figure 20 Illustration of Fast R-CNN architecture, taken from [35]

Fast RCNN reduced the training and inferencing time significantly. However, this algorithm is still depending on selective search.

Faster RCNN

Finally, in 2015, **Faster RCNN** [36] eliminates the selective search for ROIs, instead it learns the ROIs by using separate network called Region Proposal Network (RPN).

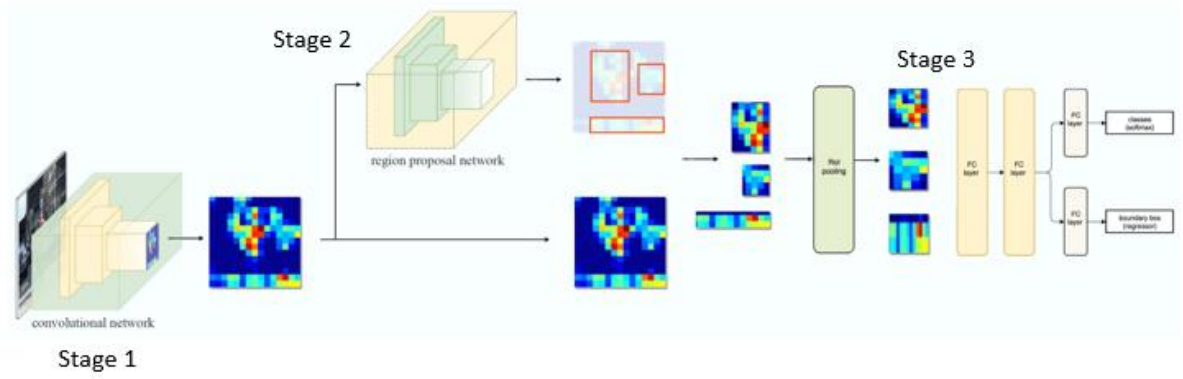


Figure 21 Illustration of Faster R-CNN architecture

To begin with, the input of the model is an image represented by H × W × D where H is the height, W is the width and D is the depths, usually 3 in RGB case. This image is passed through a series of pre-trained convolution layers (pretrained on ImageNet [37]), to extract visual features from the image (see stage 1 Figure 21**Error! Reference source not found.**). The next step is to find the region of interest for classification, in order to do that, they introduce the concept of Anchors. Anchors are fixed bounding boxes that are placed throughout the image with different sizes and ratios that are going to be used for reference when first predicting object locations. For each location in the image we have 9 anchors (see Figure 22). The RPN takes those anchors and outputs group of good proposal, it does that by returning two different outputs for each anchor (see stage 2 in Figure 21**Error! Reference source not found.**). The first one is a probability score which represents the probability of an object is found in this position, and the second output is four values of bounding boxes. However, the RPN doesn't care about the class of the object in the anchor, it cares only about presence of an object or not. As a result, the output of RPN will be 2k scores plus 4k bounding boxes.

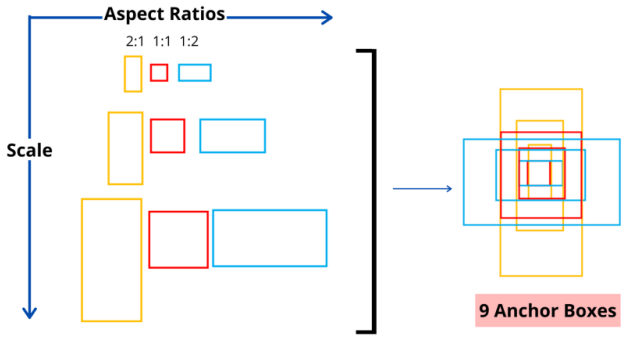


Figure 22 Anchors with different ratio and scales

After the RPN step, we have now a group of region proposal with no class assigned to them, and now the goal is to classify them. To solve this problem, they reused the output of feature extractor, and crop from it a fixed size feature map for each proposal using Region of internet pooling (RoI) that uses interpolation. After the RoI stage, the outputs are flattening and then passed to two different fully connected layers, one for classification and the other one for regression prediction, to predict the bounding boxes (see stage 3 in Figure 21).

In this model, we have four losses in total. These loses are obtained from RPN network and Detection network. The RPN doesn't care about the class of the object, it performs a binary classification to classify if the object inside the bounding boxes is background or an object for each anchor. We should mention that for the regression part of the problem where bounding boxes are predicted, we use the Smooth L1 loss function. On the other hand, the detection network uses normal classification loss, to classifies the class of the object plus the regression loss. Thus the loss for classification is normal linear plus softmax loss, which

is in other words multi class cross entropy loss, and for bounding boxes normal linear regress loss.

Table 2 shows a comparison between R-CNN, Fast RCNN and Faster R-CNN, we can clearly see how eliminating the selective search decreased the computation power required and the inferencing time. Moreover, the faster R-CNN is slightly better than Fast RCNN on both Pascal VOC test (2007 and 2012).

Table 2 Comparison between R-CNN, Fast RCNN and Faster RCNN

	R-CNN	**Fast R-CNN**	**Faster R-CNN**
Region Proposal Method	Selective Search	Selective Search	Region Proposal Network
Computation	High	High	Low
Inferencing time	40-50 secs	2-3 secs	0.2 secs
mAP on Pascal VOC 2007 test	58.5	66.9	69.9
mAP on Pascal VOC 2012 test	53.3	65.7	67.0

2.5.2 Single-stage Detectors

In this part of the report, I will talk about a group of object detection algorithms that use single network. In particular, I will focus on YOLO family.

YOLO

YOLO or You Only Look Once [38] is a single stage convolutional neural network that frames the object detection as regression problem. A single neural network is used to predict the bounding boxes and the class probabilities by looking at the image once. The authors of this paper took their inspiration from the human vision system, as the human can instantly know what object are in the image. This model can run in real-time at 45 frames per second (fps), moreover, a smaller version of the model can run at 155 fps speed.

First of all, YOLO divides the image into $S \times S$ grid (see Figure 23), and the cell that has the center of an object will be the cell responsible to detect that object. Each cell predicts B bounding boxes and a *confidence score*. The *confidence score* represents how confident the model is about having an object in that cell, and it can be calculated using the following equation:

$$\Pr(Object) * IOU_{pred}^{trurh}$$

Equation 11 Confidence score for cell grid in YOLO

Each bounding box has five predictions: *x*, *y*, *w*, *h* and a *confidence score*. The *confidence score* here is difference than the confidence score mentioned above, this one represent the IOU between prediction box and ground truth box. Moreover, the *(x, y)* represents the center of the box relative to the grid, while *(w, h)* represent the width and height respectively relative to the whole image.

Each grid cell predicts *C* conditional probabilities defined as $\Pr(Class_i/Object)$. At evaluation time, the conditional probabilities are multiplied by the box confidence predictions, as a result:

$$\Pr(Class_i/Object) * \Pr(Object) * IOU_{pred}^{trurh} = \Pr(Class_i * IOU_{pred}^{trurh})$$

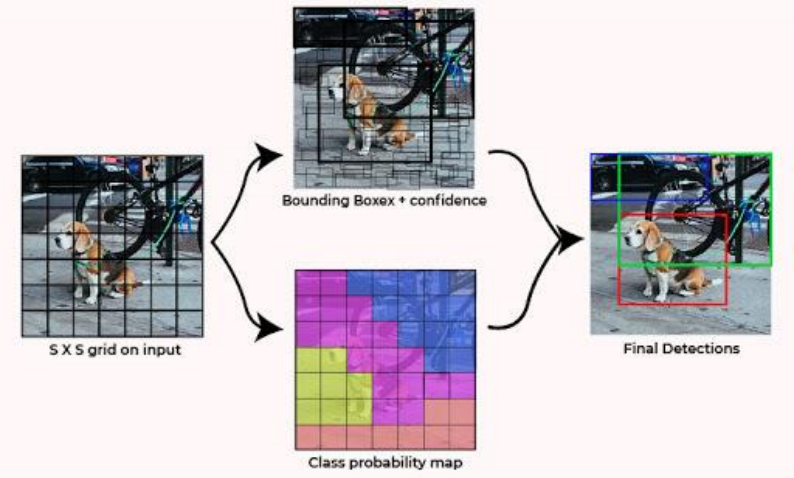


Figure 23 YOLO model deals with detections as regression problem

With $S \times S$ grid, and each cell predicts *B* bounding boxes and *C* class probabilities, those facts lead to $S \times S \times (B * 5 + C)$ predictions per image. While evaluation on Pascal VOC dataset, the authors of the paper chose $S = 7$ and $B = 2$.

Moving to talk about the actual architecture of the model, YOLO uses a convnet consists from 24 convolutional layers followed by two fully connected layers as shown in Figure 24.

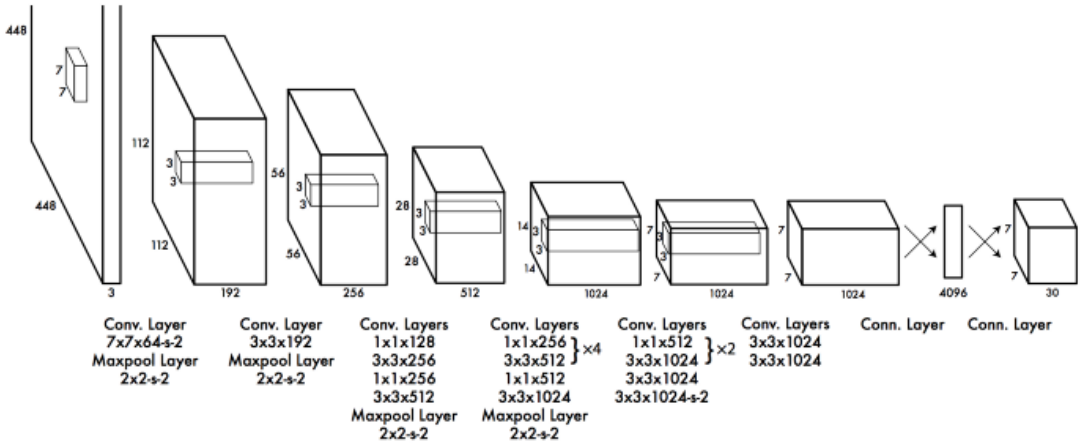


Figure 24 YOLO architecture consists of 24 convolutional layers and two fully connected layers, taken from [38]

The convolutional layer is pretrained on ImageNet with 224×224 resolution while in detection 448 × 448 resolution is used. Some layers use 1×1 convolution to reduce the depth of the feature map.

The bounding box predictor that has the highest IOU with the ground truth is considering the bounding box predictor responsible for predicting that object

Regarding the loss function, YOLO uses sum-squared error between the prediction and the ground truth to calculate the loss. On top of that, YOLO optimizes multi-part loss function that has classification loss, localization loss and confidence loss as shown in Equation 12.

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
$$

Equation 12 YOLO Multi-part loss, taken from [38]

We should note that the loss is penalizing the classification error for an object if that object is present in that grid cell. Also, that loss penalized the bounding box coordinate if that predictor is responsible to predict that object (have the highest IOU with the ground truth).

Additionally, YOLO can detect the same object more than once, to fix that, YOLO applies non-maximum suppression to eliminates duplicates with lower confidence score.

Finally, we can say that YOLO can run in real-time, and can train end-end as we are using single network. However, YOLO struggles to detect small and close objects in images

YOLOv2

Two of the researchers who worked on the original YOLO algorithm (Joseph Redmon and Ali Farhadi) created new version of it called **YOLOv2** [39] by adding some new features to improve the performance.

First of all, the authors of the paper added a batchnormaliaztion [40] in the convolutional layers, which led to remove the dropouts as no need for it anymore, and improve the performance by 2%. YOLO train the convnet on ImageNet dataset using 224×224 resolution, on the other hand, in YOLOv2 they trained the images with the same resolution but returned the resolution to 448 × 448 and continue the training for fewer epochs. This helped with increasing the performance by 4%.

The third improvement was removing the last two fully connected layers in YOLO and use anchors boxes to predict the bounding boxes as in Faster RCNN (they call the anchors as priors in the paper).

Fourth improvement is using the K-means clustering algorithm to find the top K-bounding boxes that have the best convergence with the training data.

Finally, they replaced the convnet feature extractor with custom feature extractor called Darknet-19. The reason why to replaced it is because the old one requires around 30 billion floating point operation for a single pass over an image, while Darknet-19 requires around 5.5 billion floating point operation. As a result, we have got an improvement in term of speed. Darknet mostly uses 3×3 filters to extract features and 1×1 filter to reduce channels (see Table 3).

Table 3 Darknet-19 Architecture

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 2/2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		2 × 2/2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1000
Softmax			

There are few more changes, however, I've mentioned some of them. In the below table, we can see the path from YOLO to YOLOv2, and how the performance on VOC2007 dataset rised from **63,4 mAP** to **78.6 mAP**.

Table 4 Path from YOLO to YOLOv2, taken from []

	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

YOLOv3

Few years later, Joseph Redmon and Ali Farhadi released a new version of YOLO under the name of **YOLOv3** [41]**.** The major change in YOLOv3 is replacing the Darknet-19 feature extractor used in YOLOv2 by another one larger called Darknet-53. The new feature extractor consists mainly from 3×3 and 1×1 convolutional layers like in Darknet-19, however, it has some skip connections like in ResNet (see Table 5). Thus we have a significantly large network. Besides, Darknet-53 achieved more classification accuracy than Resnet-152 with 2× faster.

Moreover, YOLOv3 uses independent logistic regression instead of softmax to perform multi-label classification (for instance "animal" and "cat") and for complexity reasons.

Table 5 Darknet-53 Feature Extractor

	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			

Additionally, YOLOv3 makes prediction at three different scales to help the model to detect objects at different scales (see in Figure 25). Also, YOLOv3 applies K-means clustering to select anchor box from predefined 9 clusters.

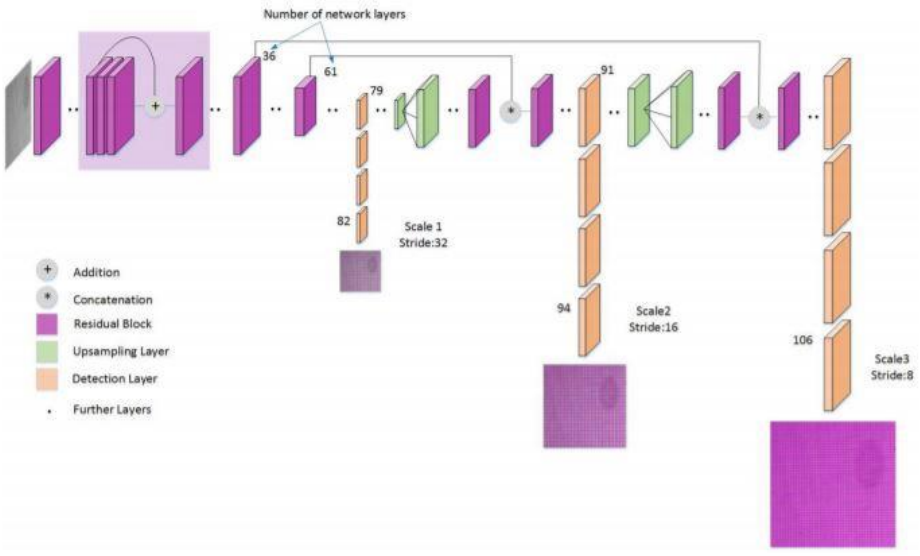


Figure 25 YOLOv3 architecture with three different scales

Finally, YOLOv3 is better than **SSD** [42] algorithm (see Table 6) and close to the State-of-the-art model but with faster inference time.

Table 6 Result of YOLOv3 vs other Detectors, taken from []

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [3]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [6]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [4]	Inception-ResNet-v2 [19]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [18]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [13]	DarkNet-19 [13]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [9, 2]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [2]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [7]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [7]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

YOLOv4

In 2020, a new version of YOLO framework has been released under the name of **YOLOv4** [43]**.** The main objective of this project was to build a powerful object detection model than can be accessible to anyone. The model achieves a 43.5% AP (65.7% AP_{50}) on MSCOCO [44] as seen in Figure 25, while still maintaining a real-time speed at 65 FPS on Tesla V100. This model can run on a single GPU.

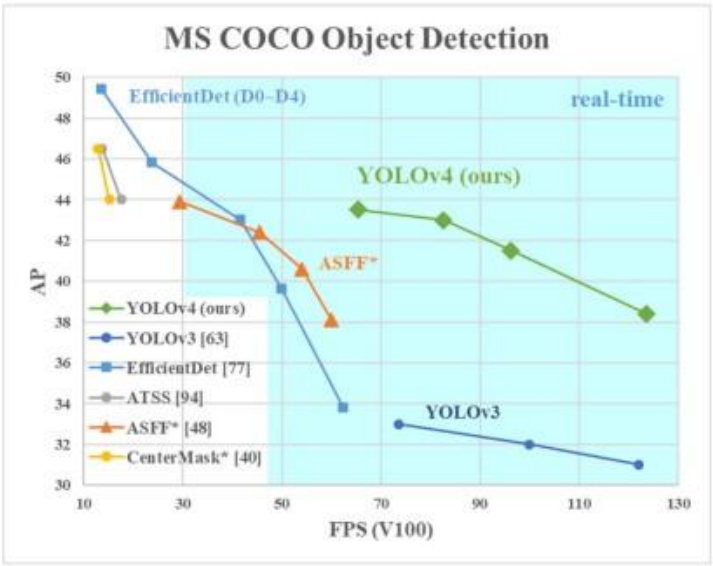


Figure 26 Comparison between YOLOv4 and other Object Detection algorithms on MSCOCO dataset

To improve the performance, authors of the paper divided the improvements into two categories. First of all, the training process is an offline operation, which means they can improve the training process without affecting the inference cost. This process called **"Bag of Freebies"**. One of these improvements was to use different data augmentation techniques to help the model to generalize on new images. Some of the techniques used in data augmentation are CutMix [45], and they introduce new one called Mosaic. Moreover, label smoothing proposed in [46] used in this model to make it more robust.

On the other hand, another type of improvement used called **"Bag of specials".** This kind of improvement increases the inference cost but significantly improve the accuracy. One of these improvement is to use Mish activation function [47] in the backbone to handle the vanishing gradient problem. Spatial Pyramid Pooling (SPP) [48] also used in the neck to eliminate the fixed size input limitation.

There are more improvements added to the Bag of freebies and back of specials (see Table 7), however I will not list them all, more details can be found in the paper [43].

In general, the Yolov4 architecture consists of:

- Backbone: CSPDarknet53 [49]
- Neck: SPP [48], PAN [50]
- Head: YOLOv3 head [41]

Table 7 Bag of freebies and Bag of specials used in the backbone and detector head in YOLOv4

	Bag of freebies	Bag of specials
Backbone	• CutMix augmentation [45] • Mosaid augmentation • DropBlock regularization • Class label smoothing	• Mish activation • Cross-stage partial connections (CSP) • Multi-input weighted residual connections (MiWRC)
detector	• CIoU-loss • DropBlock regularization • Mosaid augmentation • CmBN • Self-Adversarial Training • Eliminate grid sensitivity • Cosine annealing scheduler	• Mish activation • SPP-block • SAM-block • PAN • DIoU-NMS

YOLOv5

Less than 50 days after the release of the YOLOv4, Gleen Jocher who is the creator of mosaic augmentation mentioned in YOLOv4, has released a new version of YOLO called YOLOv5 without any official paper.

First of all, the new version is implemented in PyTorch, while YOLOv4 was built on Darknet framework (written in C and CUDA). The advantage here is now the new version is production ready and the deployment is easier. Moreover, YOLOv5 is small, as the weights size of YOLOv5 equals to 27 megabytes comparing to 244 megabytes comparing to YOLOv4 weight.

Additionally, YOLOv5 has more than one version. The size of the network and the number of parameters various between those versions.

Figure 27 shows a comparison between four version of the YOLOv5 model with EfficientDet model [51].

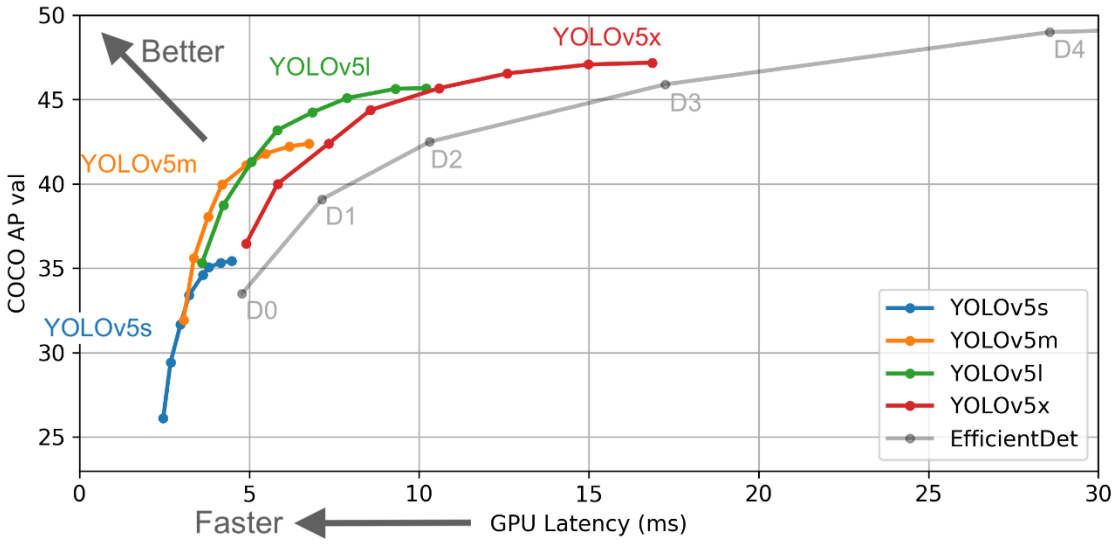


Figure 27 Result of YOLOv5 on COCO dataset, taken from https://github.com/ultralytics/yolov5

YOLOv7

Recently, in July 2022, the author of YOLOv4 has released a new version called **YOLOv7** [52]. The new model achieved 56%.8 AP and still maintaining real-time speed. They have different model architecture with different number of parameters. Like any object detection algorithm YOLOv7 has three main components which are backbone, neck and head. The backbone is pretrained on COCO dataset not on Imagenet.

There are two types of improvements in this model, improvement at architecture level and bag of freebies (like in YOLOv4).

Extended Efficient Layer Aggregation Network (E-ELAN) is used in the backbone as computational block. According to the paper, the proposed E-ELAN uses expand, shuffle,

merge cardinality to achieve the ability to continuously enhance the learning ability of the network without destroying the original gradient path. However, the official paper of ELAN block is not published yet.

On the other hand, bag of freebies is used to increase the performance without affecting the inferencing cost. Planned re-parametrized convolution is used to improve the performance. Model level re-parametrized also used, basically, this is done by using different training data with the same settings, train multiple models and then averaging their weights.

YOLOv7 models surpass the previous object detection algorithms in speed and accuracy on MSCOCO dataset. The below figure gives an idea about the AP and the speed of this model.

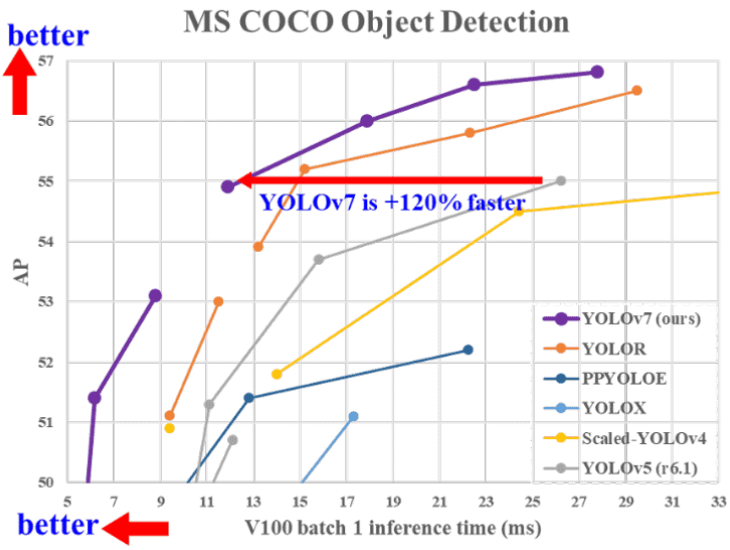


Figure 28 Comparison between YOLOv7 and other Object detections on MSCOCO dataset

YOLOv6

In 7th of Sep 2022, a team at Meituan which is a Chinese e-commerce platform created a new version of YOLO called **YOLOv6** [53]**.** The new model is written in PyTorch.

YOLOv6 comes with three main improvements. First of all, they claim that the new model is hardware friendly. The backbone and neck of YOLOv6 take the computing power, memory bandwidth into consideration. They used Rep-Pan and EfficientRep structures in the neck and backbone respectively. Moreover, YOLOv6 uses the decoupled head design introduced in **YOLOX** [54] which helped the new model to increase the speed and accuracy. Finally, YOLOv6 makes use of the anchor-free paradigm.

Table 8 Comparison between YOLOv6 models and other detection modelsTable 8 shows a comparison between different version of YOLO with the new release YOLOv6.

Table 8 Comparison between YOLOv6 models and other detection models

Method	Input Size	AP^{val}	AP^{val}_{50}	FPS (bs1)	FPS (bs32)	Latency (bs1)	Params	FLOPs
YOLOv5-N [10]	640	28.0%	45.7%	602	735	1.7 ms	1.9 M	4.5 G
YOLOv5-S [10]	640	37.4%	56.8%	376	444	2.7 ms	7.2 M	16.5 G
YOLOv5-M [10]	640	45.4%	64.1%	182	209	5.5 ms	21.2 M	49.0 G
YOLOv5-L [10]	640	49.0%	67.3%	113	126	8.8 ms	46.5 M	109.1 G
YOLOX-Tiny [7]	416	32.8%	50.3%*	717	1143	1.4 ms	5.1 M	6.5 G
YOLOX-S [7]	640	40.5%	59.3%*	333	396	3.0 ms	9.0 M	26.8 G
YOLOX-M [7]	640	46.9%	65.6%*	155	179	6.4 ms	25.3 M	73.8 G
YOLOX-L [7]	640	49.7%	68.0%*	94	103	10.6 ms	54.2 M	155.6 G
PPYOLOE-S [45]	640	43.1%	59.6%	327	419	3.1 ms	7.9 M	17.4 G
PPYOLOE-M [45]	640	49.0%	65.9%	152	189	6.6 ms	23.4 M	49.9 G
PPYOLOE-L [45]	640	51.4%	68.6%	101	127	10.1 ms	52.2 M	110.1 G
YOLOv7-Tiny [42]	416	33.3%*	49.9%*	787	1196	1.3 ms	6.2 M	5.8 G
YOLOv7-Tiny [42]	640	37.4%*	55.2%*	424	519	2.4 ms	6.2 M	13.7 G*
YOLOv7 [42]	640	51.2%	69.7%	110	122	9.0 ms	36.9 M	104.7 G
YOLOv6-N	640	35.9%	51.2%	802	1234	1.2 ms	4.3 M	11.1 G
YOLOv6-T	640	40.3%	56.6%	449	659	2.2 ms	15.0 M	36.7 G
YOLOv6-S	640	43.5%	60.4%	358	495	2.8 ms	17.2 M	44.2 G
YOLOv6-M‡	640	49.5%	66.8%	179	233	5.6 ms	34.3 M	82.2 G
YOLOv6-L-ReLU‡	640	51.7%	69.2%	113	149	8.8 ms	58.5 M	144.0 G
YOLOv6-L‡	640	52.5%	70.0%	98	121	10.2 ms	58.5 M	144.0 G

2.6 Segmentation

Moving to talk about segmentation, segmentation it the next level of object detection. In object detection, the goal is to draw a bounding box plus a label for each object in the image, however, segmentation is the task of classifying at pixel level. In other words, the goal is to label each pixel in the image.

Mainly, there are two types of image segmentation: semantic segmentation and instance segmentation. The different between them is in semantic the goal is to associates every pixel of an image with a class label such as person, car... While instance treats objects with the same class as multiple instance, in other words, instance segmentation is a combination between object detection and semantic segmentation. Figure 29 illustrates the difference between object detection, semantic segmentation and instance segmentation.

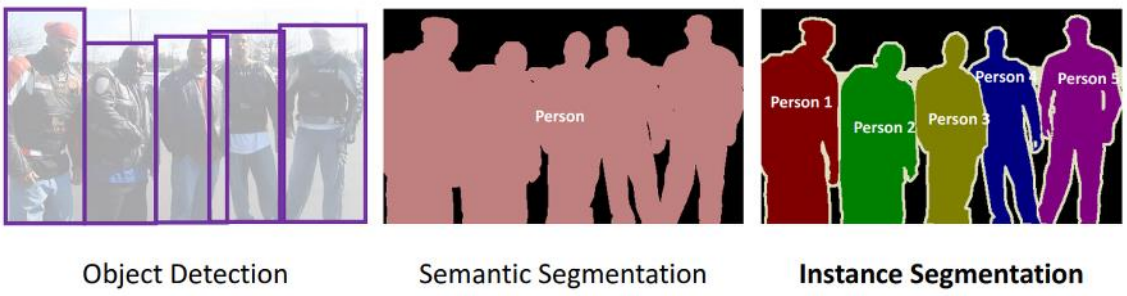


Figure 29 Difference between object detection, semantic segmentation and instance segmentation

There are some algorithms to handle the problem of semantic segmentation such as **FCN** [55], **Unet** [56], **DeepLab** [57], etc… However, I will be focusing on instance segmentation as I will use it in this project.

In 2018, a group of researcher from Meta (Facebook formally) published a paper under the name of **Mask-RCNN** [58]**.** They introduced a new algorithm to solve the problem of instance segmentation. Mask-RCNN is an extension of Faster RCNN algorithm.

Mask RCNN classify each pixel in every region of interest generated by the RPN network. It does that by adding convolutional layers on top of the outputs of RPN network to classify at pixel level, as we can see the figure below.

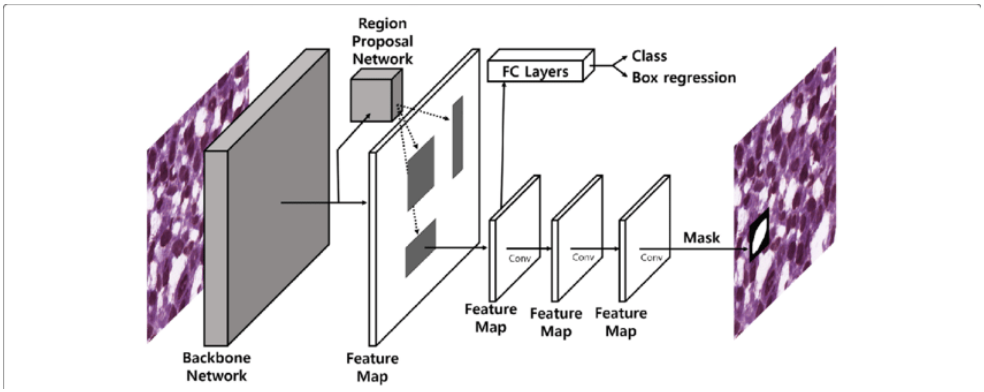


Figure 30 Mask RCNN Architecture

The authors noticed that there is misalignment between the region of the feature map and the original image, this is caused by the ROI pooling layer. Thus they make use of RoIAlign layer instead of RoI pooling used in Faster RCNN to fix this problem. It does that by using bilinear interpolation to get the exact value of the feature in each region of interest.

Mask RCNN uses the same loss as in Faster RCNN. However, they added a mask lose as well as shown in Equation 13.

Equation 13 Mask RCNN Loss

$$L = L_{cls} + L_{box} + L_{mask}$$

For each region of interest there are *K* binary mask, each one with $m \times m$ resolution. The model is trying to learn a mask for each class, there is no competition among classes for generating masks. L_{mask} uses a cross entropy loss

2.7 Evaluation metrics

The evaluation metric used to measure how good or bad the object detection algorithm is called mean average precision or mAP for short. Mean average precision is the most popular metric used in most benchmark challenges such as Pascal VOC, ImaNet, COCO, etc…

In order to understand mAP and how it works, first let us explore the IoU or intersection over union terminology. As we discussed before in section 2.5, the output of any object detection will be a bounding boxes plus a class label. IoU is the metric used to measure the overlap between the ground truth box and the predicted box, as seen in Figure 31. Additionally, for segmentation, we predict a masks instead of bounding boxes, however we can still apply the same technic.

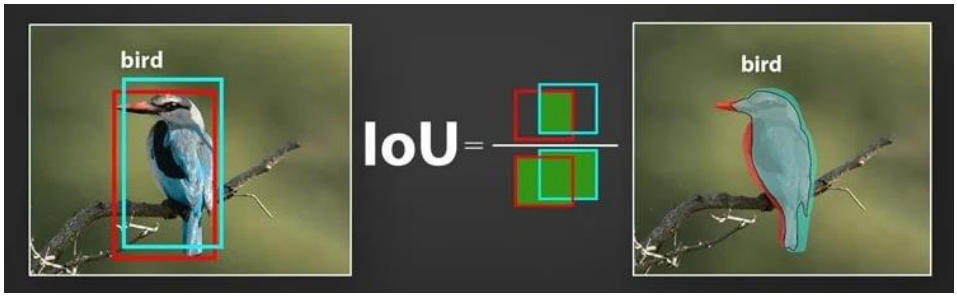


Figure 31 Illustration of the intersection over union IoU

We can calculate it by dividing the intersection by the union. The range of IoU is between 0 and 1. 0 when there is no intersection at all, 1 when the overlap is 100% perfect. Using this metric, we can identify whether the prediction is True Positive (TP), False Positive (FP), or False Negative (FN).

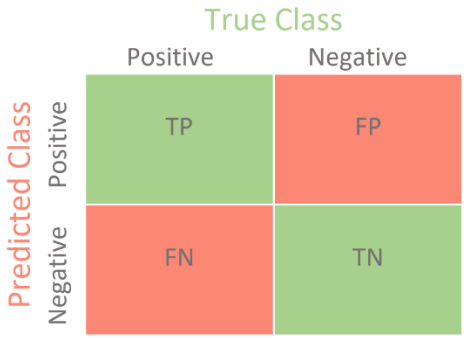


Figure 32 Example of Confusion Matrix

When the predicted value is True and the original value is True then it is TP, while if the original value is False then it is FP. When the predicted value is False and the original value is True then it is FN, while if the original value is False then it is TN. As seen in Figure 32.

If the value of IoU is greater or equal than a certain threshold, then the prediction is TP. If the value of IoU is less threshold, then the prediction is FP. If the predicted class is wrong, or there is no detection, in other words if the model predicts the wrong class or didn't predicted at all, then the prediction is FN.

Note that True Negative (TN) is not considered in object detection because it is always correctly detecting the background.

Another two terminologies used to calculate the mAP are the precision and the recall.

Precision is calculated by dividing the number of TP over the total predictions, as seen in Equation 14. In other words, the precision represents the portion of the predicted positive that are actually positive.

Equation 14 Precision Equation

$$Precision = \frac{TP}{TP+FP}$$

On the other hand, the Recall is defined as the proportion of the actual positive that were predicted correctly. It represents the sensitivity. It can be calculated using the below equation:

Equation 15 Recall Equation

$$Recall = \frac{TP}{TP+FN}$$

Both Precision and Recall are in 0 to 1 range.

Finally, now let us talk about average precision and mean average precision. Someone can say the average precision is the average of the precision values calculated using Equation 14, but it is something different. A formal definition of the average precision is the area under the precision-recall curve. The AP is calculated using the Equation 16 equation:

$$AP = \int_0^1 p(r)dr$$

Equation 16 Average Precision

After calculating the AP for each class, mean average precision is the average precision divided by the number of classes, as seen in Equation 17:

$$mAP = \frac{\sum_{i=1}^{N} AP(i)}{N}$$

Equation 17 Mean Average Precision

The official metric used in Pascal VOC challenge is the mAP with 0.5 as IoU threshold. On the other hand, COCO challenge uses another approach, where they calculate the mAP using 10 different thresholds which are defined as [0.5:0.05:0.95]. In other words, they calculate the mAP using 0.5 threshold, they increment it by 0.05 and calculate it, repeat the same process until 0.95 threshold. Finally, they sum them all and divide it by 10, as shown in Equation 18:

$$mAP = \frac{mAP_{0.5} + mAP_{0.55} + \cdots + mAP_{0.95}}{10}$$

Equation 18 mAP for COCO challenge

For a detailed example on how to calculate mean average precision see Appendix A.

3. Methodology

3.1 Dataset

First of all, choosing a good and appropriate dataset is one of the most important step in any machine learning and deep learning project. For this project I chose the **BDD100K** [59] dataset. The Berkeley Deep Drive Dataset (BDD100K) is relatively new dataset (2020), it is diverse and well annotated for several tasks such as object detection, semantic segmentation, instance segmentation, multi object tracking, etc…

This dataset consists of 100K video clips which is equal to over 1K hours of recorded driving videos in various conditions. The authors of the dataset provide images extracted from the videos as well, which of course reduce the preprocessing time to collect images from this dataset. All the images are in RGB format with 1280×720 resolution.

The dataset collected from USA in multiple locations and cities such as New York, Berkeley, San Francisco and Bay Area, as shown in Figure 33.

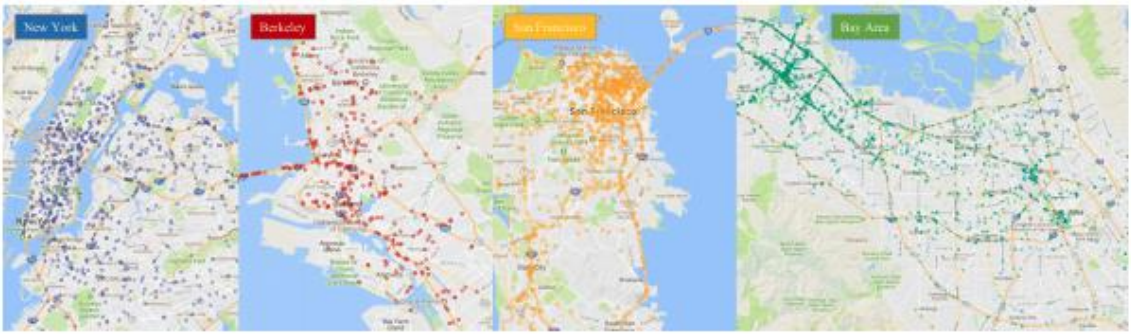


Figure 33 Geographical distribution of BDD100K data sources, taken from [59]

This dataset has train, test and validation sets. The ground truth for test set are not provided with the dataset. The training set has 70K images while the validation has 10K. For this project, for object detection part, I've randomly selected 40K for training and validation, 80 percent which is equals to 32K image for training, and 20% for validation which is equals to 8K. On the other hand, 5K images used for testing. For the segmentation part, we have only 10K images with annotations.

As mentioned above, this dataset collected in different weather condition, as seen in Figure 35. It has seven different weather conditions, although the clear condition has the highest percentages. Some of the images extracted from the videos classified as 'undefined' as the images extracted from the videos are not clear.

Moreover, BDD100K includes different scenes and places in cities, which gave this dataset an advantage over other dataset in the same domain. As illustrated in Figure 34, the images are from six different scenes including residential, city, highways, etc…

Finally, the videos were taken at different time of the day, from morning till night. Figure 36 shows the different time of the day included in this dataset: daytime, night, dawn.

All of these various gave this dataset big advantages over other dataset and helped to reduce the bias.

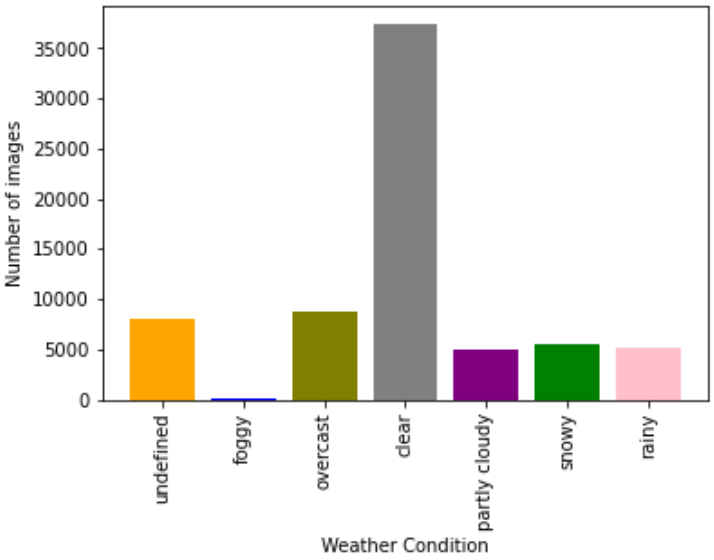


Figure 35 Weather condition in BDD100K dataset

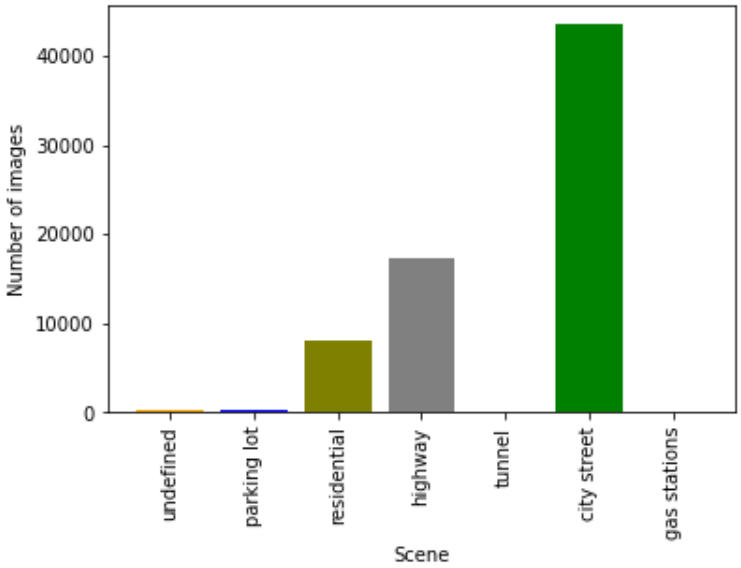


Figure 34 Different Scenes in BDD100K

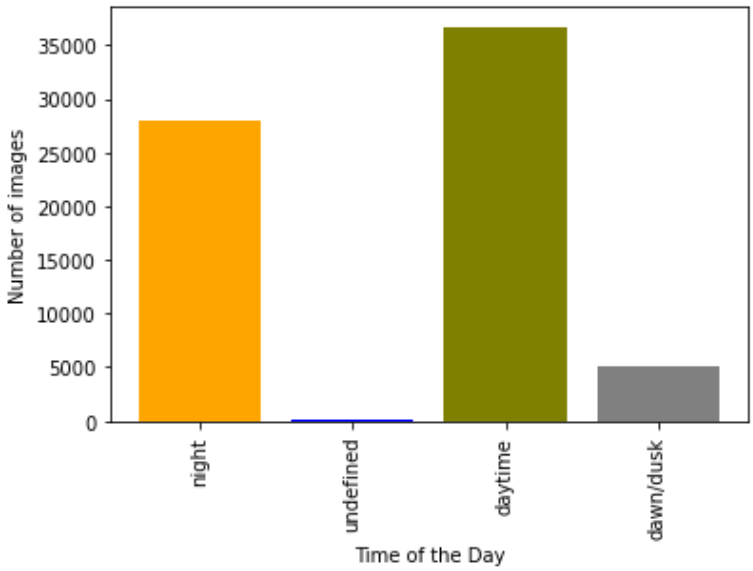


Figure 36 Time of the Day in BDD100K

For the object detection task, BDD100K has 10 different classes as seen in Figure 37. This dataset is heavily imbalanced dataset, as in most self-driving dataset, because most of the images are taken from the streets, where we have many cars. For this project, I have picked

only 4 classes to perform object detection on them. These classes are: cars, pedestrian, traffic light and traffic sign. Those are the most representative classes.

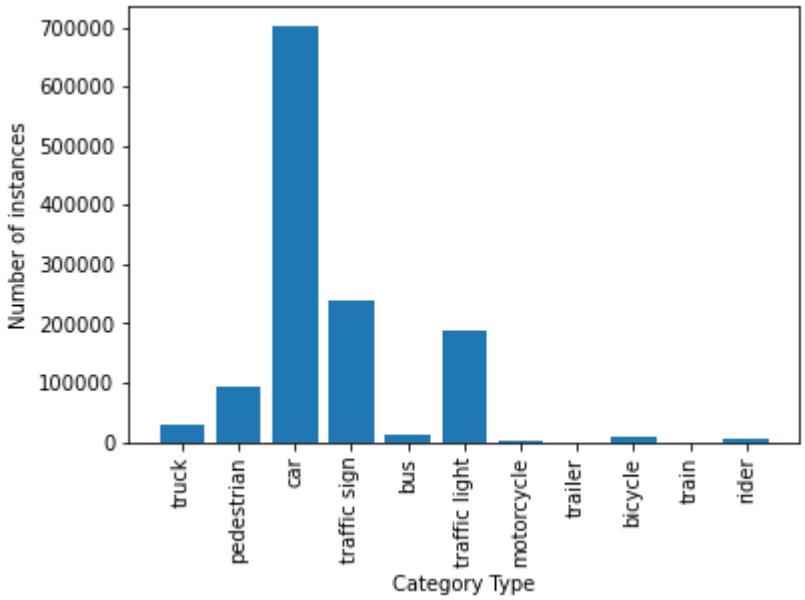


Figure 37 Classes distribution in BDD100K

Moving to talk about the bounding boxes. This dataset provides annotation for 2D object detection in JSON format. The bounding boxes are in *xyxy* format, which is the format used in Pascal VOC dataset. The distribution of number of bounding boxes per image is illustrated in Figure 38. We can see that most of the images has instances between 10 and 30 per image.

Figure 39 shows a scatter plot for width vs high for all bounding boxes in each category. We can see that most of the bounding boxes are small except in some particular cases like trains and some cars.

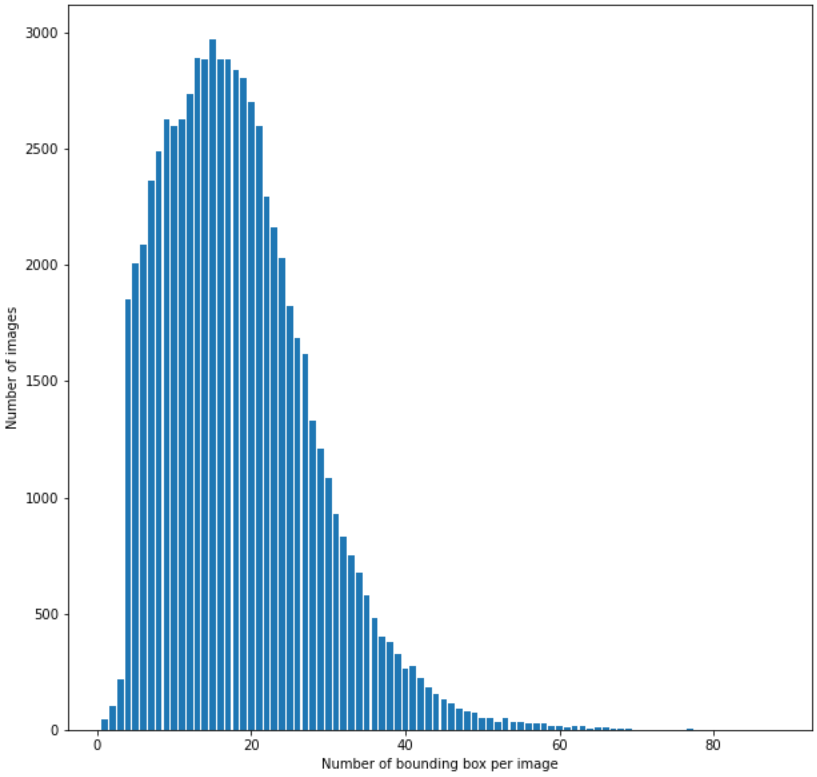


Figure 38 Distribution of bounding boxes per image

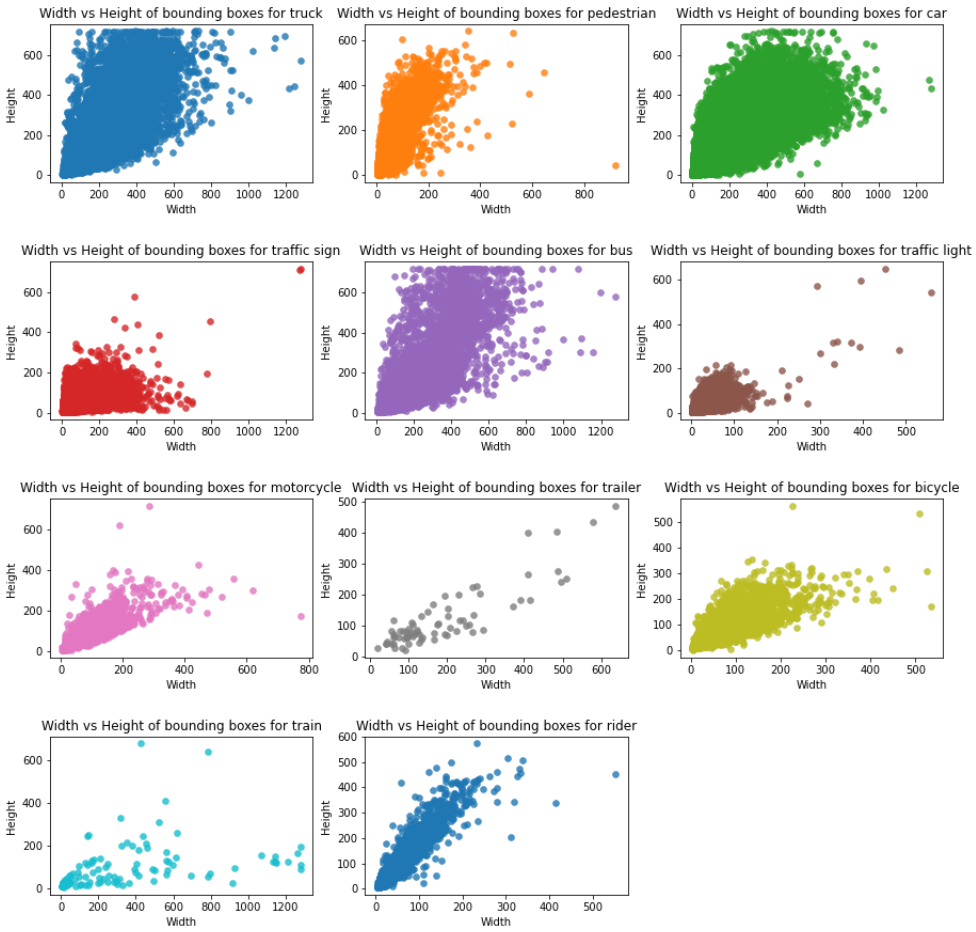


Figure 39 Width and High for bounding boxes for all classes in BDD100K

Final point to discuss in this dataset is the mean and standard deviation. It is recommended in computer vision to normalize picture pixel values in relation to the mean and standard deviation of the dataset. This can be helpful for transfer learning and for obtaining consistent results when using a model on other pictures. Since we frequently can't load the entire dataset in memory and must cycle over it in chunks, computing these statistics can occasionally be a little challenging in reality.

With that being said, I've calculated the mean and std using the following equations:

Equation 19 Mean Equation

$$\mu = \frac{1}{n}\sum_{1}^{n} x_i$$

Equation 20 Standard Deviation Equation

$$\sigma^2 = \frac{1}{n}\sum_{1}^{n} {x_i}^2 - \mu^2$$

As a result, the mean of this dataset is **[0.2787, 0.2926, 0.2899]** with [**0.2474, 0.2653, 0.2760]** Standard Deviation.

The code used to generate those chart and data can be found in mean and std and EDA.

3.2 Data Preparation

After performing some analysis for the dataset, we must do some data preprocessing to prepare the data for the training phase.

First of all, I have selected only four categories to perform object detection on, so I checked if each image has at least one instance from those categories.

Second step was resizing. Each image is resized to a smaller size, to 640×640 pixels to be specific. To original size of the images is 1280×720 pixels, which is too big. The purpose of this step is to reduce the training and inference time.

After that, the normalization step is very important as I mentioned in above section. Thus, each image is normalized using the mean and stander deviation over all three channels (RGB).

Next, I performed some data augmentation techniques to make the models generalize on new images and new scenarios. Some of the techniques I have used are: horizontal flip, crop and pad, pixel dropout, etc...

In Table 9 we can see the data augmentation techniques used in this project for each model.

Table 9 Data Augmentation techniques used

Model	Augmentation techniques
Faster RCNN & Mask RCNN	• Horizontal Flip • Crop and Pad • Pixel Dropout
YOLO family	• HSV augmentation • Rotation • Translation • Scale • Shear • Image Perspective • Flip up-down • Flip left-right

	• Mosaic • Mixup • Segment copy-paste

While augmenting the data, we must change the annotation as well. For instance, when we resize or add padding for an image, the coordinates of the bounding boxes must be changed, otherwise we will give the model wrong annotations.

Figure 40 illustrates the pipeline used in the preprocessing phase.

Figure 40 Data preprocessing pipeline

Final note in this section is about the bounding boxes format. In the literature we have three different format used in object detection domain which are Pascal VOC, COCO and YOLO. We can convert any of them to another format using specific functions.

The bounding box annotations provided with the dataset is in Pascal VOC format. Basically the Pascal voc is in *xyxy* format, first *xy* presents the $x_{min}$and y_{min}, and the last xy presents x_{max} and y_{max}. Both Faster RCNN and Mask RCNN accept this format, so no need to do anything.

The second format is COCO format, which is in (*x*, *y*, *w*, *h*) format, where x and y represent the center of the bounding box and w and h represent the width and height.

Finally, the YOLO format is the same as COCO format, but the coordinates must be normalized with the width and high of the image. All models in YOLO family requires the ground truth to be in YOLO format.

Figure 41 shows the coordinates of bounding boxes for three objects, and we can see how the coordinates of the boxes are normalized in respect to the width and height of the image.

As that being said, to perform the object detection with YOLO family we must convert all the annotation to YOLO format.

Check Appendix B to see the code of converting the annotations. Moreover, In Readme file in project's GitHub page, there are instruction how to convert the annotation to YOLO through terminal.

Figure 41 YOLO Format

On the other hand, for the segmentation task, the goal is to augment the instance segmentation data with drivable area segmentation. As mention above, we have only 10K images with instance level annotation. In addition, we have almost 70K images with drivable area annotation. The intersection images between these two list of images has 2976 images, which I will use in the segmentation task. The goal in this part is to combine the drivable area masks alongside with the instance segmentation level masks and run Mask RCNN to see how the result will be. For this task, the classes I will try to segment are: cars, persons and road.

3.3 Model Selection

For the object detection task, I have selected seven different model architecture, some of them from the same model but with different sizes. For instance segmentation task, I used Mask RCNN. I have started training all my models from pre-trained weights as starting point.

Table 10 Model Architecture

Model	Backbone	Number of parameters (M)
Faster RCNN	Resnet50	43
Mask RCNN	Resnet50	45.9
YOLOv5s	CSPDarknet53	7.2
YOLOv5l	CSPDarknet53	46.5

YOLOv5x	CSPDarknet53	86.7
YOLOv7	E-ELAN	37
YOLOv7-X	E-ELAN	71
YOLOv6s	CSPStackRep Block	17.2

In Table 10 we can see the backbone used in each model and the number of parameter for each model. There are various models sized, some of them are really big, for example, YOLOv5x and YOLOv7x, some of them are in the middle like Faster RCNN, YOLOv7, and there is one small model with 7.2 M parameter, which is YOLOv5s.

Table 11 illustrates the training parameters. Adam optimizer is the best optimizer, so I used it in all models. Also the learning rate is very small 0.00001, the learning rate is very important hyperparameter, and we should choose it carefully. I chose to go with very small learning rate as we have a lot of images and a lot if instances per image. Faster RCNN and Mask RCNN start to overfit after 10 epochs. The batch size is not the same for all models, that's because some models are bigger than others, thus bigger batch won't fit into the memory.

Table 11 Training parameters

Model	Optimizer	Learning Rate	Train/val/test	Number of Epochs	Batch size
Faster RCNN	Adam	1e-5	32K/8K/5K	10	1
Mask RCNN	Adam	1e-5	2380/297/297	10	1
YOLOv5s	Adam	1e-5	32K/8K/5K	50	32
YOLOv5l	Adam	1e-5	32K/8K/5K	50	32
YOLOv5x	Adam	1e-5	32K/8K/5K	50	16
YOLOv7	Adam	1e-5	32K/8K/5K	50	32
YOLOv7x	Adam	1e-5	32K/8K/5K	50	16
YOLOv6s	Adam	1e-5	32K/8K/5K	50	32

In the segmentation task, I trained the model on 50K images to detect and segment the drivable area for 5 epochs. Each epoch took about 4 hours of training. After that, I have used the same weights to retrain the model on the 2976 images, which presents the intersection between the instance segmentation images and the drivable segmentation images, in order to detect and segment the drivable area plus the instances. Figure 42 illustrates the methodology I followed in order to train the model to segment the root and the instances at the same time using MaskRCNN.

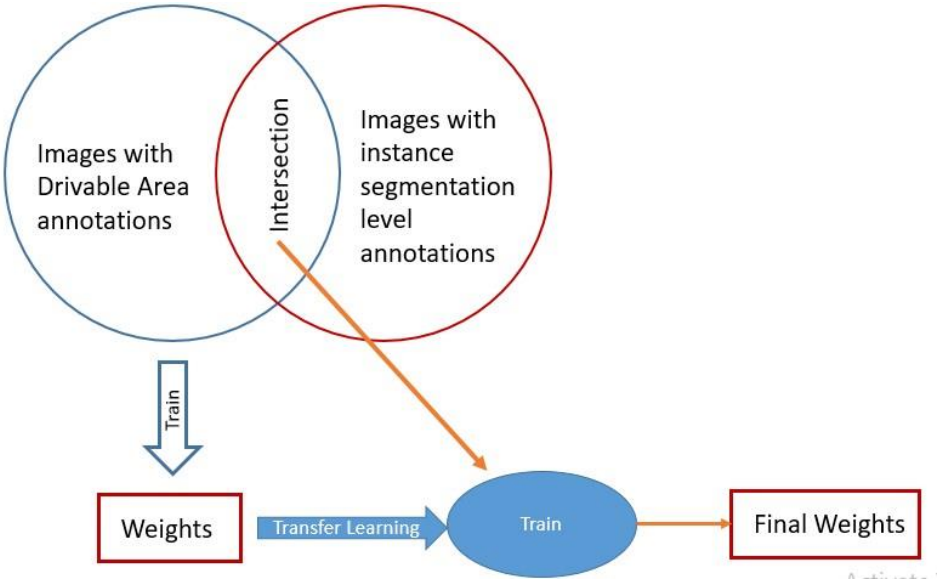


Figure 42 diagram illustrates the methodology for segmentation task

3.4 Tools

I have used several tools and libraries in order to complete this project. First of all, Python is the only programming language used in this project as it has a lot of libraries used in machine learning and deep learning. Pytorch [60] is the framework used in this project. I chose Pytorch over tensorflow because it is more flexible, also because PyTorch comes with a lot of implement models and pre trained weights. On top of pytorch, I have used Pytorch Lightening [61] which is a framework built on top of pytorch to facilitate and accelerate the research. PyCharm and jupyter notebooks alongside with anaconda made my life easier. Flask framework were used to build APIs. In addition, I have employed the weight and bias MLops platform to monitor my models while training.

All the packages and libraries I have used are available in the requirement.txt file.

I trained all my models on the cloud using a platform called JarvisLab.

3.5 Experiment Implementation

As said in 3.4, the whole framework is implement using Python programming language. The BDD class is inherited from the Dataset torch class. The **BDD** is just an interface for the **BDDDetection** and **BDDisntanceSegmentationDrivable** classes, so both of them are inherited from the **BDD** (see Figure 43). In both classes, there are some method to be implement, those methods are specific for each class, however they share some methods and arguments.

BDDDetection is the class responsible for returning the images and bounding boxes as tensors to the detection model. **BDDisntanceSegmentationDrivable** is the class responsible for the instance segmentation task.

On the other hand, **FasterRCNN** and **MaskRCNN** classes are inherited from the **LighteningModule** provided by PyTorch Lightening. Each class has some specific functions to load the model, define the optimizer, training and validation steps, etc…

The YOLO models are independent projects; I've cloned them from GitHub. There are instructions on the Readme file in the project's GitHub page how to clone them and run training and inferencing using these models.

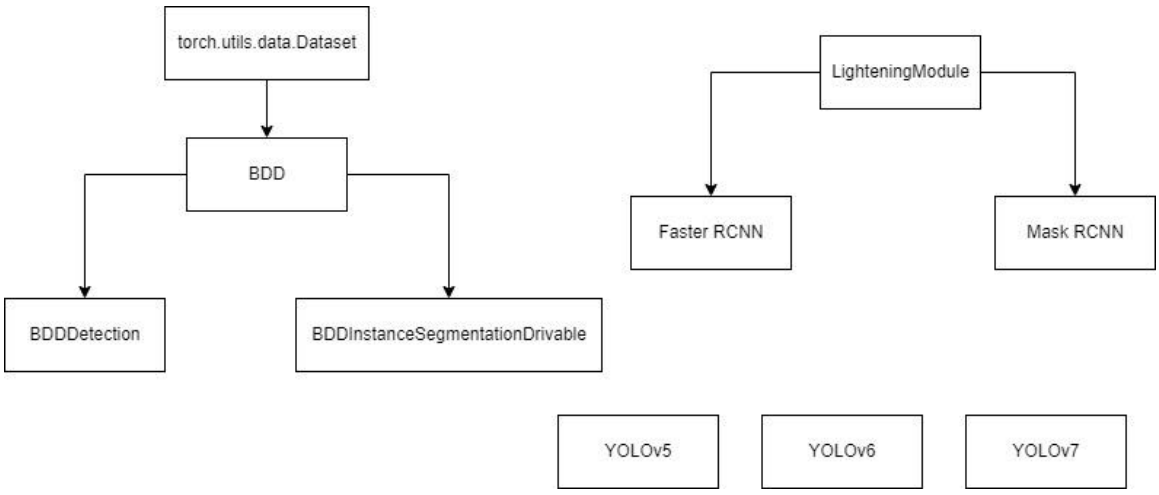


Figure 43 Framework Implementation

In Figure 44 we can see the project structure.

```
├─src
│  ├─config
│  │  ├─defaults.py # default configuration
│  ├─dataset
│  │  ├─bdd.py  # Superclass dataset, Parent class
│  │  ├─bdd_detetcion.py # Subclass for detection task
│  │  ├─bdd_drivable_segmentation.py # Subclass for drivabel area segmetation task
│  │  ├─bdd_instance_segmentation.py # Subclass for instance segmetation task
│  │  ├─bdd_instance_segmentation_drivable.py # Subclass for instance segmentation with drivable
area task
│  │  ├─bdd_utils.py  # file contains useful method
│  ├─dbs
│  │  ├─test_db.json # pre-created test db
│  │  ├─train_db.json # pre-created train db
│  │  ├─val_db.json # pre-created val db
│  ├─models
│  │  ├─Detection
│  │  │  ├─detection_models.py  # file contains the method to return the Faster RCNN model
│  │  │  ├─Faster_RCNN.py # Faster RCNN class
│  │  ├─Segmentation
│  │  │  ├─FCN.py # FCN class
│  │  │  ├─DeepLab.py # DeepLabv3+ class
│  │  │  ├─MaskRCNN.py # MaskRCNN class
│  │  │  ├─segmenation_models.py # file contains the method to return the models for
segmentatation
│  ├─utils
│  │  ├─DataLoaders.py # dataloader function
│  │  ├─utils.py # useful function
│  │  ├─Data Augmentation.ipynb # Notebook contains tutorial for data augmentation
│  ├─BDD Detection.ipynb # Notebook contains tutorial how to use detection class
│  ├─BDD Drivable Area.ipynb # Notebook contains tutorial how to use drivable class
│  ├─BDD Instance Segmentation.ipynb # Notebook contains tutorial how to use instance
segmentation class
│  ├─YOLO Format.ipynb # Notebook contains tutorial how convert xyxy format to yolo format
├─doc
│  ├─images # some images
├─notebooks
│  ├─Faster RCNN Notebook.ipynb # Faster RCNN notebook
│  ├─FCN Notebook.ipynb # FCN notebook
├─api
│  ├─app.py  # Flask App
├─yolov5  # yolov5 repository
├─yolov6  # yolov6 repository
├─yolov7  # yolov7 repository
├─dataset
│  ├─bdd100k
│  │  ├─images
│  │  │  ├─10K
│  │  │  ├─100K
│  │  ├─labels
│  │  │  ├─det_20
│  │  │  ├─drivable
│  │  │  ├─lane
│  │  │  ├─pan_seg
│  │  │  ├─ins_seg
│  │  │  ├─pan_seg
├─train.py  # file contains the functions to train Faster RCNN and MASk RCNN
├─test.py  # file contains the functions to evaluate the models
├─detect.py  # file contains the functions to run inference
├─prepare.py  # file used to prepare the data to YOLO algorithms
├─util.py  # contains useful functions to train, test and detect
```

Figure 44 Project Structure

4. Results

I have compared the models in terms of accuracy and speed. Additionally, I have tested the models on different hardware to test the speed. In the following subsection I will display some charts and figures to compare the results. I have resized all the input images to 640×640 pixels in all my experiments, unless I said otherwise.

4.1 Accuracy Results

Let's compare the accuracy results for all models. When I say accuracy I mean the mean average precision (mAP) I have talked about in section 2.7.

To begin with, I will compare the mean average precision used in MSCOCO competition (mAP [0.5:0.05:0.95]) for all models, also the mean average precision used in Pascal competition (mAP [0.5]) accompanied with mean average precision with 0.75 IoU threshold.

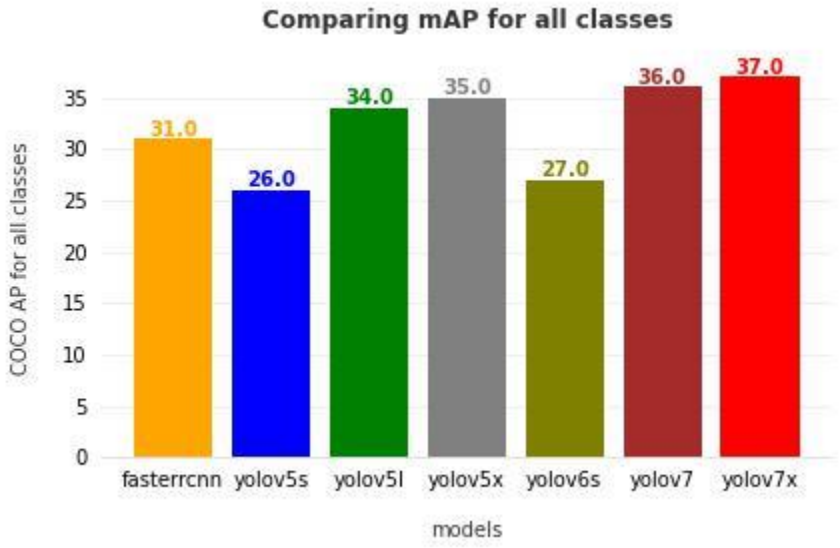


Figure 45 COCO AP for all classes

Figure 45 illustrates a comparison between all models using the COCO AP for all classes. On the other hand, Figure 46 and Figure 47 use 0.5 and 0.75 IoU threshold respectively to compare the models.

As we can see from the plots, yolov7 models (yolov7 and yolov7x) overcome all the other versions of YOLO family, as well as Faster RCNN model in all three metrics (COCO AP, PASCAL AP and mAP75).

On the other hand, yolov5s and yolov6s were the worst models among all, with 26 and 27 COCO AP, 55 PASCAL AP and 21 and 23 for mAP75 respectively.

Faster RCNN model gave a good result and it is comparable with other models like yolov5l and yolov5x, although yolov7 models still better than Faster RCNN.

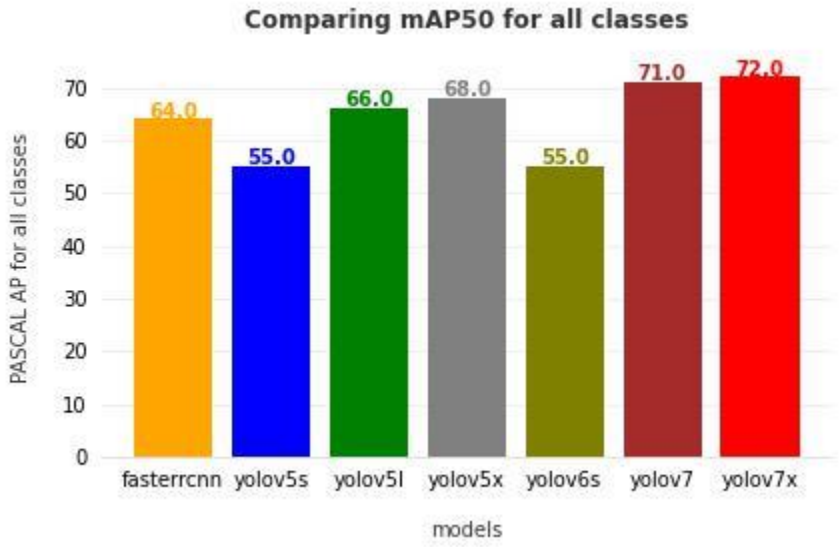


Figure 46 PASCAL AP for all classes

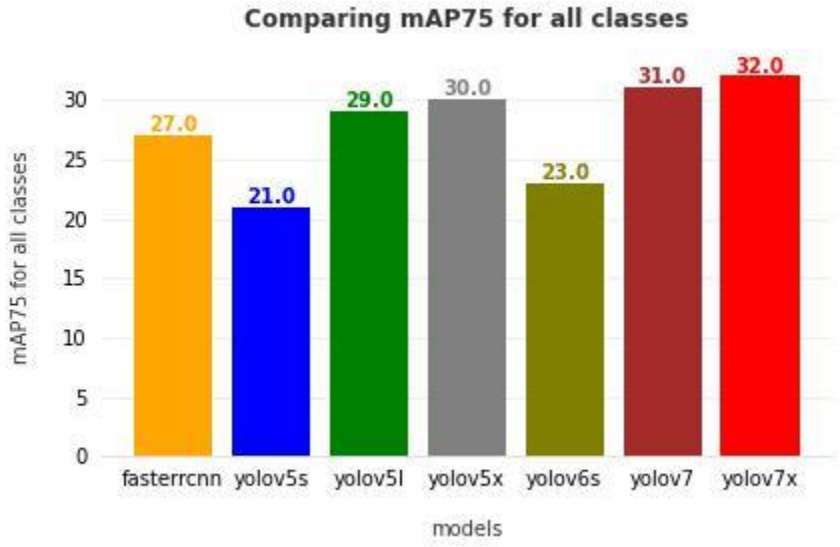


Figure 47 mAP with 0.75 IoU threshold for all classes

To understand the models better, I have decided to compare them in more details. Figure 48, Figure 49 and Figure 50 show the result of comparing the seven models using the COCO AP for different objects sizes, in particular small, medium and large objects.

For small objects (Figure 48Figure 48), as in previous comparison, yolov7 models are the best, yolov7x is the best one to be specific. Although, models like yolov5l, yolov5x and Faster RCNN gave a decent result as well in detecting small objects. On the other hand, yolov6s and yolov5s struggled in detecting small objects.

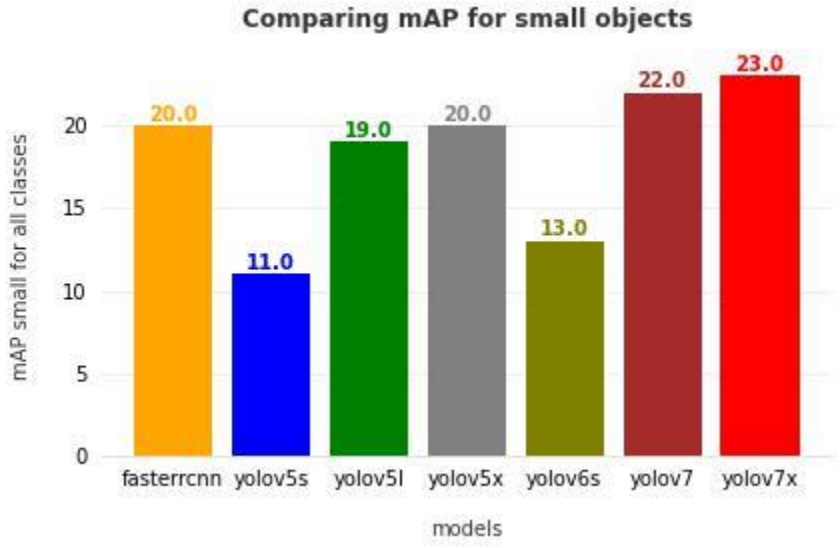


Figure 48 mAP for small size objects

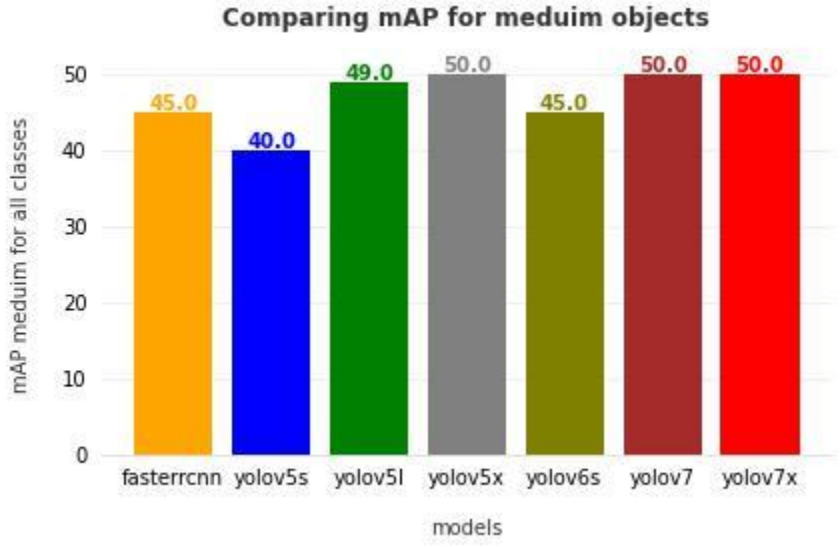


Figure 49 mAP for medium size objects

For medium sized objects, almost all models performed well as we can see in Figure 49Figure 49. In this case, yolov7, yolov7x, yolov5x and yolov5l had almost the same results. Faster RCNN and yolov6s, on the other hand, had the same mAP as well. However, yolov5s seems to struggle in detecting medium sized objects with the worst mAP among all.

On contrast, nearly all models achieved a good result in detecting large objects as we can see in Figure 50. The interesting thing here is the result of yolo6s, it overcomes all other models even yolov7 and yolov7x. In addition, all YOLO models are better than Faster RCNN in this case.

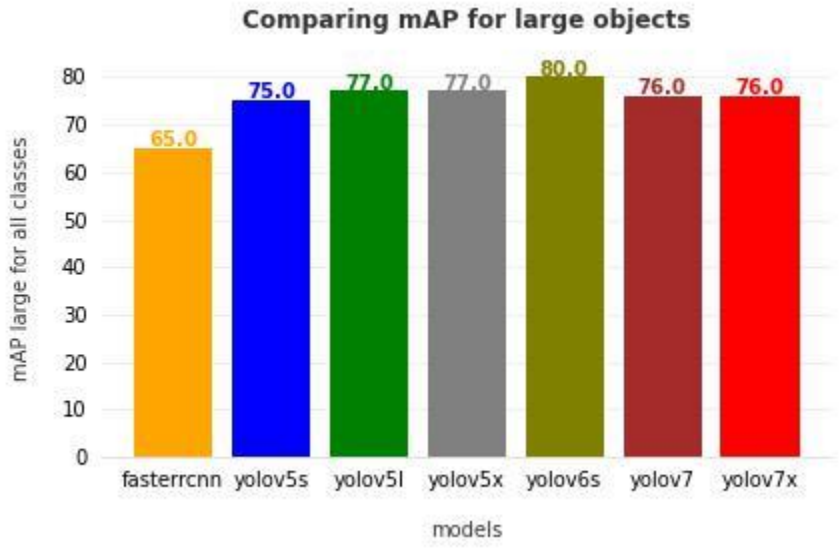


Figure 50 mAP for large objects

All the data that I have used to plot the accuracy comparison is available in Table 12. The cells with orange background represent the best result in each column.

Table 12 Models comparison

Models	COCO AP	PASCAL AP	mAP75	mAP small objects	mAP small objects	mAP large objects
Faster RCNN	31	64	27	20	45	65
YOLOv5s	26	55	21	11	40	75
YOLOv5l	34	66	29	19	49	77
YOLOv5x	35	68	30	20	**50**	77
YOLOv6s	27	55	23	13	45	**80**
YOLOv7	36	71	31	22	**50**	76
YOLOv7x	**37**	**72**	**32**	**23**	**50**	76

See Appendix C for comparison at class level for COCO AP.

For the segmentation task, I've got **48 COCO AP**, which is pretty good results. A small note here, in segmentation, we can use the mean average precision metric but instead of calculating the IoU using the bounding boxes we use the masks.

4.2 Speed Results

In this study, I have tested the algorithms on two different GPUs, which are **RTX5000** and **A100**. In order to do that, I've calculated the average time each model takes to predict and detect objects in images using **batch size equals one**. In other words, I have tested the inference time for each image per millisecond, thus the frame per second (fps) can be calculating by dividing 1000 over the inference time per millisecond. In this case, the less the inference time is the better.

In Figure 51, we can see a scatter plot for the inference time in ms vs COCO AP on RTX5000. Yolov6s and yolov5s are the fastest models but with poor AP. Faster RCNN is slow but with a good AP. Yolov7x is the best in terms of speed and accuracy together with 14.3 ms per image and 37 mAP. The price of RTX5000 GPU with 16GB memory is almost 3K$.

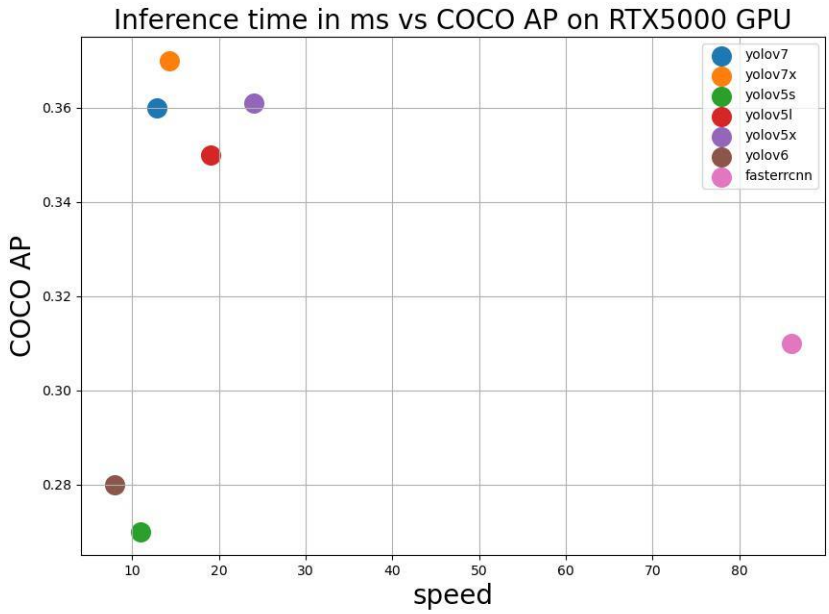


Figure 51 Inference time (ms/img) vs COCO AP on RTX5000 GPU

In Figure 52, we can see a scatter plot for the inference time in ms vs COCO AP for on A100 GPU. The speed of YOLO algorithms on this particular GPU is exterimly fast. However, as usual, yolov7x algorithm is the best if we want to take into consideration the speed and accuracy togheter, with 10.5 ms per image speed. Faster RCNN model speed on this GPU is 26 ms, but it's still not enough. The price of A100 GPU is almost 33K$.

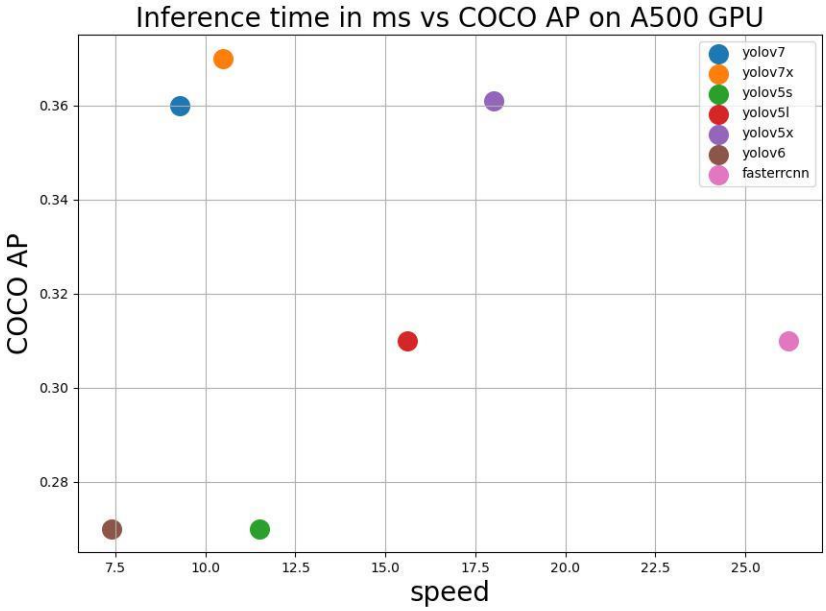


Figure 52 Inference time (ms/img) vs COCO AP on A100 GPU

In Figure 53, I ran the yolo family algorithms on the RTX5000 GPU with different input sizes. I tested the algorithms with 256×256, 384×384, 512×512, 640×640, 768×768, 896 ×896 pixels with **32 as batch size**, then I checked the mAP and the speed in ms. Yolov7 (yolov7 and yolov7x) models are the best among the others. Yolov6 perfomed poorly with low resolution. Yolov5 models performed well, but not enough to beat Yolov7 models.

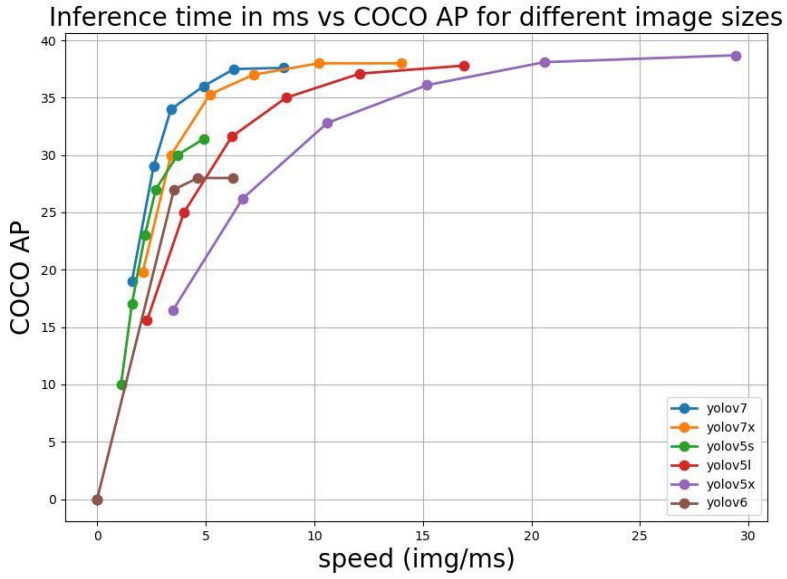


Figure 53 Running YOLO algorithms with different input image sizes

In table below, there is a general comparison between object detection algorithms in terms of number of parameters, accuracy and speed. As we can see, the biggest model I have used in this project is the yolov5x model with 86.7 million parameters. The most accurate model is yolov7x with 37 COCO AP (mAP [0.5:0.05:0.95]). Finally, the fastest model we have here is the YOLOv6s which takes 8 ms per image.

Table 13 General Comparison between object detection algorithms in terms of speed, accuracy and number of parameters

Models	Number of parameters (M)	COCO AP	Speed on RTX5000 GPU (ms)
Faster RCNN	43	31	86
YOLOv5s	7.2	26	11
YOLOv5l	46.5	34	19
YOLOv5x	**86.7**	35	24
YOLOv6s	17.2	27	**8**
YOLOv7	37	36	12.8
YOLOv7x	71	**37**	14.3

4.3 Results Demo

In this section I will display results on two pictures. More results can be found in this notebook on project's GitHub page. Some pictures from Mask RCNN model are also displayed. A video demo shows the result of running Yolov7x model in real time can be accessed through this link: https://www.youtube.com/watch?v=Iz9UvtoOEIY&t=4s

Figure 54 Demo 6 Faster RCNN

Figure 55 Demo 6 YOLO7X

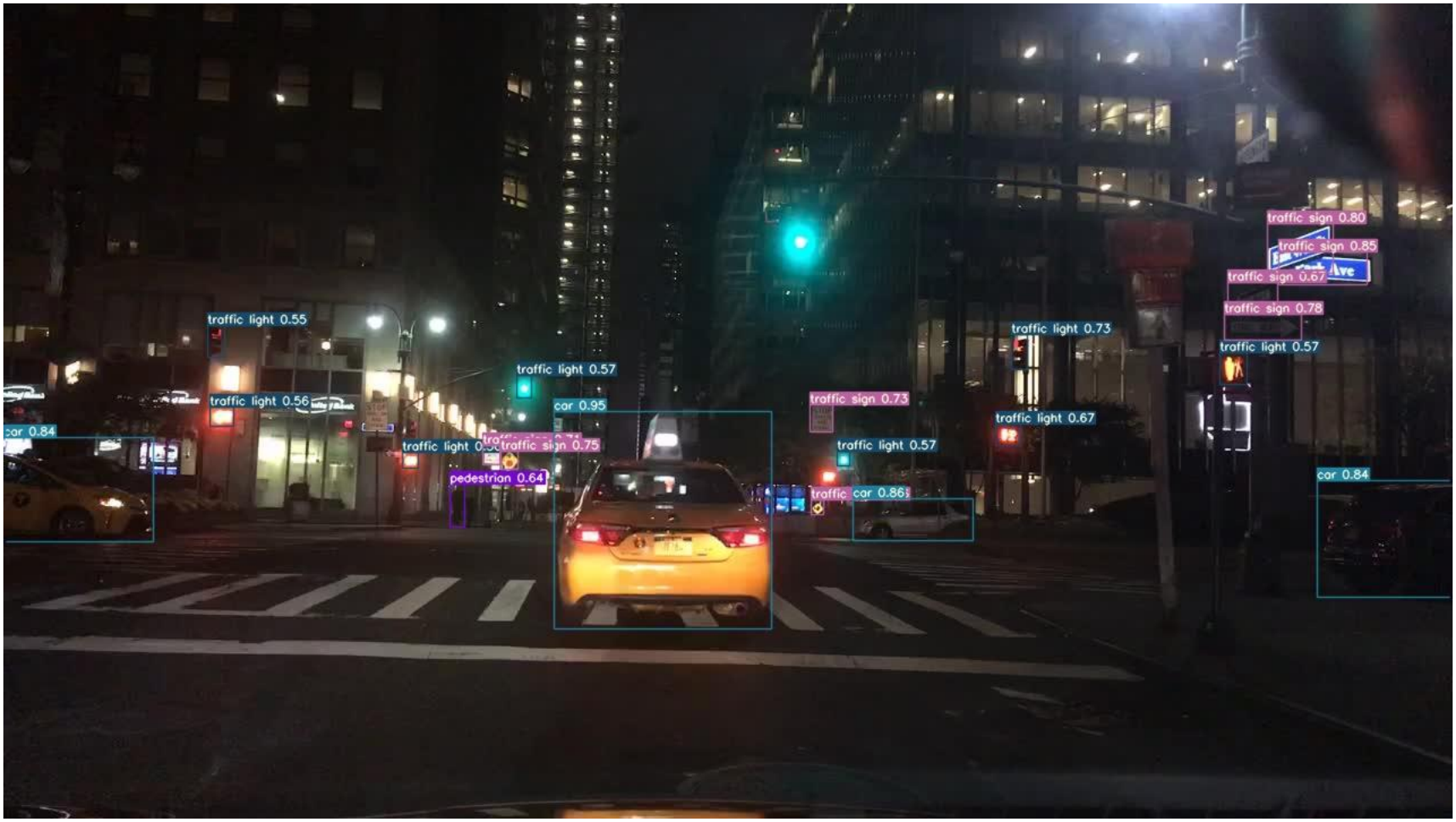


Figure 56 Demo 6 YOLO7

Figure 57 Demo 6 YOLO5s

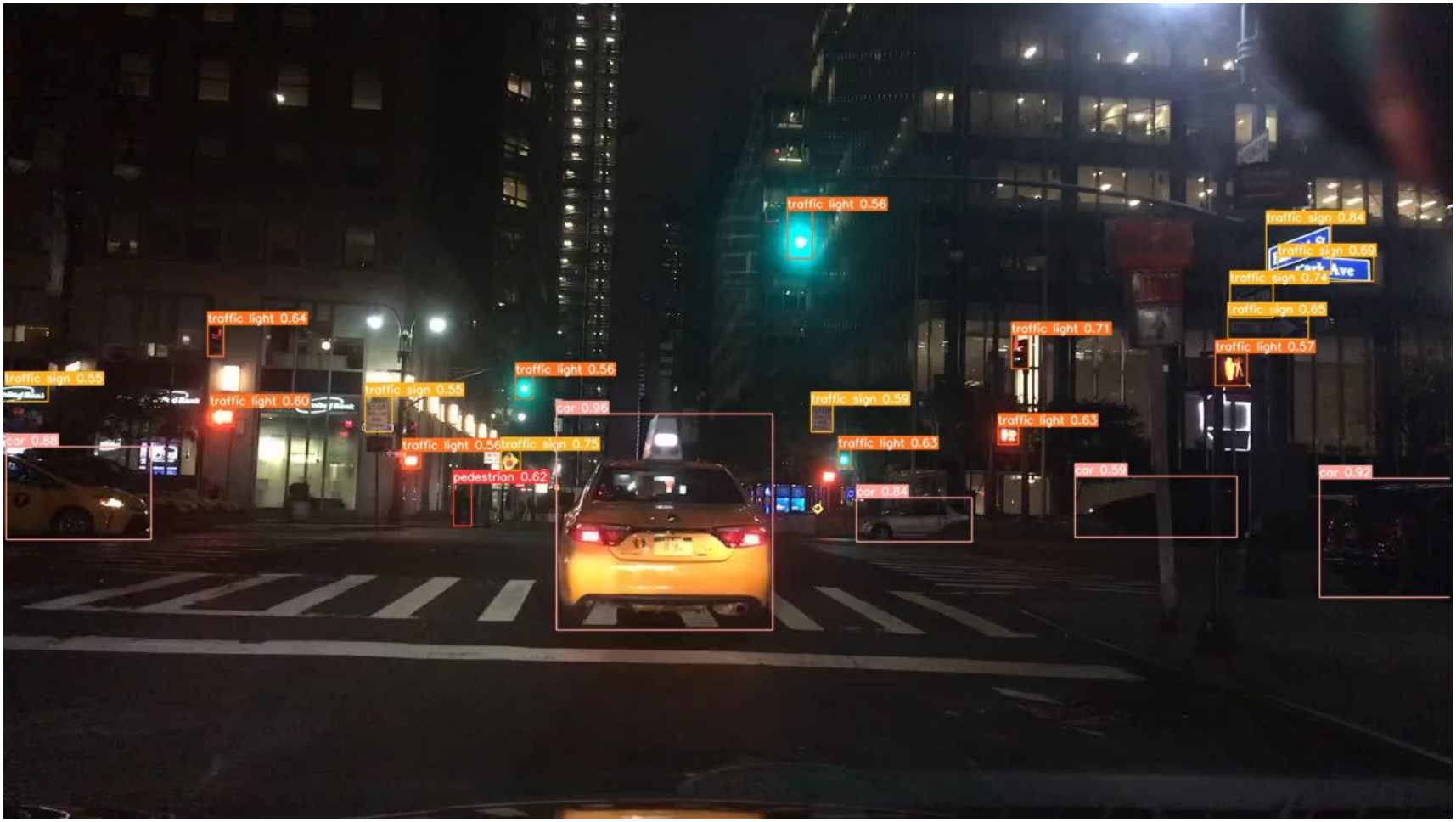


Figure 58 Demo 6 YOLO5l

Figure 59 Demo 6 YOLO5x

Figure 60 Demo 6 YOLO6s

Figure 61 Demo 15 FasterRCNN

Figure 62 Demo 15 YOLOv7x

Figure 63 Demo 15 YOLOv7

Figure 64 Demo 15 YOLOv5x

Figure 65 Demo 15 YOLO5l

Figure 66 Demo 15 YOLOv5s

Figure 67 Demo 15 YOLOv6s

Figure 68 Mask RCNN result 1

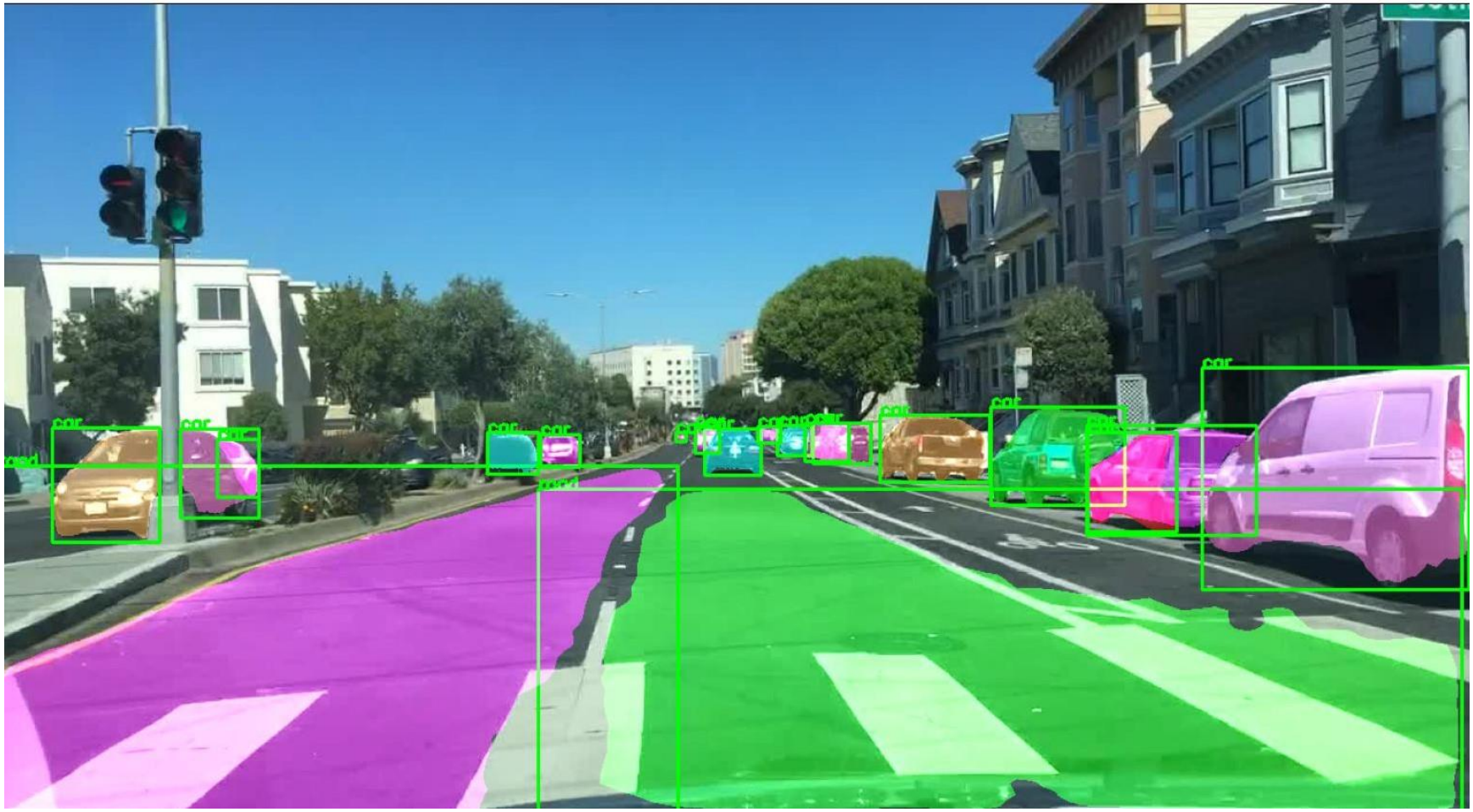


Figure 69 Mask RCNN result 2

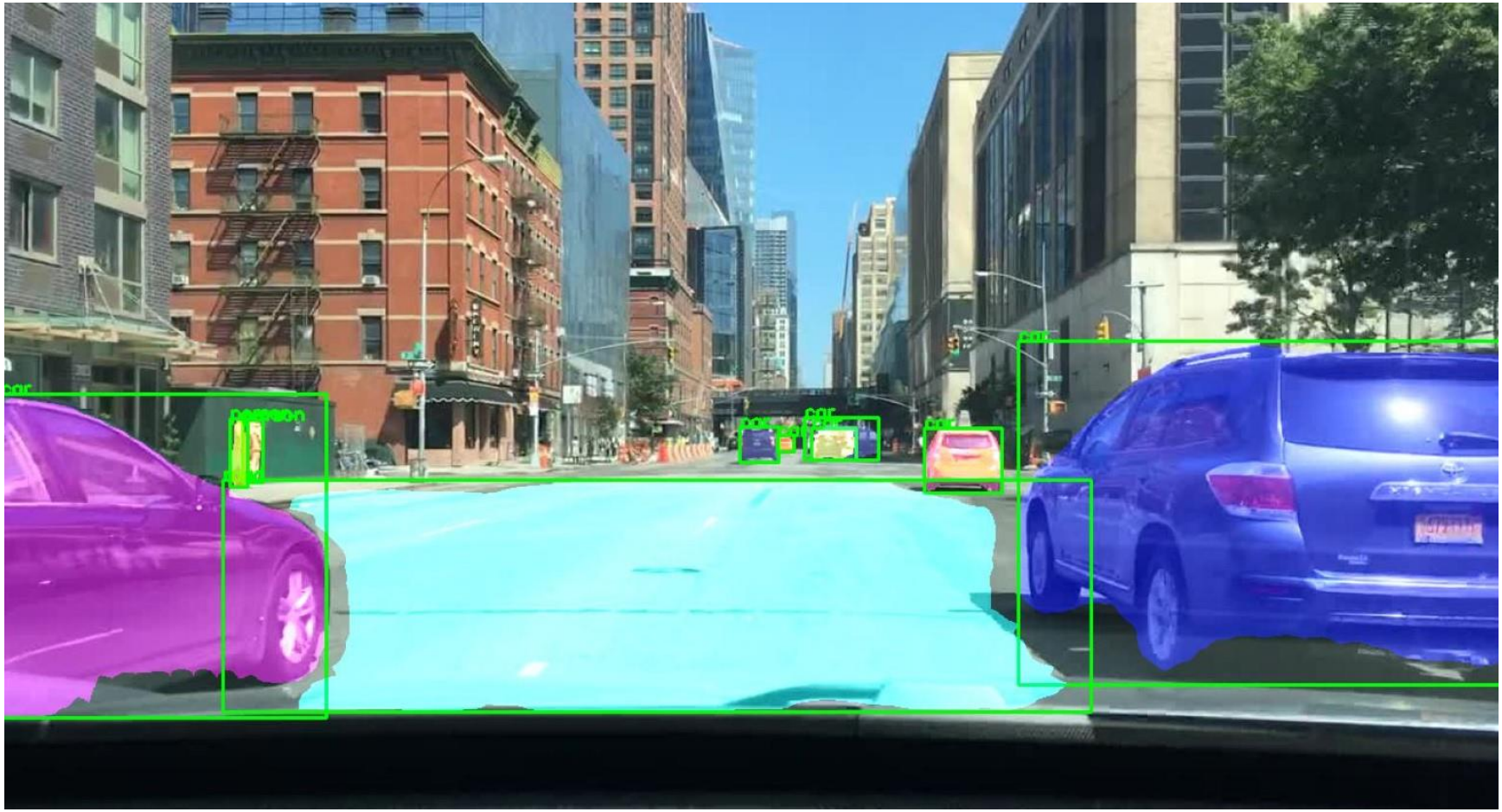


Figure 70 Mask RCNN result 3

5. Discussion

My main objective in this research was to analyze and study different object detection algorithms and see whether these algorithms can be employed in self-driving cars. In order to do that, I worked with BDD100K dataset and I have tested several models on it.

Table 14 General Comparison

Models	Number of parameters (M)	COCO AP	Speed on RTX5000 GPU (ms)	Frame per second (1 as batch size)
Faster RCNN	43	31	86	12
YOLOv5s	7.2	26	11	90
YOLOv5l	46.5	34	19	52
YOLOv5x	**86.7**	35	24	42
YOLOv6s	17.2	27	**8**	**125**
YOLOv7	37	36	12.8	78
YOLOv7x	71	**37**	14.3	70

Overall, we can say that the one stage object detectors are better than the two stages object detector in both speed and accuracy.

The results I have represented in Table 14 show that if we want to take into consideration the speed and accuracy together YOLOv7x model is the best among all other models I have used.

If we want to compare YOLOv7x and YOLOv5x, we see that yolov7x beat yolov5x in speed (14.3 and 24 ms/img respectively) and accuracy (37 and 35 COCO mAP respectively), with 71 million parameters for yolov7x less than yolov5x which has 86.7 million.

On the other hand, if we take a look at the number of parameters in yolov6s and yolov5s, we see that yolov6s has approximately 10 million parameters less than yolov5s, while the inference time per image for yolov6s and less than yolov5s.

After these comparisons, we can say that we must focus on the architecture design and the type of the backbone chosen in these algorithms, which play a major role in achieving better accuracy and fast inference.

Furthermore, if we compare Yolov7 with Yolov7x, we find that Yolov7x has twice as many parameters as Yolov7, even though Yolov7x's accuracy is slightly greater than Yolov7's (37 and 36 COCO AP, respectively), and Yolov7 is a little bit faster than Yolov7x.

We may come to the conclusion that increasing the size of the same model and the number of parameters will not always result in a significant improvement in the accuracy metric.

Regarding hardware selection, we observed that all models could operate in real time on an A100 GPU. However, A100 GPU is so expensive, and it is not ideal to employ it in a car. On the contrary, some algorithms like yolov7 and yolov7x performed really well on RTX5000 GPU. The RTX5000 GPU can be used in any self-driving machine, as it is reasonably priced.

The primary research question was: **"Are the current object detection algorithms efficient enough to be used in self-driving cars?"**

After all the interpretations and discussions, we can say yes, the current SOTA models are efficient enough to be used in self-driving industry. However, we didn't achieve our primary goal which is to build a fully autonomous vehicle that can drive and make sense of the world by itself without any human. In other words, even with these powerful detection algorithms we have now, we didn't make it to the level 5 of the driving automation I mentioned in section 1.1.

The reason why is we have not achieve yet the general intelligence, in all domains in AI such as computer vision, natural language processing, reinforcement learning, etc... Our current models still have lack of sense in some cases.

To put what I said into context, let us take an example. Let's say we trained a detection algorithm on thousands of objects, and we decided to employ it in self-driving car. The algorithm will perform extremely well on objects from the training set. The algorithm, however, will struggle with unfamiliar items like a **straw** on the road or a **strange animal**. He is unsure whether to pass over the straw, which poses no threat to anyone, or not pass over the strange animal.

6. Evaluation, Reflection and Conclusion

6.1. Evaluation

When I started this project, my aim was to work on a project which combines deep learning and self-driving. My first proposal was about modifying one of the exciting models and see whether my improvement can help in increasing speed and/or accuracy. However, it turns out that I don't have enough time to do that, also the authors and the contributors of the model weren't really helpful. As a result, I made some changes in the proposal and project objectives, but the core objective remains the same, which is to work on a project in the self-driving industry using deep learning approach.

I started this project by collecting and reading research papers related to the computer vision domain. This was one of the most important step in order to understand the algorithms used and how they differ from each other. During this phase, I wrote a small review for each paper, which helped me in writing this report after.

The second phase was to start working with the data. BDD100K is a large dataset, so I had to do some exploratory data analysis first in order to be close to my data. After that, I decided what are the classes I want to work on and what is the percentage or data I have to use.

The third step was to start implementing the framework by building the preprocessing pipeline and data loader. As I worked with different algorithms and each one requires a specific input format. I have prepared the data for each model. In addition, each model output the data in different manner, so I had to process the output to generate the final results and compare the models.

After that, I started evaluating each model in terms of speed and accuracy. Moreover, I have tested the models with two different hardware GPUs.

With each step, I documented the whole process with side notes, which helped me in writing the whole report.

One of the most challenging part was to train and evaluate the models each one in different environment.

The cloud environment I used is JarvisLab, as I mentioned earlier. I did not use the university's lab Robert Milner, because I have faced problems with the data storage, it wasn't enough for my big data. In addition, I faced a lot of problem with Hyperion, as I don't have full control over it, in other words I can't uninstall and install any library I want. Also I wasn't able to run more than one model at a time. All these reasons forced me chose JarvisLab, as it is easy to setup and use also it's cheap. I have paid 0.44$ per hour for RTX5000 GPU and 1.16$ per hour for A100 GPU (I got 10% student discount). In total, I paid about 150$ on the cloud to train and evaluate my models.

I primarily relied two books during the whole process of researching and writing my report in order to have a thorough understanding of the subject. Those book are Deep Learning and artificial intelligence a modern approach. I recommend everyone interested to know more about the field to read them.

6.2. Reflection

During this project, I learned all a lot of new information. I passed through the process of solving any machine learning or deep learning project, starting by researching, then collect the data, analyze it, and train models finally evaluate them. I also created a REST API for project.

I can now comprehend and handle all of the tasks involved in a machine learning project, including the challenges of a real-life implementation.

6.3. Future Work

During this study, I came through lot of papers and models that can be used in self-driving industry. Unfortunately, due the lack of time, I worked only with algorithms mentioned in this reports.

Some of these papers, one called YOLOP [62]. This model can perform several vision tasks at the same time. It would be good to explore the architecture of this model in the future.

In addition, I would like to explore the vision transformer [63] and see how the result would be on the self-driving industry. The transformers are usually used in the sequence analysis like in text generation, Chabot, machine translations, etc… However, some research shows that we can rely on transformer in vision domain.

References

[1] Sociery of Automotive Engineer (SAE), "SAE," 3 May 2021. [Online]. Available: https://www.sae.org/blog/sae-j3016-update. [Accessed 1 July 2022].

[2] Market Data Forecast, "Market Data Forecast," Market Data Forecast, January 2022. [Online]. Available: https://www.marketdataforecast.com/market-reports/artificial-intelligence-in-transportation-market. [Accessed 2022].

[3] T. M. Mitchell, Machine Learning (McGraw-Hill International Editions Computer Science Series), 1997.

[4] Fisher, "iris," 1936.

[5] T.-Y. a. M. M. a. B. S. a. H. J. a. P. P. a. R. D. a. D. P. a. Z. C. L. Lin, Microsoft COCO: Common Objects in Context, Springer, 2014.

[6] D. M. W. Wolpert, "No free lunch theorems for optimization," in *No free lunch theorems for optimization*, IEEE, 1997, p. 67–82.

[7] C. a. V. V. Cortes, "Support-vector networks," *Machine learning,* vol. 20, pp. 273--297, 1995.

[8] A. a. R. W. Makiewicz, "{Principal components analysis (PCA)," *Computers & Geosciences,* vol. 19, pp. 303--342, 1993.

[9] Deepmind, "deepmind," [Online]. Available: https://www.deepmind.com/research/highlighted-research/alphago. [Accessed 2022].

[10] Y. B. A. C. Ian Goodfellow, "The Curse od Dimensionality," in *Deep Learning*, Massachusetts Institute of Technology, 2016, pp. 155-156.

[11] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review,* vol. 65, no. American Psychological Association, p. 386, 1958.

[12] Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks,* vol. 61, pp. 85--117, 2015.

[13] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375,* 2018.

[14] I. Goodfellow and Y. Bengio, "Sigmoid Units for Bernoulli Output Distribution," in *Deep Learning*, 2016, pp. 182--184.

[15] I. Goodfellow and Y. Bengio, "Softmax Units for Miltinoulli Output Distribution," in *Deep Learning*, 2016, pp. 184--187.

[16] D. E. a. H. G. E. a. W. R. J. Rumelhart, "Learning representations by back-propagating errors," *nature,* vol. 323, no. Nature Publishing Group, pp. 533--536, 1986.

[17] N. a. H. G. a. K. A. a. S. I. a. S. R. Srivastava, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research,* vol. 15, no. JMLR. org, pp. 1929--1958, 2016.

[18] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747,* 2016.

[19] Y. B. A. C. Ian Goodfellow, "Batch and Minibatch Algorithm," in *Deep Learning*, 2016, pp. 227--282.

[20] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks,* vol. 12, no. Elsevier, pp. 145--151, 1999.

[21] J. a. H. E. a. S. Y. Duchi, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research,* vol. 12, 2011.

[22] D. P. a. B. J. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.

[23] W. a. W. Z. Rawat, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation,* vol. 29, no. MIT Press, pp. 2352--2449, 2017.

[24] A. R. a. P. M. a. R. S. Pathak, "Application of deep learning for object detection," *Procedia computer science,* vol. 132, no. Elsevier, pp. 1706--1717, 2018.

[25] A. a. O.-E. S. a. O. S. a. V.-M. V. a. G.-R. J. Garcia-Garcia, "A review on deep learning techniques applied to semantic segmentation," *arXiv preprint arXiv:1704.06857,* 2017.

[26] I. a. A. K. a. R. H. a. H. K. a. A. A. Jeelani, "Real-world mapping of gaze fixations using instance segmentation for road construction safety applications," *arXiv preprint arXiv:1901.11078,* 2019.

[27] J. a. L. Y. a. Z. Y. F. Zhu, "Object tracking in structured environments for video surveillance applications," *IEEE transactions on circuits and systems for video technology,* vol. 20, no. IEEE, pp. 223--235, 2009.

[28] S. a. X. J. Song, "Sliding shapes for 3d object detection in depth images," in *European conference on computer vision*, 2014, pp. 634--651.

[29] Y. a. B. Y. a. H. G. LeCun, "Deep learning," *nature,* vol. 521, no. Nature Publishing Group, pp. 436--444, 2015.

[30] K. a. Z. A. Simonyan, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* 2014.

[31] K. a. Z. X. a. R. S. a. S. J. He, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770--778.

[32] A. G. a. Z. M. a. C. B. a. K. D. a. W. W. a. W. T. a. A. M. a. A. H. Howard, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861,* 2017.

[33] A. a. W. C.-Y. a. L. H.-Y. M. Bochkovskiy, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934,* 2020.

[34] R. a. D. J. a. D. T. a. M. J. Girshick, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580--587.

[35] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440--1448.

[36] S. a. H. K. a. G. R. a. S. J. Ren, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems,* vol. 28, p. 2015.

[37] J. a. D. W. a. S. R. a. L. L.-J. a. L. K. a. F.-F. L. Deng, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248--255.

[38] J. a. D. S. a. G. R. a. F. A. Redmon, "You only look once: Unified, real-time object detection," *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 779--788, 2016.

[39] J. a. F. A. Redmon, "YOLO9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263--7271.

[40] S. a. S. C. Ioffe, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, 2015, pp. 448--456.

[41] J. a. F. A. Redmon, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767,* 2018.

[42] W. a. A. D. a. E. D. a. S. C. a. R. S. a. F. C.-Y. a. B. A. C. Liu, "Ssd: Single shot multibox detector," in *European conference on computer vision*, 2016, pp. 21--37.

[43] A. a. W. C.-Y. a. L. H.-Y. M. Bochkovskiy, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934,* 2020.

[44] T.-Y. a. M. M. a. B. S. a. H. J. a. P. P. a. R. D. a. D. P. a. Z. C. L. Lin, "Microsoft coco: Common objects in context," in *European conference on computer vision*, 2014, pp. 740--755.

[45] S. a. H. D. a. O. S. J. a. C. S. a. C. J. a. Y. Y. Yun, "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6023--6032.

[46] M. A. a. N. S. a. R. M. a. B. N. a. W. Y. Islam, "Label refinement network for coarse-to-fine semantic segmentation," *arXiv preprint arXiv:1703.00551,* 2017.

[47] D. Misra, "Mish: A self regularized non-monotonic neural activation function," *arXiv preprint arXiv:1908.08681,* 2019.

[48] K. a. Z. X. a. R. S. a. S. J. He, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence,* vol. 37, pp. 1904--1916, 2015.

[49] C.-Y. a. L. H.-Y. M. a. W. Y.-H. a. C. P.-Y. a. H. J.-W. a. Y. I.-H. Wang, "CSPNet: A new backbone that can enhance learning capability of CNN," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, 2020, pp. 390--391.

[50] S. a. Q. L. a. Q. H. a. S. J. a. J. J. Liu, "Path aggregation network for instance segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8759--8768.

[51] M. a. P. R. a. L. Q. V. Tan, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10781--10790.

[52] C.-Y. a. B. A. a. L. H.-Y. M. Wang, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *arXiv preprint arXiv:2207.02696,* 2022.

[53] C. a. L. L. a. J. H. a. W. K. a. G. Y. a. L. L. a. K. Z. a. L. Q. a. C. M. a. N. W. a. o. Li, "YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications," *arXiv preprint arXiv:2209.02976,* 2022.

[54] Z. a. L. S. a. W. F. a. L. Z. a. S. J. Ge, "Yolox: Exceeding yolo series in 2021," *arXiv preprint arXiv:2107.08430,* 2021.

[55] J. a. S. E. a. D. T. Long, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431--3440.

[56] O. a. F. P. a. B. T. Ronneberger, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, 2015, pp. 234--241.

[57] L.-C. a. P. G. a. K. I. a. M. K. a. Y. A. L. Chen, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence,* vol. 40, p. 2017, 834--848.

[58] K. a. G. G. a. D. P. a. G. R. He, "Mask r-cn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961--2969.

[59] F. a. C. H. a. W. X. a. X. W. a. C. Y. a. L. F. a. M. V. a. D. T. Yu, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 2636--2645.

[60] A. a. G. S. a. M. F. a. L. A. a. B. J. a. C. G. a. K. T. a. L. Z. a. G. N. a. A. L. a. D. A. a. K. A. a. Y. E. a. D. Z. Paszke, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024--8035.

[61] The PyTorch Lightning team, "PyTorch Lightning," PyTorch Lightning, [Online]. Available: https://www.pytorchlightning.ai.

[62] D. W. a. M. L. a. W. Z. a. X. Wang, "YOLOP: You Only Look Once for Panoptic Driving Perception," 2021.

[63] A. a. B. L. a. K. A. a. W. D. a. Z. X. a. U. T. a. D. M. a. M. M. a. H. G. a. G. S. a. o. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929,* 2020.

Appendix

Project Proposal

1 Introduction and objectives

In recent years, automation industry has expanded and we can see its applications in every corner in our lives. Autonomous vehicles is one of its applications, which is the major topic of my dissertation. Before we continue, lets understand the **autonomous vehicle** industry and its resolution. First of all, an autonomous vehicle or autonomous car is like any intelligent agent, has the ability of sensing its environment and operate without human interaction, in other words, the human is not required to interfere in any stage nor to be present in the vehicle at all.

The Society of Automotive Engineer (SAE) has define six levels of driving automation, ranging from 0 (fully manual) to level 5 (fully autonomous) [1] (see figure 1). Autonomous cars heavily rely on sensors, actuators, algorithms, machine learning systems, and huge computing power to execute software.

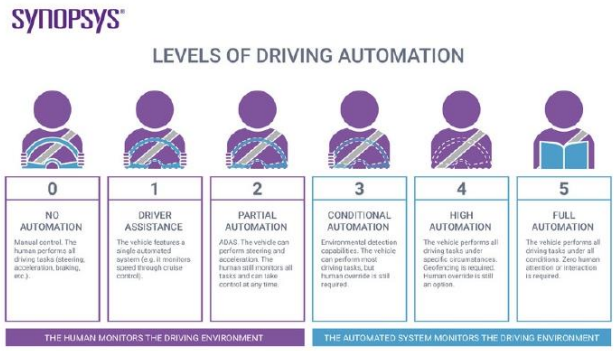


Figure 1: Six levels of driving automation, source

With today's data and massive computational power, **Artificial Intelligence** is poised to take the automobile industry by storm, accelerating the development of level 4 and 5 autonomous cars. From a financial side, the Vantahe Market Research reported thet the global automotive artificial intelligence market is forcast to grow to US$7,676.92 Million by 2028, with a CAGR of 31.30% in the 2022-2028 period [2]. These statistics give us a clear idea of how this industry is growing really fast and its importance in the future.

In order to make the autonomous vehicle really autonomous, we have to make it see the world like the human see, at least the essential part of it. If we think about it, what should the car see in order to operate in smooth way? First of all it has to avoid object obviously, make a decision whether the road is drivable or not, finally the position of the line. In other words, in each autonomous car we must have a **panoptic driving system** (see figure 2). So we can say that our goal is to build AI model to perform object detection, segment the derivable area and lane detection on **Real Time**.

Figure 2: Panoptic driving system

The purpose of this project is to analyse and study the current State of the art algorithms and see if they are efficient enough to work in the self driving car industry.

The beneficiaries of this project are:

- Anyone working in the self driving car industry.
- Researchers in the Computer Vision community.
- Anyone working in the Robotics industry.

The question posed for the research project is as follows:

"Are the current object detection algorithms efficient enough to be used in self driving cars?"

2 Critical Context

To make the vehicle navigate, the AI model need to extract visual information from the environments, and as we discussed earlier, the main three parts are:

- Object detection (cars, persons...).
- Drivable area segmentation.
- Lanes detection.

There are numerous state-of-the-art models to solve each problem individually and their are very good at it. However, running the models one by one can take a long time, since we want the results in real-time. Now I will do a small literature review for some critical models.

2.1 Object Detection

Object detection is one of the most interesting application of AI, it allows to classify and locate objects in images and videos. In recent years, deep learning made object detection tasks relatively easy, if we have the data and computational power of course. In order to do object detection we have to complete two tasks:

- **Classification**: Classify the objects (e.g. car, person, bicycle...).
- **Regression**: Predict the position of bounding boxes (e.g. x1,y1,x2,y2)

Currently, the family of object detection is divided to two parts, first one uses two-stages to gets output, the others are only one-stage methods.

2.1.1 Two Stage Detectors

In two stage detectors, and as the name suggest, they are using two steps to perform object detection. First step is to extract the possible positions where we can have an object, and second step is to classify this object. In [3] and [4] they were using selective search algorithm to extract region proposals followed by **Convnet** and a classifier to classify the objects. However this approach was very slow. In [5], they came with an object detection algorithm called **Faster R-CNN** (see figure 3) that eliminate the selective search, instead the network now learns the region proposals. There is a separate network used to predict the region proposals, after that the predicted region proposals are reshaped using **RoI pooling layer**, which is used to classify the image within the proposed region and predict the bounding boxes.

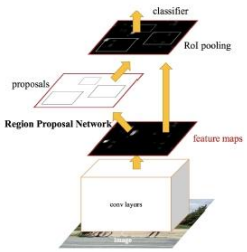


Figure 3: Faster R-CNN architecture [5]

2.1.2 One Stage Detectors

You Only Look Once or **YOLO** [6] family are a group of algorithm much different from the region algorithm (see fugure 4). In **YOLO**, a single convolutions network predicts the bounding boxes and classify the object which makes it a lot faster than all two-stage object detection. The bounding boxes having the class probability above a specific threshold is selected and used to locate the object within the images. However, **YOLO** struggles to detect small object in an image.

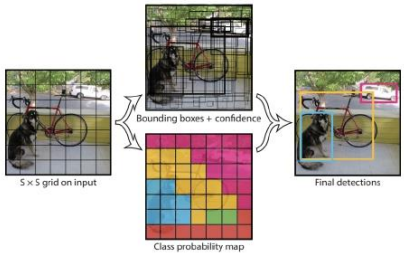


Figure 4: YOLO model [6]

2.2 Segmentation

Segmentation is the task of of labelling each pixel of the image into predefined set of classes. It has lot of applications and it can be used in several fields such as image compression, scene understanding, healthcare... Mainly we have two types of image segmentation: **Semantic segmentation** and **Instance segmentation**. The different between them is in **semantic** the goal is to associates every pixel of an image with a class label such as person, car... While **instance** treats objects with the same class as multiple instance.

There are number of CNN-based models to do instance and semantic tasks. The general architecture of the models consists of series of conv layers with pooling layers for **downsampling**, after that **upsampling** techniques are used, this kind of architecture is know as **encoder-decoder** (see figure 5). For instance, **FCN** [7] was the first model to solve the problem of semantic segmentation. On the other hand, **Mask R-CNN** [8] was the first model to handle to problem of instance segmentation.

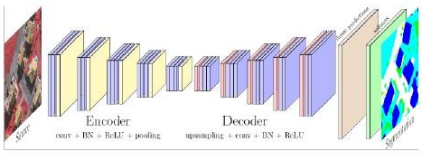


Figure 5: Encoder-Decoder Architecture (Source)

2.3 Multi-Task Approach

The purpose of multi task learning is to learn better representation by sharing information among multiple tasks. In our case, in image analysis, CNN-based multitask can share convolutions layers. **Mask-RCNN** [8] is a multi task model based on **Faster RCNN** [5] to performs instance segmentation by combining object detection and semantic segmentation (see figure 6 for example).

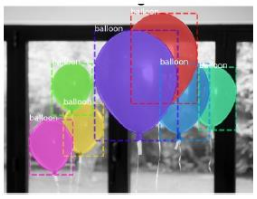


Figure 6: Mask R-CNN example

3 Methodology

3.1 Collecting Data

To begin with, the data is the most important thing in AI, because from it the model can actually learn something. I have only three months to complete this project (meat be extend to six months if I got an internship), therefor I must find a well annotated dataset because I don't want to waste time on collecting data and labeling it.
After searching, I found numerous of dataset that are used in this field (autonomous driving field). However, **The Berkeley DeepDrive** [9] dataset from **UC Berkeley** is the best one for my project. It consists of more than 100K HD videos and images. It is well annotated dataset (see figure 8), it has annotations for segmentation, object detection, drivable areas and lane markings, which is exactly what I am looking for. It has 10 classes for object detection and they are distributed as we can see in the figure 7:

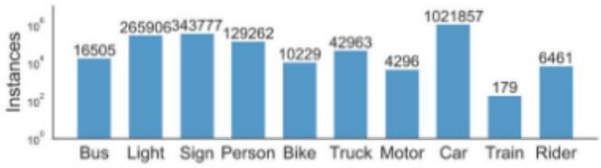


Figure 7: Categories distribution in BDD100K

The Data in **BDD100K** are collected from 4 different locations (SF, Berkeley, Bay Area, New York). The dataset possess geographic, environmental and weather diversity, which is useful for training models so that they are less likely to be surprised by new conditions.

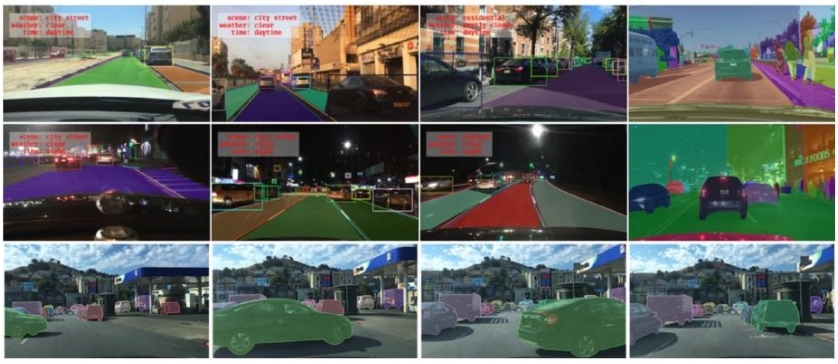


Figure 8: Example from BDD100K dataset.

3.2 Tools

In this project, I will be using **Python** as a programming language, as it has a lot of libraries that I can use in AI/ML field. **PyTorch** is the deep learning framework I will be using since it is more pythonic than other libraries that as Tensorflow, moreover, it has **torchvision** repository that has a lot of pre-defined and ready to use models and backbones such as resnet [10].

For computational power, **Hyperion** is a good choice, however, a lot of time the script stopped for no reason while running on it, so, and like I did in my coursework, I may use **JarvisLabs**, which is a platform where I can rent a GPU from my choice, with storage from my choice as well, I can get 20% discount as a student.

3.3 Evaluation

To evaluate the model, again we have three different tasks, so we have to evaluate each task separately. **Mean Average Processing (mAP)** is the metric for evaluating object detection. For drivable area segmentation and lane detection, **Mean Intersection over union (mIoU)** is the one used to evaluate such problems.

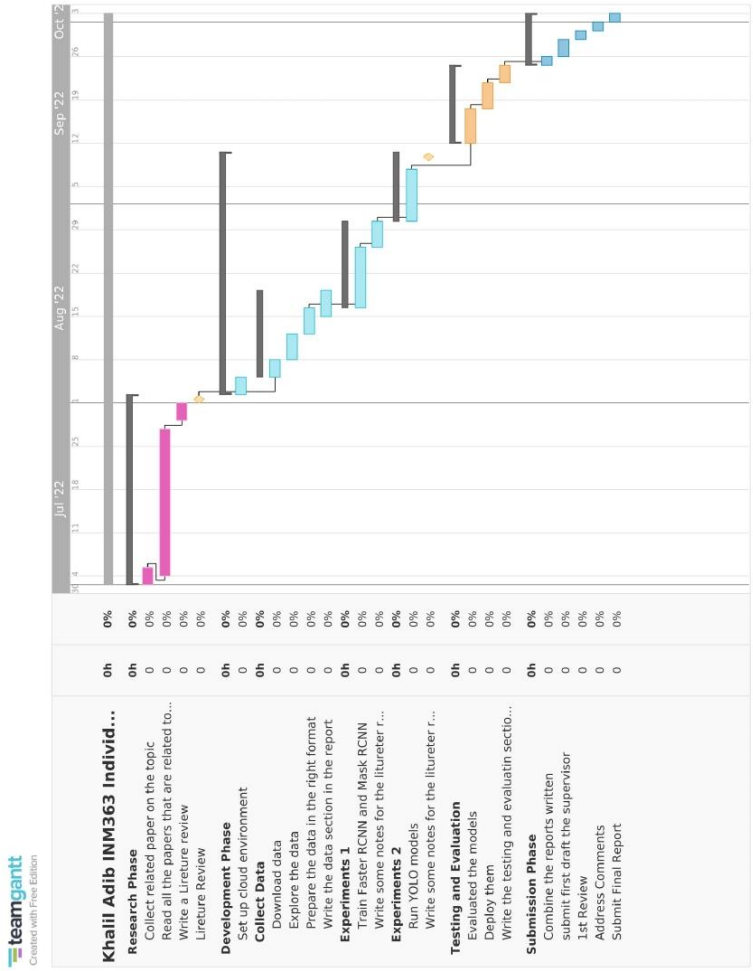
4 Risks

In this section, I will list some risks that can encounter me while I am working on the projects in the table bellow.

Description	Likelihood (1 – 3)	Consequence (1 – 5)	Impact (L x C)	Mitigation
The Data is now open source, what can happen if it won't be available in the future?	2	4	8	There are other open source datasets can be used
Hyperion is always give some errors related to CUDA a lot of times	3	3	6	We can use other platforms which are paid, however there is discount for students
Online Platform are expensive	2	4	8	Use Robert Milner Lab
Code lost or accidently written over	2	5	10	Use version control software (e.g.: Git and GitHub)
Network takes too long to converges	3	4	7	Show the results that I've achieved, or rent more powerful GPUs
Original models code aren't accessible anymore	2	5	10	Build something similar by combining available models

Figure 9: Risks

5 Work Plan


Jul '22
Aug '22
Sep '22
Oct '2
Khalil Adib INM363 Individ...
Research Phase
Collect related paper on the topic
Read all the papers that are related to...
Write a Lireture review
Lireture Review
Development Phase
Set up cloud environment
Collect Data
Download data
Explore the data
Prepare the data in the right format
Write the data section in the report
Experiments 1
Train Faster RCNN and Mask RCNN
Write some notes for the litureter r...
Experiments 2
Run YOLO models
Write some notes for the litureter r...
Testing and Evaluation
Evaluated the models
Deploy them
Write the testing and evaluatin sectio...
Submission Phase
Combine the reports written
submit first draft the supervisor
1st Review
Address Comments
Submit Final Report
teamgantt
Created with Free Edition

Ethical, Legal Professional Issues

A.1 If you answer YES to any of the questions in this block, you must apply to an appropriate external ethics committee for approval and log this approval as an External Application through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
1.1	Does your research require approval from the National Research Ethics Service (NRES)? *e.g. because you are recruiting current NHS patients or staff?* *If you are unsure try - https://www.hra.nhs.uk/approvals-amendments/what-approvals-do-i-need/*	**NO**
1.2	Will you recruit participants who fall under the auspices of the Mental Capacity Act? *Such research needs to be approved by an external ethics committee such as NRES or the Social Care Research Ethics Committee - http://www.scie.org.uk/research/ethics-committee/*	**NO**
1.3	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation? *Such research needs to be authorised by the ethics approval system of the National Offender Management Service.*	**NO**
A.2 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee, you must apply for approval from the Senate Research Ethics Committee (SREC) through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
2.1	Does your research involve participants who are unable to give informed consent? *For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf.*	**NO**
2.2	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	**NO**
2.3	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	**NO**
2.4	Does your project involve participants disclosing information about special category or sensitive subjects? *For example, but not limited to: racial or ethnic origin; political opinions; religious beliefs; trade union membership; physical or mental health; sexual life; criminal offences and proceedings*	**NO**
2.5	Does your research involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning that affects the area in which you will study?	**NO**

Ethical, Legal Professional Issues

	Please check the latest guidance from the FCO - http://www.fco.gov.uk/en/	
2.6	Does your research involve invasive or intrusive procedures? *These may include, but are not limited to, electrical stimulation, heat, cold or bruising.*	**NO**
2.7	Does your research involve animals?	**NO**
2.8	Does your research involve the administration of drugs, placebos or other substances to study participants?	**NO**
A.3 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee or the SREC, you must apply for approval from the Computer Science Research Ethics Committee (CSREC) through Research Ethics Online - https://ethics.city.ac.uk/ **Depending on the level of risk associated with your application, it may be referred to the Senate Research Ethics Committee.**		*Delete as appropriate*
3.1	Does your research involve participants who are under the age of 18?	**NO**
3.2	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? *This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.*	**NO**
3.3	Are participants recruited because they are staff or students of City, University of London? *For example, students studying on a particular course or module.* *If yes, then approval is also required from the Head of Department or Programme Director.*	**NO**
3.4	Does your research involve intentional deception of participants?	**NO**
3.5	Does your research involve participants taking part without their informed consent?	**NO**
3.5	Is the risk posed to participants greater than that in normal working life?	**NO**
3.7	Is the risk posed to you, the researcher(s), greater than that in normal working life?	**NO**
A.4 If you answer YES to the following question and your answers to all other questions in sections A1, A2 and A3 are NO, then your project is deemed to be of MINIMAL RISK. **If this is the case, then you can apply for approval through your supervisor under PROPORTIONATE REVIEW. You do so by completing PART B of this form.** **If you have answered NO to all questions on this form, then your project does not require ethical approval. You should submit and retain this form as evidence of this.**		*Delete as appropriate*
4	Does your project involve human participants or their identifiable personal data? *For example, as interviewees, respondents to a survey or participants in testing.*	**NO**

References

[1] SAE, "Sae levels of driving automation™ refined for clarity and international audience." https://www.sae.org/blog/sae-j3016-update.

[2] E. Kunz, "Global automotive artificial intelligence (ai) market." (source).

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[4] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[7] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[8] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[9] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2636–2645, 2020.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Appendix A

I will go through an example on how to calculate mean average precision for object detection. Let's say we want to detect objects in Figure 72. Figure 71 is the ground truth and Figure 73 is the predicted.

Note: this example is taken from this webpage.

Figure 72 Original Image

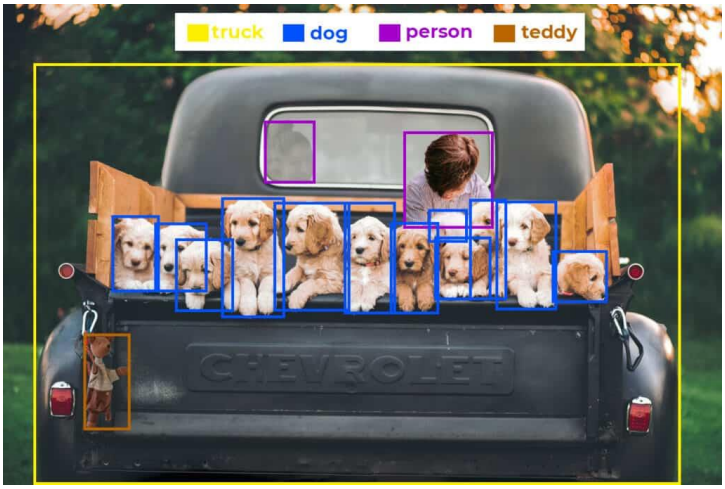


Figure 71 Ground truth

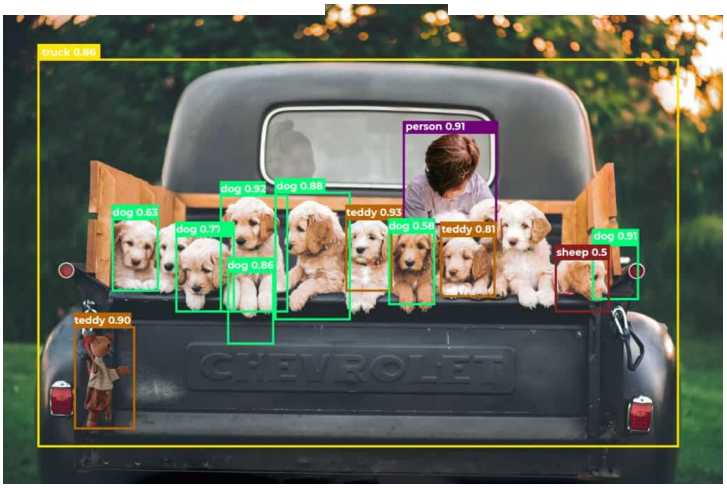


Figure 73 Predicted boxes and classes

We have four different classes which are: person, dog, truck and teddy. The average precision is calculated for each class separately. Let us take dog as an example. In Table 15 we can see the confidence score of every dog plus the correctness of the prediction.

Table 15 Detection for every dog with confidence scores and matches Gt and IoU

Detections							
Conf.	0.63	0.77	0.92	0.86	0.88	0.58	0.91
Matches GT by IoU?	TP	TP	TP	FP	TP	TP	FP

The next step is to calculate the precision and recall. In order to that, we have to follow the following steps:

- Sort the confidence score in descending order.
- Calculate the cumulative TP and FP.
- Calculate the precision and recall for each raw.

Table 16 illustrates how to calculate the precision and recall for dog class.

Table 16 Precision and Recall for dog class

Preds.	Conf.	Matches	Cumulative TP	Cumulative FP	Precision	Recall
	0.92	TP	1	0	1/(1+0) = 1	1/16 = 0.08
	0.91	FP	1	1	1/(1+1) = 0.5	1/16 = 0.08
	0.88	TP	2	1	2/(2+3) = 0.66	2/16 = 0.16
	0.86	FP	2	2	0.5	0.16
	0.77	TP	3	2	0.6	0.25
	0.63	TP	4	2	0.66	0.33
	0.58	TP	5	2	0.71	0.41

The next step is to plot the precision and recall graph and calculate the mAP using the interpolation method, as seen in Figure 74. The interpolation method were introduced in Pascal VOC 2007 challenge, where the precision values are interpolated across 11 values:

0, 0.1, 0.2…1. We take the maximum precision value to the right at each interpolation point.

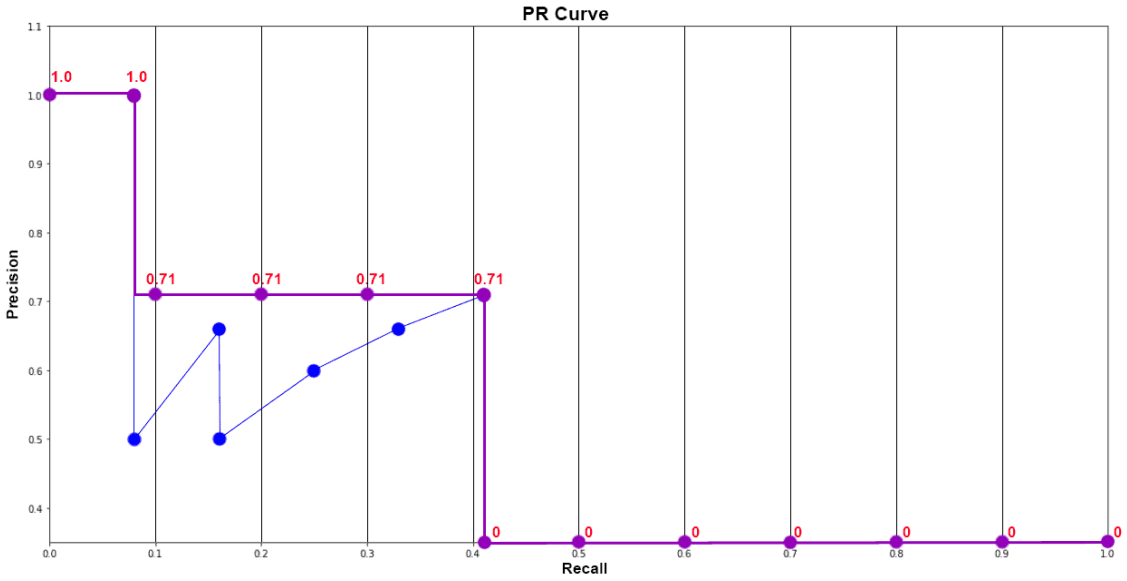


Figure 74 Precision and Recall with interpolation

As a result, the $AP_{dog} = \frac{1}{11} \times \sum interpolation\ values = \frac{1}{11} \times (1 + 0.71 + 0.71 + 0.71 + 0.71 + 0 + 0 + 0 + 0 + 0 + 0) = 0.349 = 34.9\%$

If we do that for every class we will get the Table 17:

Table 17 AP for all classes

Class	dog	person	sheep	truck	teddy
AP	0.349	0.545	0.0	1	0.5

Finally, the mAP is the mean of all the AP which is equal to 47.88% is this case.

Appendix B

```
# method to convert xyxy to xywh yolo format
def convert_pascal_to_yolo(xmin, ymin, xmax, ymax, img_w, img_h):
    dw = 1. / (img_w)
    dh = 1. / (img_h)
    x = (xmin + xmax) / 2.0 - 1
    y = (ymin + ymax) / 2.0 - 1
    w = xmax - xmin
    h = ymax - ymin
    x = round(x * dw, 4)
    w = round(w * dw, 4)
    y = round(y * dh, 4)
    h = round(h * dh, 4)
    return (x, y, w, h)


def create_yolo_annotation(bdd):
    """
    method to convert to create new db from the bdd.db object that have
yolo format
    :param bdd: BDD class instance
    :return: yolo_deque object that has all the annotation in yolo
format
    """
    yolo_deque = deque()
    # YOLO_Format = namedtuple("YOLO", "cls x y w h")
    print("Start converting.")
    for item in tqdm(bdd.db):
        image_path = item['image_path']
        image = Image.open(image_path)
        width, height = image.size
        boxes = item['bboxes']
        classes = item['classes']
        yolo_boxes = []
        for i, box in enumerate(boxes):
            cls = classes[i]
            x, y, w, h = convert_pascal_to_yolo(box[0], box[1], box[2],
box[3], width, height)
            # yolo_format = YOLO_Format(cls, x, y, w, h)
            yolo_boxes.append([cls, x, y, w, h])

        yolo_deque.append({
            'image_name': item['image_path'].split('\\')[-1],
            'image_path': image_path,
            'width': width,
            'height': height,
            'yolo_boxes': yolo_boxes
        })
    print("Finish from converting")
    return yolo_deque


def move_files(yolo_deque, images_folder_destination,
labels_folder_destination):
    """
```

```
    method to copy the files from the original source to a new
destination
    :param yolo_deque: deque object, which is the output of the
create_yolo_annotation method
    :param images_folder_destination: the path to the folder where we
want to copy the images to
    :param labels_folder_destination: the path to the folder where we
want to create the labels (txt files)
    :return: None
    """
    assert os.path.isdir(images_folder_destination), "Folder does not
exist!"
    assert os.path.isdir(labels_folder_destination), "Folder does not
exist!"

    print("Start copying files.")
    for item in tqdm(yolo_deque):
        # shutil.copy(item['image_path'],
os.path.join(folder_destination, 'images', stage))
        # create_annotation_file(item, folder_destination, stage)
        shutil.copy(item['image_path'], images_folder_destination)
        create_annotation_file(item, labels_folder_destination)

    print(f"All images are in {images_folder_destination} and all
labels are in {labels_folder_destination}.")


def create_annotation_file(yolo_item, labels_folder_destination):
    text_file_name = yolo_item['image_name'].replace('.jpg',
'.txt').split('/')[-1]
    file_path = os.path.join(labels_folder_destination, text_file_name)
    file = open(file_path, 'a')
    for line in yolo_item['yolo_boxes']:
        file.write(" ".join(str(item) for item in line))
        file.write('\n')
    file.close()
```

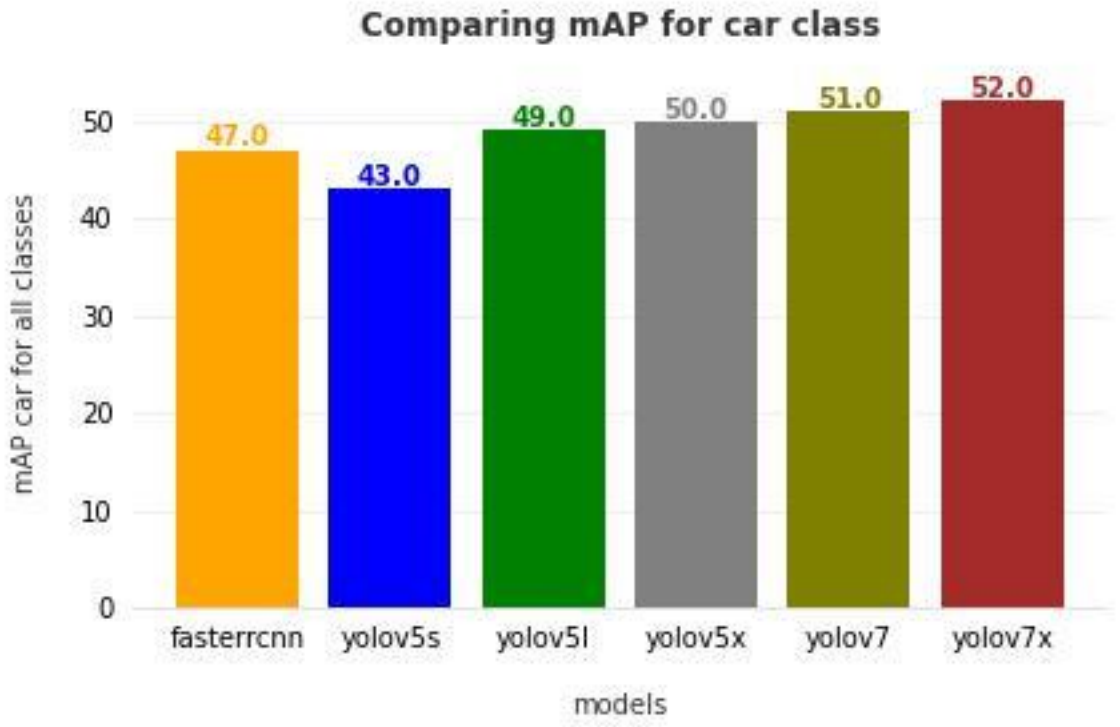
```
import os
import yaml
import argparse
from src.config.defaults import cfg
from src.utils.utils import *
from src.dataset.bdd_detetcion import BDDDetection

if __name__ == '__main__':
    # Define the parser
    parser = argparse.ArgumentParser()
    parser.add_argument('--data', type=str, default='', help='')
    parser.add_argument('--yolo_version', type=str, default='yolov5',
choices=['yolov5', 'yolov7'], help='which '

'version of '

'yolo do you '

'want to use')
```

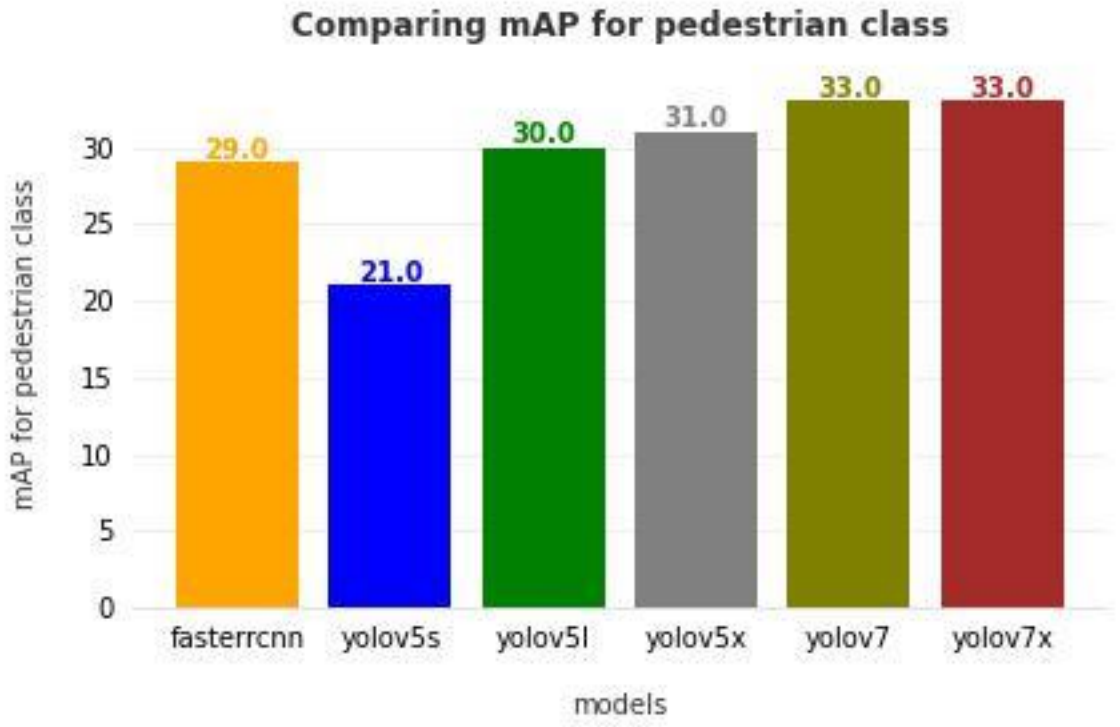
```
    parser.add_argument('--dataset_path', type=str, help='The path to
the dataset folder where you want to upload the '
                                                         'data')

    # Fetch the params from the parser
    args = parser.parse_args()
    version = args.yolo_version
    dataset_path = args.dataset_path

    with open(args.data, 'r') as f:
        data = yaml.safe_load(f)  # data from .yaml file

    obj_cls = data['classes']  # the classes we want to work one
    relative_path = data['relative_path']  # relative path to the
dataset

    ######################################## Datasets
########################################
    bdd_train_params = {
        'cfg': cfg,
        'relative_path': relative_path,
        'stage': 'train',
        'obj_cls': obj_cls,
    }

    bdd_train = BDDDetection(**bdd_train_params)

    bdd_val_params = {
        'cfg': cfg,
        'relative_path': relative_path,
        'stage': 'val',
        'obj_cls': obj_cls,
    }

    bdd_val = BDDDetection(**bdd_val_params)

    bdd_test_params = {
        'cfg': cfg,
        'stage': 'test',
        'relative_path': relative_path,
        'obj_cls': obj_cls,
    }

    bdd_test = BDDDetection(**bdd_test_params)
    print(50*'#')
    print(
        f"We have {len(bdd_train)} training images, {len(bdd_val)}
validation images and {len(bdd_test)} test images.")

    print(50 * '#')
    ######################################## Prepare Data
########################################

    """
    Yolov5 [https://github.com/ultralytics/yolov5] takes the data in
YOLO annotation format
```

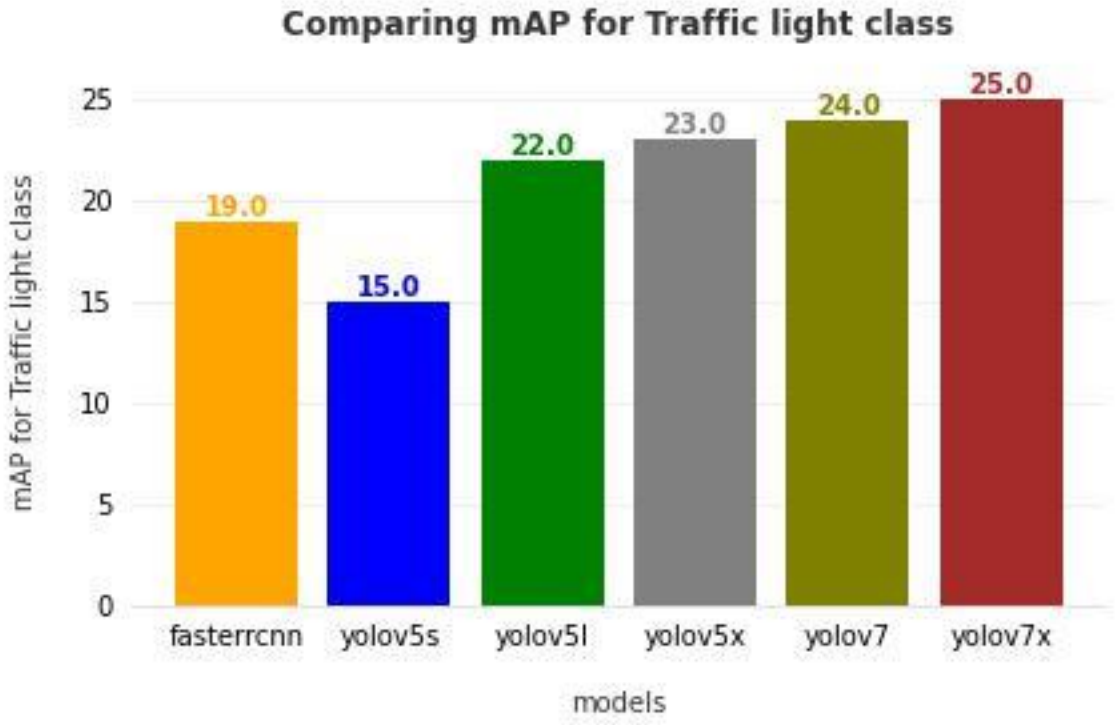
```
    YOLO format is basically (x, y, w, h), like COCO format but
normalized
    the annotation must be in txt file, each image has it's own
annotation file
    The data should be in a folder and under this folder we should
have two folders: images and labels
    in images we should have three different folders: train, test, val
    same thing for labels
    """
    if version == 'yolov5':
        """
        Dataset folder's structure for Yolov5 is as follows:
        ├─dataset
        │  ├─images
        │  │  ├─train
        │  │  ├─test
        │  │  ├─val
        │  ├─labels
        │  │  ├─train
        │  │  ├─test
        │  │  ├─val
        """
        # train data
        images_training_path = os.path.join(dataset_path, 'images',
'train')
        labels_training_path = os.path.join(dataset_path, 'labels',
'train')

        # val data
        images_val_path = os.path.join(dataset_path, 'images', 'val')
        labels_val_path = os.path.join(dataset_path, 'labels', 'val')

        # test data
        images_test_path = os.path.join(dataset_path, 'images',
'test')
        labels_test_path = os.path.join(dataset_path, 'labels',
'test')

    elif version == 'yolov7':
        """
        Dataset folder's structure for Yolov7 is as follows:
        ├─dataset
        │  ├─train
        │  │  ├─images
        │  │  ├─labels
        │  ├─test
        │  │  ├─images
        │  │  ├─labels
        │  ├─val
        │  │  ├─images
        │  │  ├─labels
        """
        # train data
        images_training_path = os.path.join(dataset_path, 'train',
'images')
        labels_training_path = os.path.join(dataset_path, 'train',
'labels')
```

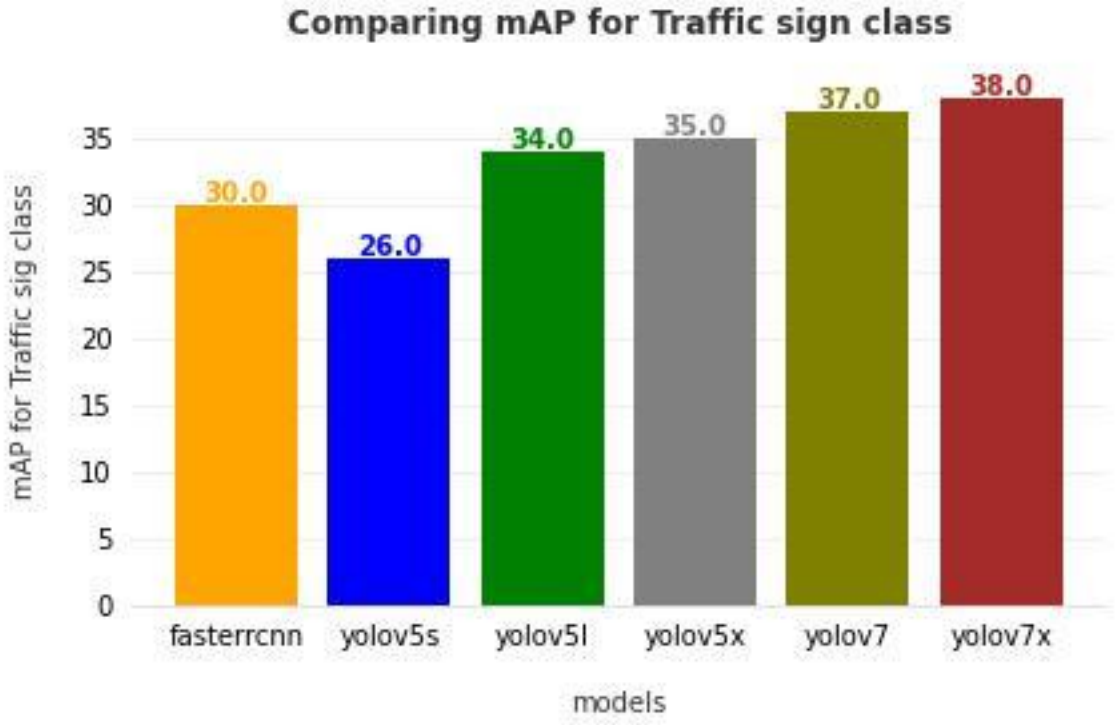
```
        # val data
        images_val_path = os.path.join(dataset_path, 'val', 'images')
        labels_val_path = os.path.join(dataset_path, 'val', 'labels')
        # test data
        images_test_path = os.path.join(dataset_path, 'test',
'images')
        labels_test_path = os.path.join(dataset_path, 'test',
'labels')

    print(50 * '#')
    # create annotation for training
    yolo_train = create_yolo_annotation(bdd_train)
    # move the data to the specific path
    move_files(yolo_train, images_training_path, labels_training_path)
    print(50 * '#')

    # create annotation for validation
    yolo_val = create_yolo_annotation(bdd_val)
    # move the data to the specific path
    move_files(yolo_val, images_val_path, labels_val_path)
    print(50 * '#')
    # create annotation for test
    yolo_test = create_yolo_annotation(bdd_test)
    # move the data to the specific path
    move_files(yolo_test, images_test_path, labels_test_path)
    print(50 * '#')
```

Appendix C


Comparing mAP for car class
mAP car for all classes
0
10
20
30
40
50
47.0
43.0
49.0
50.0
51.0
52.0
fasterrcnn
yolov5s
yolov5l
yolov5x
yolov7
yolov7x
models


Comparing mAP for pedestrian class
mAP for pedestrian class
0
5
10
15
20
25
30
29.0
21.0
30.0
31.0
33.0
33.0
fasterrcnn
yolov5s
yolov5l
yolov5x
yolov7
yolov7x
models


Comparing mAP for Traffic light class
mAP for Traffic light class
0
5
10
15
20
25
19.0
15.0
22.0
23.0
24.0
25.0
fasterrcnn
yolov5s
yolov5l
yolov5x
yolov7
yolov7x
models


Comparing mAP for Traffic sign class
mAP for Traffic sig class
0
5
10
15
20
25
30
35
30.0
26.0
34.0
35.0
37.0
38.0
fasterrcnn
yolov5s
yolov5l
yolov5x
yolov7
yolov7x
models

Code

Config.py

```
# https://github.com/rbgirshick/yacs
from yacs.config import CfgNode as CN

_C = CN()

# Dataset params
_C.DATASET = CN(new_allowed=True)
_C.DATASET.DATASET_NAME = 'Bdkk100K'
_C.DATASET.ROOT = 'dataset/bdd100k'
_C.DATASET.IMAGE_ROOT = 'images/100k/images/100k'
_C.DATASET.IMAGE_10K_ROOT = 'images/10k/bdd100k/images/10k'
_C.DATASET.LABEL_ROOT = 'labels/det_20'
_C.DATASET.SEMANTIC_SEGMENTATION_ROOT = ''
_C.DATASET.INSTANCE_SEGMENTATION_ROOT = 'labels/ins_seg/colormaps'
_C.DATASET.INSTANCE_SEGMENTATION_POLYGON_ROOT =
'labels/ins_seg/polygons'
_C.DATASET.PANOPTIC_SEGMENTATION = ''
_C.DATASET.DRIVABLE_AREA_MASK = 'labels/drivable/masks'
_C.DATASET.DRIVABLE_AREA_POLYGON_ROOT= 'labels/drivable/polygons'
_C.DATASET.LANE_ROOT = ''
_C.DATASET.TRAIN = 'train'
_C.DATASET.TEST = 'val'
_C.DATASET.IMAGE_FORMAT = 'jpg'

_C.DATASET.TASKS = ['detection', 'drivable area segmentation', 'Lane
Segmentation', 'semantic segmentation',
                    'instance segmentation', 'panoptic segmentation']

_C.DATASET.DETECTION_CLASSES = ['__bgr__', 'pedestrian', 'rider',
'car', 'truck', 'bus', 'train', 'motorcycle', 'bicycle',
                                'traffic light', 'traffic sign']
_C.DATASET.SEGMENTATION_CLASSES = ['road', 'sidewalk', 'building',
'wall', 'fence', 'pole', 'traffic light', 'traffic '

'sign',
                                   'vegetation', 'terrain', 'sky',
'person', 'rider', 'car', 'truck', 'bus', 'train',
                                   'motorcycle', 'bicycle']

_C.DATASET.INSTANCE_CLASSES = ['__bgr__', 'bicycle', 'person',
'caravan', 'car', 'bus', 'train', 'trailer', 'motorcycle','truck',
'rider']

_C.DATASET.PANOPTIC_CLASSES = ['unlabeled', 'dynamic', 'ego vehicle',
'ground', 'static', 'parking', 'rail track',
                               'road', 'sidewalk', 'bridge',
'building', 'fence', 'garage', 'guard rail', 'tunnel',
                               'wall', 'banner', 'billboard', 'lane
divider', 'parking sign', 'pole', 'polegroup',
                               'street light', 'traffic cone', 'traffic
device', 'traffic light', 'traffic sign',
                               'traffic sign frame', 'terrain',
```

```
'vegetation', 'sky', 'person', 'rider', 'bicycle',
                              'bus', 'car', 'caravan', 'motorcycle',
'trailer', 'train', 'truck']


# Detection params
_C.DETECTION = CN(new_allowed=True)
_C.DETECTION.MODELS = ['Faster_RCNN']
_C.DETECTION.BACKBONE = ['resnet50', 'MobileNetV3-Large']


# Drivable Area Segmentation params
_C.DRIVABLE_AREA = CN(new_allowed=True)
_C.DRIVABLE_AREA.MODELS = ['FCN', 'DeepLab']
_C.DRIVABLE_AREA.BACKBONE = ['resnet50', 'resnet101']
_C.DRIVABLE_AREA.DEEPLAB_BACKBONE = ['resnet50', 'resnet101',
'mobilenet']

# Instance Segmentation params
_C.INSTANCE_SEGMENTATION = CN(new_allowed=True)
_C.INSTANCE_SEGMENTATION.BACKBONE = ['resnet50']
_C.INSTANCE_SEGMENTATION.VERSION = ['v1', 'v2']

cfg = _C
```

Bdd.py

```
# Import Libraries
import os
import random
from pathlib import Path
from PIL import Image
import json
from collections import deque

from torch.utils import data
import torchvision.transforms as transforms


class BDD(data.Dataset):
    """
    Dataset class for the BDD100K dataset to be used in this project
    """

    def __init__(self,
                 cfg,
                 stage,
                 obj_cls,
                 db_path=None,
                 relative_path='..',
                 image_size=400,
                 transform=None,
                 seed=356,
                 ):
        """
        Constructor for BDD class
```

```
        :param cfg: yacs configuration that contains all the necessary
information about the dataset and labels
        :param stage: to select the stage (train, val, test)
        :param obj_cls: list contains the objects we want to detect
        :param db_path: db path for pre created db
        :param relative_path: relative dataset path
        :param image_size: image size when resizing the image
        :param transform: torchvision. Transforms as input
        """

        # Check the stage
        assert stage in ['train', 'val', 'test'], "stage must be :
'train', 'val', 'test'"

        # Load the root of all tasks
        self.root = Path(relative_path) / Path(cfg.DATASET.ROOT)  #
Parent root
        self.images_root = self.root / Path(cfg.DATASET.IMAGE_ROOT)  #
images root
        self.labels_root = self.root / Path(cfg.DATASET.LABEL_ROOT)  #
detection root
        self.drivable_root = self.root /
Path(cfg.DATASET.DRIVABLE_AREA_MASK)  # drivable area masks root
        self.semantic_segmentation_root = self.root /
Path(cfg.DATASET.SEMANTIC_SEGMENTATION_ROOT)  # sem seg masks root
        self.instance_segmentation_root = self.root /
Path(cfg.DATASET.INSTANCE_SEGMENTATION_ROOT)  # ins seg masks root
        self.panoptic_root = self.root /
Path(cfg.DATASET.PANOPTIC_SEGMENTATION)  # panoptic seg masks root
        self.lane_root = self.root / Path(cfg.DATASET.LANE_ROOT)  #
lane masks root

        self.stage = stage
        self.obj_cls = obj_cls
        self.image_size = image_size
        self.transform = transform

        # load images
        self.images = list(self.images_root.glob('**/*.jpg'))

        self.db = deque()  # deque object to hold the info

        # class to index and index to class mapping
        self.cls_to_idx, self.idx_to_cls = self.create_idx()

        random.seed(seed)  # Fixed seeds

    # method to create two different dictionaries for mapping
    def create_idx(self):
        cls_to_idx = {}
        idx_to_cls = {}
        idx = 0

        for obj in self.obj_cls:
            cls_to_idx[self.obj_cls[idx]] = idx
            idx_to_cls[idx] = self.obj_cls[idx]
            idx += 1
```

```
            # if obj is a traffic light, add the class with the color
except the NA
            """
            if obj == 'traffic light':

                cls_to_idx['tl_G'] = idx
                idx_to_cls[idx] = 'tl_G'
                idx += 1

                cls_to_idx['tl_R'] = idx
                idx_to_cls[idx] = 'tl_R'
                idx += 1

                cls_to_idx['tl_Y'] = idx
                idx_to_cls[idx] = 'tl_Y'
                idx += 1

            else:
                cls_to_idx[self.obj_cls[idx]] = idx
                idx_to_cls[idx] = self.obj_cls[idx]
                idx += 1
            """


        return cls_to_idx, idx_to_cls

    # method to split the data into train and val based on percentage
    def split_data(self, db, train_size=80):
        db = list(db)
        to_idx = (train_size * len(db)) // 100
        if self.stage == 'train':
            train_db = db[:to_idx]
            return deque(train_db)
        elif self.stage == 'val':
            val_db = db[to_idx:]
            return deque(val_db)

        else:
            return deque(db)

    @staticmethod
    def xyxy_to_xywh(x1, y1, x2, y2):
        """
        static method to convert x,y,x,y format to x,y,w,h format
        :param x1: x1 position
        :param y1: y1 position
        :param x2: x2 position
        :param y2: y2 position
        :return: x, y, w, h
        """
        x = x1
        y = y1
        w = x2 - x1
        h = y2 - y2
        return (x, y, w, h)

    @staticmethod
```

```
    def xywh_to_xyxy(x, y, w, h):
        """
        static method to convert x,y,w,h format to x,y,x,y format
        :param x: x position
        :param y: y position
        :param w: width
        :param h: height
        :return: x1, y1, x2, y2
        """
        x1 = x
        y1 = y
        x2 = w - x1
        y2 = h - y1
        return (x1, y1, x2, y2)

    def image_transform(self, img):
        """
        image transform if the given one is None
        :param img: PIL image
        :return: image tensor with applied transform on it
        """
        if self.transform is None:
            t_ = transforms.Compose([
                transforms.ToTensor(),  # convert the image to tensor
                transforms.Normalize(mean=[0.407, 0.457, 0.485],
                                     std=[0.229, 0.224, 0.225])  #
normalize the image using mean ans std
            ])
            return t_(img)
        else:
            return self.transform(img)

    def get_image(self, idx, apply_transform=False):
        """
        method the get image from the list of images
        :param idx: index of the image in the list of images
        :param apply_transform: Boolean value to check if we want to
apply transform or no
        :return: PIL image or Tensor
        """
        image = Image.open(self.db[idx]['image_path'])
        if apply_transform:
            image = self.image_transform(image)

        return image

    def export_db(self, path):
        """
        method to export the database
        :param path: path to where we want to export the database
        :return: None
        """
        print(f"Exporting {self.stage}_db DB...")
        with open(os.path.join(path, f'{self.stage}_db.json'), "w") as
outfile:
            json.dump(list(self.db), outfile)
        print(f"DB {self.stage}_db Exported.")
```

```
    # method to create the database, task specific, to be implemented
in the children classes
    def __create_db(self):
        raise NotImplementedError

    # method to display the image, task specific, to be implemented in
the children classes
    def display_image(self, idx):
        raise NotImplementedError

    def __len__(self):
        """
        method to return the length of the database
        :return: length of the db
        """
        return len(self.db)

    # method to get img and target from the db, task specific, to be
implemented in the children classes
    def __getitem__(self, idx):
        raise NotImplementedError
```

bdd_detection.py

```
# Import Libraries
import random
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
import albumentations as A
import cv2
import json
from tqdm import tqdm
from collections import deque

import torch

from .bdd import BDD

# Define color map to be used when displaying the images with bounding
boxes
COLOR_MAP = ['blue', 'orange', 'green', 'red', 'purple', 'brown',
'pink', 'gray', 'olive', 'cyan', 'blue']

"""
CLS_TO_IDX = {
    '__bgr__': 0,
    'pedestrian': 1,
    'car': 2,
    'bus': 2,
    'truck': 2,
    'traffic light': 3,
    'traffic sign': 4,
```

```
    'bicycle': 5,
    'motorcycle': 5,
}

IDX_TO_CLS = {
    0: '__bgr__',
    1: 'pedestrian',
    2: 'car',
    3: 'traffic light',
    4: 'traffic sign',
    5: 'motorcycle'
}
"""



class BDDDetection(BDD):
    """
    BDDDetection class, specific class for the detection task on
BDD100K dataset
    """

    def __init__(self,
                 cfg,
                 stage,
                 obj_cls=['__bgr__', 'pedestrian', 'car', 'traffic
light', 'traffic sign'],
                 db_path=None,
                 relative_path='..',
                 image_size=400,
                 transform=None):
        """
        Constructor for BDDDetection class
        :param cfg: yacs configuration that contains all the necessary
information about the dataset and labels
        :param stage: to select the stage (train, val, test)
        :param obj_cls: list contains the objects we want to detect
        :param db_path:  db path for pre created db
        :param relative_path: relative dataset path
        :param image_size:  tuple that contains the image size (w, h)
        :param transform: torchvision. Transforms as input
        """
        super(BDDDetection, self).__init__(cfg, stage, obj_cls,
db_path, relative_path, image_size, transform)

        # check if the classes are in the DETECTION_CLASSES
        assert all(cls in cfg.DATASET.DETECTION_CLASSES for cls in
                   obj_cls), f"Please choose classes from the
following: {cfg.DATASET.DETECTION_CLASSES}"

        # load pre created db
        if db_path:
            with open(db_path, 'r') as f:
                self.db = json.load(f)
        else:
            # or create it and split the data
            self.db = self.__create_db()
```

```
            # self.db = self.split_data(_db)


    def split_data(self, labels, train_size=80):
        to_idx = (train_size * len(labels)) // 100
        if self.stage == 'train':
            train_db = labels[:to_idx]
            return train_db
        elif self.stage == 'val':
            val_db = labels[to_idx:]
            return val_db
        else:
            return labels

    def __create_db(self):
        """
        private method to create the database for the class
        :return: deque object that holds the database
        """
        detection_db = deque()
        # labels_path = self.labels_root / Path('det_train.json' if
self.stage == 'train' else 'det_val.json')

        # load the labels from the json file
        labels_path = self.labels_root / Path('det_val.json' if
self.stage == 'test' else 'det_train.json')
        with open(labels_path, 'r') as labels_file:
            labels = json.load(labels_file)

        random.shuffle(labels)

        # labels = self.__filter_data(labels)

        if self.stage == 'test':
            labels = random.sample(labels, 2500)
        else:
            labels = random.sample(labels, 40000)

        labels = self.split_data(labels)

        # loop through the labels
        for item in tqdm(labels):
            # image_path = str(self.images_root / Path('train' if
self.stage == 'train' else 'test') / Path(item['name']))
            image_path = str(self.images_root / Path('val' if
self.stage == 'test' else 'train') / Path(item['name']))

            classes = []  # list of classes in one image
            bboxes = []  # list of bboxes in one image

            # if the annotation has 'labels' key in the dic
            if 'labels' in item.keys():
                objects = item['labels']  # hold the objects
                # print(objects)

                # loop through object in objects
                for obj in objects:
```

```
                    category = obj['category']  # hold the category of
the object

                    # if the category is in the classes we want to
predict
                    if category in self.obj_cls:
                        x1 = obj['box2d']['x1']
                        y1 = obj['box2d']['y1']
                        x2 = obj['box2d']['x2']
                        y2 = obj['box2d']['y2']

                        # bbox = [x1, y1, x2 - x1, y2 - y1]  # bbox of
form: (x, y, w, h) MSCOCO format
                        bbox = [x1, y1, x2, y2]



                        cls = self.cls_to_idx[category]

                        bboxes.append(bbox)
                        classes.append(cls)

                        """
                        # if the category is traffic light load it with
the color
                        if category == 'traffic light':
                            # print(obj['attributes'])
                            color =
obj['attributes']['trafficLightColor']
                            if color != 'NA':
                                # print(color)
                                category = "tl_" + color

                                x1 = obj['box2d']['x1']
                                y1 = obj['box2d']['y1']
                                x2 = obj['box2d']['x2']
                                y2 = obj['box2d']['y2']

                                # bbox = [x1, y1, x2 - x1, y2 - y1]  #
bbox of form: (x, y, w, h) MSCOCO format
                                if format == 'xyxy':
                                    bbox = [x1, y1, x2, y2]

                                cls = self.cls_to_idx[category]

                                bboxes.append(bbox)
                                classes.append(cls)

                        else:
                            x1 = obj['box2d']['x1']
                            y1 = obj['box2d']['y1']
                            x2 = obj['box2d']['x2']
                            y2 = obj['box2d']['y2']

                            # bbox = [x1, y1, x2 - x1, y2 - y1]  # bbox
of form: (x, y, w, h) MSCOCO format
```

```
                            if format == 'xyxy':
                                bbox = [x1, y1, x2, y2]

                            cls = self.cls_to_idx[category]

                            bboxes.append(bbox)
                            classes.append(cls)
                        """
                # if we have one and more objects in the image append
the image path, bboxes and classes to the db
                if len(classes) > 0:
                    detection_db.append({
                        'image_path': image_path,
                        'bboxes': bboxes,
                        'classes': classes
                    })
        # finally, return the db
        return detection_db

    # method to apply data augmentation
    def data_augmentation(self, image, bboxes, labels):
        """
        method to apply image augmentation technics to reduce
overfitting
        :param image: numpy array with shape of HxWx3 (RGB image)
        :param bboxes: list of bounding boxes, each box must have
(xmin, ymin, xmax, ymax)
        :param labels: idx of the labels
        :return: image, masks, bboxes
        """

        class_labels = [self.idx_to_cls[label] for label in labels]
        for idx, box in enumerate(bboxes):
            box.append(class_labels[idx])

        augmentation_transform = A.Compose([
            A.HorizontalFlip(p=0.5),  # Random Flip with 0.5
probability
            A.CropAndPad(px=100, p=0.5),  # crop and add padding with
0.5 probability
            A.PixelDropout(dropout_prob=0.01, p=0.5),  # pixel dropout
with 0.5 probability
        ], bbox_params=A.BboxParams(format='pascal_voc',
min_visibility=0.3))  # return bbox with xyxy format

        transformed = augmentation_transform(image=image,
bboxes=bboxes)

        transformed_boxes = []
        transformed_labels = []
        for box in transformed['bboxes']:
            box = list(box)
            label = box.pop()
            transformed_boxes.append(box)
            transformed_labels.append(label)

        labels = [self.cls_to_idx[label] for label in
```

```
transformed_labels]

        return transformed['image'], transformed_boxes, labels

    # method to display the image with the bounding boxes
    def display_image(self, idx, display_labels=True):
        image = self.get_image(idx, apply_transform=False)

        classes = self.db[idx]['classes']
        bboxes = self.db[idx]['bboxes']

        # plot the image
        fig, ax = plt.subplots()
        ax.imshow(image)
        for i in range(len(classes)):
            # print(classes[i], color_map[classes[i]])
            bbox = bboxes[i]
            # for bbox in bboxes:
            # to load to correspond color map
            rect = patches.Rectangle((bbox[0], bbox[1]), bbox[2] -
bbox[0], bbox[3] - bbox[1],
                                     edgecolor=COLOR_MAP[classes[i]],
                                     facecolor="none", linewidth=2)
            ax.add_patch(rect)
            if display_labels:
                plt.text(bbox[0], bbox[1], self.idx_to_cls[classes[i]],
verticalalignment="top", color="white",
                         bbox={'facecolor': COLOR_MAP[classes[i]],
'pad': 0})

        plt.axis('off')
        plt.show()

    # collate function to be used with the dataloader, since the not
all the images has the same number of objects
    def collate_fn(self, batch):
        return tuple(zip(*batch))

    def __getitem__(self, idx):
        """
        method to return the item based on the index
        :param idx: index of the image in db
        :return: img and targets
        """
        image = self.get_image(idx, apply_transform=False)
        labels = self.db[idx]['classes']
        boxes = self.db[idx]['bboxes']
        if self.stage == 'train':
            image, boxes, labels =
self.data_augmentation(np.array(image), boxes, labels)
        image = self.image_transform(image)
        targets = {
            'labels': torch.tensor(labels, dtype=torch.int64),
            'boxes': torch.tensor(boxes, dtype=torch.float)
        }

        return image, targets
```

Bdd_drivable_segmentation.py

```
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import albumentations as A
from albumentations.pytorch import ToTensorV2


import torch
from torch.utils import data
import torchvision.transforms as transforms

from .bdd import BDD

COLOR_MAP = ['blue', 'red']


class BDDDrivableSegmentation(BDD):
    """
    BDDDrivableSegmentation class, specific class for the drivable area
segmentation task on BDD100K dataset
    """
    def __init__(self,
                 cfg,
                 stage,
                 obj_cls=['direct', 'alternative', 'background'],
                 relative_path='..',
                 image_size=400,
                 transform=None):
        """
        Constructor for BDDDrivableSegmentation class
        :param cfg: yacs configuration that contains all the necessary
information about the dataset and labels
        :param stage: to select the stage (train, val, test)
        :param obj_cls: list contains the objects we want to detect
        :param relative_path: relative dataset path
        :param image_size:  tuple that contains the image size (w, h)
        :param transform: torchvision. Transforms as input
        """
        super(BDDDrivableSegmentation, self).__init__(cfg=cfg,
                                                      stage=stage,
                                                      obj_cls=obj_cls,

relative_path=relative_path,

image_size=image_size,

transform=transform)

        self.cls_to_idx, self.idx_to_cls = self.create_idx()

        _db = self.__create_db()
        self.db = self.split_data(_db)

    def data_augmentation(self):
```

```
        if self.stage == 'train':
            compose = A.Compose([
                A.Resize(height=self.image_size,
width=self.image_size),
                A.Rotate(limit=35, p=0.5),
                A.HorizontalFlip(p=0.5),
                A.VerticalFlip(p=0.1),
                A.Normalize(
                    mean=[0.0, 0.0, 0.0],
                    std=[1.0, 1.0, 1.0]
                ),
                ToTensorV2()
            ])

        else:
            compose = A.Compose([
                A.Resize(height=self.image_size,
width=self.image_size),
                A.Normalize(
                    mean=[0.407, 0.457, 0.485],
                    sstd=[0.229, 0.224, 0.225]
                ),
                ToTensorV2()
            ])

        return compose

    def __create_db(self):
        """
        method to create the db of the class
        :return: list ot Pathlib objects of the masks
        """
        masks_path = self.drivable_root / Path('val' if self.stage ==
'test' else 'train')
        return list(masks_path.glob('**/*.png'))

    def _get_mask(self, idx):
        """
        method to get the mask of the image
        :param idx: index of the mask in the db
        :return: np array
        """
        mask = np.array(Image.open(str(self.db[idx])))
        # mask = torch.tensor(mask, dtype=torch.uint8)
        return mask

    def get_image(self, idx, apply_transform=False):
        """
        method to return the image
        :param idx: index of the mask in the db
        :param apply_transform: Boolean value, if we want to apply the
transform or not
        :return: PIL image or Tensor type
        """
        image_name = str(self.db[idx]).split('/')[-1].replace('.png',
'.jpg')
        image_path = str(self.images_root / Path('val' if self.stage ==
```

```
'test' else 'train') / image_name)
        image = Image.open(image_path)
        if apply_transform:
            image = self.image_transform(image)

        return image

    def display_image(self, idx, mask=None, alpha=0.5):
        """
        method to display the image with the mask
        :param idx: index of the mask in the db
        :param mask: mask of the image
        :param alpha: degree of transparency
        :return: None
        """
        image = np.array(self.get_image(idx, False))
        plt.imshow(image)
        if mask is not None:
            plt.imshow(mask, alpha=alpha)
            plt.title('Image with Mask')

        plt.axis('off')
        plt.show()

    def __getitem__(self, idx):
        """
        method to return the item based on the index
        :param idx: index of the image in db
        :return: img and mask
        """

        label = dict()
        img = self.get_image(idx, apply_transform=False)
        mask = self._get_mask(idx)
        augmentation = self.transform(image=img, mask=mask)
        image = augmentation['image']
        mask = augmentation['mask']
        # mask = torch.tensor(self._get_mask(idx), dtype=torch.long)
        mask = mask.long()
        return img, mask
```

bdd_instance_segmentation.py

```
# Import Libraries
import os
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import albumentations as A
import cv2
from PIL import Image
import json
from collections import deque
from tqdm import tqdm
```

```
from .bdd_utils import to_mask, bbox_from_instance_mask,
get_coloured_mask

import torch
import torchvision.transforms as transforms

from .bdd import BDD

# Define color map to be used when displaying the images with bounding
boxes
COLOR_MAP = ['blue', 'orange', 'green', 'red', 'purple', 'brown',
'pink', 'gray', 'olive', 'cyan', 'blue']

class BDDInstanceSegmentation(BDD):
    """
    BDD class for Instance Segmentation task
    """

    def __init__(self,
                 cfg,
                 stage,
                 obj_cls=['__bgr__', 'person', 'car', 'rider',
'bicycle', 'motorcycle', 'truck', 'bus'],
                 relative_path='..',
                 image_size=640,
                 transform=None):
        """
        Constructor for BDDInstanceSegmentation class
        :param cfg: yacs configuration that contains all the necessary
information about the dataset and labels
        :param stage: to select the stage (train, val, test)
        :param obj_cls: list contains the objects we want to detect
        :param relative_path: relative dataset path
        :param image_size:  tuple that contains the image size (w, h)
        :param transform: torchvision. Transforms as input
        """
        super(BDDInstanceSegmentation, self).__init__(cfg=cfg,
                                                      stage=stage,
                                                      obj_cls=obj_cls,

relative_path=relative_path,

image_size=image_size,

transform=transform)

        # check if the classes are in the DETECTION_CLASSES
        assert all(cls in cfg.DATASET.INSTANCE_CLASSES for cls in
                   obj_cls), f"Please choose classes from the
following: {cfg.DATASET.INSTANCE_CLASSES}"

        self.cls_to_idx, self.idx_to_cls = self.create_idx()

        if self.stage == 'train' or self.stage == 'val':
            self.images_root = self.root /
Path(cfg.DATASET.IMAGE_10K_ROOT + '/train')  # images root
            self.instance_segmentation_root = self.root / Path(
```

```
                cfg.DATASET.INSTANCE_SEGMENTATION_ROOT + '/train')  #
ins seg masks root
            self.polygon_root = self.root / Path(
                cfg.DATASET.INSTANCE_SEGMENTATION_POLYGON_ROOT +
'/ins_seg_train.json')  # polygon root
        elif self.stage == 'test':
            self.images_root = self.root /
Path(cfg.DATASET.IMAGE_10K_ROOT + '/val')  # images root
            self.instance_segmentation_root = self.root / Path(
                cfg.DATASET.INSTANCE_SEGMENTATION_ROOT + '/val')  # ins
seg masks root
            self.polygon_root = self.root / Path(
                cfg.DATASET.INSTANCE_SEGMENTATION_POLYGON_ROOT +
'/ins_seg_val.json')  # polygon root

        _db = self.__create_db()
        self.db = self.split_data(_db)

    def data_augmentation(self, image, masks, bboxes, labels):
        """
        method to apply image augmentation technics to reduce
overfitting
        :param image: numpy array with shape of HxWx3 (RGB image)
        :param masks: list of masks, each mask must have the same W and
H with the image (2D mask)
        :param bboxes: list of bounding boxes, each box must have
(xmin, ymin, xmax, ymax)
        :param labels: idx of the labels
        :return: image, masks, bboxes
        """
        class_labels = [self.idx_to_cls[label] for label in labels]
        for idx, box in enumerate(bboxes):
            box.append(class_labels[idx])

        augmentation_transform = A.Compose([
            A.HorizontalFlip(p=0.5),  # Random Flip with 0.5
probability
            A.CropAndPad(px=100, p=0.5),  # crop and add padding with
0.5 probability
            A.PixelDropout(dropout_prob=0.01, p=0.5),  # pixel dropout
with 0.5 probability
        ], bbox_params=A.BboxParams(format='pascal_voc',
min_visibility=0.3))  # return bbox with xyxy format

        transformed = augmentation_transform(image=image, masks=masks,
bboxes=bboxes)

        transformed_boxes = []
        transformed_labels = []
        for box in transformed['bboxes']:
            box = list(box)
            label = box.pop()
            transformed_boxes.append(box)
            transformed_labels.append(label)

        labels = [self.cls_to_idx[label] for label in
transformed_labels]
```

```
        return transformed['image'], transformed['masks'],
transformed_boxes, labels

    def __create_db(self):
        """
        method to create the db of the class
        :return: deque object contains the necessary information
        """
        polygon_annotation = deque(self.__load_annotations())
        db = deque()
        for polygon in tqdm(polygon_annotation):
            filtered_labels = self.__filter_labels(polygon['labels'])

            if len(filtered_labels):
                db.append({
                    'image_path': self.images_root /
Path(polygon['name']),
                    'mask_path': self.instance_segmentation_root /
Path(polygon['name'].replace('.jpg', '.png')),
                    'labels': filtered_labels
                })

        return db

    def __load_annotations(self):
        """
        method to load the annotation from json
        :return: list of annotations
        """
        with open(self.polygon_root, 'r') as f:
            polygon_annotation = json.load(f)

        return polygon_annotation

    def __filter_labels(self, labels):
        """
        method to filter the labels according to the objects passed to
the constructor
        :param labels: list of dictionaries for the objects in the
image
        :return: list of filtered labels
        """
        filtered_labels = []
        for label in labels:
            if label['category'] in self.obj_cls:
                filtered_labels.append(label)

        return filtered_labels

    def image_transform(self, img):
        """
        image transform if the given one is None
        :param img: PIL image
        :return: image tensor with applied transform on it
        """
        if self.transform is None:
```

```
            t_ = transforms.Compose([
                transforms.ToTensor(),  # convert the image to tensor
                transforms.Normalize(mean=[0.407, 0.457, 0.485],
                                     std=[0.229, 0.224, 0.225])  #
normalize the image using mean ans std
            ])
            return t_(img)
        else:
            return self.transform(img)

    def get_image(self, idx, apply_transform=False):
        """
        method to return the image
        :param idx: index of the mask in the db
        :param apply_transform: Boolean value, if we want to apply the
transform or not
        :return: PIL image or Tensor type
        """
        image_path = self.db[idx]['image_path']
        image = Image.open(image_path).convert('RGB')
        if apply_transform:
            image = self.image_transform(image)

        return image

    def get_mask(self, idx):
        """
        method to get the mask of the image
        :param idx: index of the mask in the db
        :return: np array
        """
        mask_path = self.db[idx]['mask_path']
        mask = np.array(Image.open(mask_path))
        # mask = torch.tensor(mask, dtype=torch.uint8)
        return mask

    def _get_labels(self, idx):
        image_annotation = self.db[idx]
        mask_shape =
np.array(Image.open(image_annotation['mask_path'])).shape
        target = {}
        boxes = []
        masks = []
        labels = []
        for label in image_annotation['labels']:
            poly2d = label['poly2d'][0]['vertices']
            mask = to_mask(mask_shape, poly2d)
            box = bbox_from_instance_mask(mask)
            label = self.cls_to_idx[label['category']]
            if box is not None:
                if box[0] != box[2] and box[1] != box[3]:
                    masks.append(np.array(mask, dtype=np.uint8))
                    boxes.append(box)
                    labels.append(label)

        target['boxes'] = boxes
        target['labels'] = labels
```

```
        target['masks'] = masks

        return target

    # method to display the image, task specific, to be implemented in
the children classes
    def display_image(self, image, masks, boxes, labels):
        if isinstance(image, Image.Image):
            image = np.array(image)
        for mask in masks:
            rgb_mask = get_coloured_mask(mask)
            image = cv2.addWeighted(image, 1, rgb_mask, 0.5, 0)

        fig, ax = plt.subplots(figsize=(20,20))
        ax.imshow(image)
        for i, mask in enumerate(labels):
            bbox = boxes[i]
            rect = patches.Rectangle((bbox[0], bbox[1]), bbox[2] -
bbox[0], bbox[3] - bbox[1],
                                     edgecolor=COLOR_MAP[labels[i]],
                                     facecolor="none", linewidth=2)
            plt.text(bbox[0], bbox[1], self.idx_to_cls[labels[i]],
verticalalignment="top",
                     color=COLOR_MAP[labels[i]])

            ax.add_patch(rect)

        plt.axis('off')
        plt.show()


    # collate function to be used with the dataloader, since the not
all the images has the same number of objects
    def collate_fn(self, batch):
        return tuple(zip(*batch))

    def __len__(self):
        return len(self.db)

    def __getitem__(self, idx):
        image = self.get_image(idx, False)
        target = self._get_labels(idx)

        image, masks, bboxes, labels =
self.data_augmentation(np.array(image), target['masks'],
target['boxes'], target['labels'])

        target['boxes'] = torch.tensor(bboxes, dtype=torch.float32)

        target['labels'] = torch.tensor(labels, dtype=torch.int64)

        target['masks'] = torch.tensor(np.array(masks, dtype=np.uint8))

        image_id = torch.tensor([idx])
        area = (target['boxes'][:, 3] - target['boxes'][:, 1]) *
(target['boxes'][:, 2] - target['boxes'][:, 0])
        # suppose all instances are not crowd
```

```
        iscrowd = torch.zeros((len(self.cls_to_idx),),
dtype=torch.int64)

        target['image_id'] = image_id
        target['area'] = area
        target['iscrowd'] = iscrowd

        image = self.image_transform(image)

        return image, target
```

Bdd_instance_segmnetation_drivable.py

```
# Import Libraries
import os
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import albumentations as A
import cv2
from PIL import Image
import json
from collections import deque
from tqdm import tqdm
from sklearn.model_selection import train_test_split

from .bdd_utils import to_mask, bbox_from_instance_mask,
get_coloured_mask

import torch
import torchvision.transforms as transforms

from .bdd import BDD

# Define color map to be used when displaying the images with bounding
boxes
COLOR_MAP = ['blue', 'orange', 'green', 'red', 'purple', 'brown',
'pink', 'gray', 'olive', 'cyan', 'blue']


class BDDInstanceSegmentationDrivable(BDD):
    """
    BDD class for Instance Segmentation task
    """

    def __init__(self,
                 cfg,
                 stage,
                 obj_cls=['__bgr__', 'person', 'car', 'rider',
'bicycle', 'motorcycle', 'truck', 'bus'],
                 relative_path='..',
                 image_size=640,
                 transform=None):
        """
        Constructor for BDDInstanceSegmentation class
```

```
        :param cfg: yacs configuration that contains all the necessary
information about the dataset and labels
        :param stage: to select the stage (train, val, test)
        :param obj_cls: list contains the objects we want to detect
        :param relative_path: relative dataset path
        :param image_size:  tuple that contains the image size (w, h)
        :param transform: torchvision. Transforms as input
        """
        super(BDDInstanceSegmentationDrivable, self).__init__(cfg=cfg,
                                                              stage=stage,
                                                              obj_cls=obj_cls,

relative_path=relative_path,

image_size=image_size,

transform=transform)

        self.cls_to_idx, self.idx_to_cls = self.create_idx()
        self.image_root = os.path.join(self.root,
cfg.DATASET.IMAGE_10K_ROOT, 'train')
        self.images = os.listdir(self.image_root)
        self.instance_masks_path = os.path.join(self.root,
cfg.DATASET.INSTANCE_SEGMENTATION_ROOT, 'train')
        self.instance_masks = os.listdir(self.instance_masks_path)
        self.polygon_root = os.path.join(self.root,
cfg.DATASET.INSTANCE_SEGMENTATION_POLYGON_ROOT,
                                         'ins_seg_train.json')

        self.polygon_drviable_root = os.path.join(self.root,
cfg.DATASET.DRIVABLE_AREA_POLYGON_ROOT, 'drivable_train.json')
        self.drivable_masks_path = os.path.join(self.root,
cfg.DATASET.DRIVABLE_AREA_MASK, 'train')
        self.drivable_masks = os.listdir(self.drivable_masks_path)

        self.available_images = self.intersection()
        _db = self.__create_db()
        self.db = self.split_data(_db)

    def intersection(self):
        masks = [mask.replace('.png', '.jpg') for mask in
self.drivable_masks]
        lst3 = [value for value in self.images if value in masks]
        return lst3

    # method to split the data into train and val based on percentage
    def split_data(self, db, train_size=80):
        db = list(db)

        train_db, test_db = train_test_split(db, test_size=1 -
(train_size / 100), random_state=42)

        val_db, test_db = train_test_split(test_db, test_size=0.5,
random_state=42)

        if self.stage == 'train':
            return deque(train_db)
```

```
        elif self.stage == 'val':
            return deque(val_db)
        elif self.stage == 'test':
            return deque(test_db)

    def data_augmentation(self, image, masks, bboxes, labels):
        """
        method to apply image augmentation technics to reduce
overfitting
        :param image: numpy array with shape of HxWx3 (RGB image)
        :param masks: list of masks, each mask must have the same W and
H with the image (2D mask)
        :param bboxes: list of bounding boxes, each box must have
(xmin, ymin, xmax, ymax)
        :param labels: idx of the labels
        :return: image, masks, bboxes
        """
        class_labels = [self.idx_to_cls[label] for label in labels]
        for idx, box in enumerate(bboxes):
            box.append(class_labels[idx])

        augmentation_transform = A.Compose([
            A.HorizontalFlip(p=0.5),  # Random Flip with 0.5
probability
            A.CropAndPad(px=100, p=0.5),  # crop and add padding with
0.5 probability
            A.PixelDropout(dropout_prob=0.01, p=0.5),  # pixel dropout
with 0.5 probability
        ], bbox_params=A.BboxParams(format='pascal_voc',
min_visibility=0.3))  # return bbox with xyxy format

        transformed = augmentation_transform(image=image, masks=masks,
bboxes=bboxes)

        transformed_boxes = []
        transformed_labels = []
        for box in transformed['bboxes']:
            box = list(box)
            label = box.pop()
            transformed_boxes.append(box)
            transformed_labels.append(label)

        labels = [self.cls_to_idx[label] for label in
transformed_labels]

        return transformed['image'], transformed['masks'],
transformed_boxes, labels

    def __create_db(self):
        """
        method to create the db of the class
        :return: deque object contains the necessary information
        """
        masks_polygon, drivable_polygon = self.__load_annotations()
        db = deque()
        for polygon_key in tqdm(masks_polygon):
            polygon = masks_polygon[polygon_key]
```

```
            if polygon['name'] in self.available_images:
                filtered_labels =
self.__filter_labels(polygon['labels'])
                if 'labels' in
drivable_polygon[polygon['name']].keys():
                    drivable_labels =
drivable_polygon[polygon_key]['labels']
                    for drivable_label in drivable_labels:
                        drivable_label['category'] = 'road'
                        filtered_labels.append(drivable_label)

                if len(filtered_labels):
                    db.append({
                        'image_path': os.path.join(self.image_root,
polygon['name']),
                        'mask_path':
os.path.join(self.instance_masks_path, polygon['name'].replace('.jpg',
'.png')),
                        'drivable_path':
os.path.join(self.drivable_masks_path, polygon['name'].replace('.jpg',
'.png')),
                        'labels': filtered_labels
                    })
        return db

    def __load_annotations(self):
        """
        method to load the annotation from json
        :return: list of annotations
        """
        with open(self.polygon_root, 'r') as f:
            polygon_annotation = json.load(f)

        with open(self.polygon_drviable_root, 'r') as f:
            polygon_drivable_annotation = json.load(f)

        masks_annotations = dict()
        drivable_annotations = dict()

        for polygon in polygon_annotation:
            masks_annotations[polygon['name']] = polygon

        for drivable_polygon in polygon_drivable_annotation:
            drivable_polygon
            drivable_annotations[drivable_polygon['name']] =
drivable_polygon

        return masks_annotations, drivable_annotations

    def __filter_labels(self, labels):
        """
        method to filter the labels according to the objects passed to
the constructor
        :param labels: list of dictionaries for the objects in the
image
        :return: list of filtered labels
        """
```

```
        filtered_labels = []
        for label in labels:
            if label['category'] in self.obj_cls:
                filtered_labels.append(label)

        return filtered_labels

    def image_transform(self, img):
        """
        image transform if the given one is None
        :param img: PIL image
        :return: image tensor with applied transform on it
        """
        if self.transform is None:
            t_ = transforms.Compose([
                transforms.ToTensor(),  # convert the image to tensor
                transforms.Normalize(mean=[0.407, 0.457, 0.485],
                                     std=[0.229, 0.224, 0.225])  #
normalize the image using mean ans std
            ])
            return t_(img)
        else:
            return self.transform(img)

    def get_image(self, idx, apply_transform=False):
        """
        method to return the image
        :param idx: index of the mask in the db
        :param apply_transform: Boolean value, if we want to apply the
transform or not
        :return: PIL image or Tensor type
        """
        image_path = self.db[idx]['image_path']
        image = Image.open(image_path).convert('RGB')
        if apply_transform:
            image = self.image_transform(image)

        return image

    def get_drivable_mask(self, idx):
        """
        method to get the mask for the drivable area
        :param idx:
        :return:
        """
        drivable_mask_path = self.db[idx]['drivable_path']
        mask = np.array(Image.open(drivable_mask_path))
        # mask = torch.tensor(mask, dtype=torch.uint8)
        return mask

    def get_mask(self, idx):
        """
        method to get the mask of the image
        :param idx: index of the mask in the db
        :return: np array
        """
        mask_path = self.db[idx]['mask_path']
```

```
        mask = np.array(Image.open(mask_path))
        # mask = torch.tensor(mask, dtype=torch.uint8)
        return mask

    def _get_labels(self, idx):
        image_annotation = self.db[idx]
        mask_shape =
np.array(Image.open(image_annotation['mask_path'])).shape
        target = {}
        boxes = []
        masks = []
        labels = []
        for label in image_annotation['labels']:
            poly2d = label['poly2d'][0]['vertices']
            mask = to_mask(mask_shape, poly2d)
            box = bbox_from_instance_mask(mask)
            label = self.cls_to_idx[label['category']]
            if box is not None:
                if box[0] != box[2] and box[1] != box[3]:
                    masks.append(np.array(mask, dtype=np.uint8))
                    boxes.append(box)
                    labels.append(label)


        target['boxes'] = boxes
        target['labels'] = labels
        target['masks'] = masks

        return target

    # method to display the image, task specific, to be implemented in
the children classes
    def display_image(self, image, masks, boxes, labels):
        if isinstance(image, Image.Image):
            image = np.array(image)
        for idx, mask in enumerate(masks):
            rgb_mask, color = get_coloured_mask(mask)
            image = cv2.addWeighted(image, 1, rgb_mask, 0.5, 0)
            bbox = [(int(boxes[idx][0]), int(boxes[idx][1])),
(int(boxes[idx][2]), int(boxes[idx][3]))]
            cv2.rectangle(image, bbox[0], bbox[1], color=color,
thickness=2)
            cv2.putText(image, self.idx_to_cls[labels[idx]], bbox[0],
cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0),
                        thickness=2)

        plt.imshow(image)
        plt.axis('off')
        plt.show()

    # collate function to be used with the dataloader, since the not
all the images has the same number of objects
    def collate_fn(self, batch):
        return tuple(zip(*batch))

    def __len__(self):
        return len(self.db)
```

```
    def __getitem__(self, idx):
        image = self.get_image(idx, False)
        target = self._get_labels(idx)

        image, masks, bboxes, labels =
self.data_augmentation(np.array(image), target['masks'],
target['boxes'],

target['labels'])

        target['boxes'] = torch.tensor(bboxes, dtype=torch.float32)

        target['labels'] = torch.tensor(labels, dtype=torch.int64)

        target['masks'] = torch.tensor(np.array(masks, dtype=np.uint8))

        image_id = torch.tensor([idx])
        area = (target['boxes'][:, 3] - target['boxes'][:, 1]) *
(target['boxes'][:, 2] - target['boxes'][:, 0])
        # suppose all instances are not crowd
        iscrowd = torch.zeros((len(self.cls_to_idx),),
dtype=torch.int64)

        target['image_id'] = image_id
        target['area'] = area
        target['iscrowd'] = iscrowd

        image = self.image_transform(image)

        return image, target
```

Bdd_utils.py

```
# Import Libraries
import numpy as np
import random
import json
import matplotlib.pyplot as plt
from matplotlib.path import Path
import torch
from torchvision.io import read_image
from torchvision.ops import masks_to_boxes


######################################################################
########################################
# DISPLAY
######################################################################
########################################
def to_mask(mask_shape, poly2d):
    """
    function to convert 2D polygon to 2D mask
    :param mask_shape: the shape of the mask we want to return
    :param poly2d: a list of x and y coordinates
    :return: np.array object
```

```
    """
    nx, ny = mask_shape[1], mask_shape[0]
    # Create vertex coordinates for each grid cell...
    # (<0,0> is at the top left of the grid in this system)
    x, y = np.meshgrid(np.arange(nx), np.arange(ny))
    x, y = x.flatten(), y.flatten()

    points = np.vstack((x, y)).T

    path = Path(poly2d)
    grid = path.contains_points(points)
    grid = grid.reshape((ny, nx))

    return grid


def bbox_from_instance_mask(mask):
    """
    Function to extract bounding boxes from masks
    :param mask_path: str, path for the mask
    :return: Tensor contains the bounding boxes
    """
    mask = torch.tensor(mask, dtype=torch.uint8)  # read the mask as a
tensor type
    obj_ids = torch.unique(mask)  # get the unique colors
    obj_ids = obj_ids[1:]  # get rid of the last color which is 2 (in
my case 2 is the background)

    masks = mask == obj_ids[:, None, None]  # split the color-encoded
mask into a set of boolean masks.
    boxes = masks_to_boxes(masks)  # get the bounding boxes
    if len(boxes):
        return boxes[0].tolist()
    else:
        return None


def get_coloured_mask(mask):
    """
    random_colour_masks
      parameters:
        - image - predicted masks
      method:
        - the masks of each predicted object is given random colour for
visualization
    """
    colours = [[0, 255, 0], [0, 0, 255], [255, 0, 0], [0, 255, 255],
[255, 255, 0], [255, 0, 255], [80, 70, 180],
               [250, 80, 190], [245, 145, 50], [70, 150, 250], [50,
190, 190]]
    r = np.zeros_like(mask).astype(np.uint8)
    g = np.zeros_like(mask).astype(np.uint8)
    b = np.zeros_like(mask).astype(np.uint8)
    color = colours[random.randrange(0, 10)]
    r[mask == 1], g[mask == 1], b[mask == 1] = color
    coloured_mask = np.stack([r, g, b], axis=2)
    return coloured_mask, color
```

Detection_models.py

```
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor,
fasterrcnn_resnet50_fpn, \
    fasterrcnn_resnet50_fpn_v2, fasterrcnn_mobilenet_v3_large_fpn
from torchvision.models.detection import
FasterRCNN_ResNet50_FPN_V2_Weights,
FasterRCNN_MobileNet_V3_Large_FPN_Weights
from torchvision.models import ResNet50_Weights,
MobileNet_V3_Large_Weights


def get_fasterrcnn(num_classes: int = 91,
                   backbone: str = 'resnet50',
                   pretrained: bool = True,
                   pretrained_backbone: bool = True) -> FasterRCNN:
    """
    function to return the Faster RCNN model
[https://arxiv.org/abs/1506.01497]
    :param num_classes: the number of classes (plus background)
    :param backbone: the backbone we want to use
    :param pretrained: if we want to use pretrained weights or not
    :param pretrained_backbone: if we want to use pretrained backbone
or not
    :return: FasterRCNN instance
    """

    # Resnet50 as backbone
    if backbone == 'resnet50':
        # if pretrained weights
        if pretrained:
            params_ = {
                'weights': FasterRCNN_ResNet50_FPN_V2_Weights.DEFAULT,
                'weights_backbone': ResNet50_Weights.DEFAULT
            }
            model = fasterrcnn_resnet50_fpn_v2(**params_)

        # if pretrained backbone
        elif pretrained_backbone:
            params_ = {
                'weights_backbone': ResNet50_Weights.DEFAULT
            }
            model = fasterrcnn_resnet50_fpn_v2(**params_)

        # training from scratch
        else:
            model = fasterrcnn_resnet50_fpn_v2()

    # MobileNet backbone
    elif backbone == 'MobileNetV3-Large':
        # if pretrained weights
        if pretrained:
            params_ = {
                'weights':
FasterRCNN_MobileNet_V3_Large_FPN_Weights.DEFAULT,
                'weights_backbone': MobileNet_V3_Large_Weights.DEFAULT
```

```
            }
            model = fasterrcnn_mobilenet_v3_large_fpn(**params_)

        # if pretrained backbone
        elif pretrained_backbone:
            params_ = {
                'weights_backbone': MobileNet_V3_Large_Weights.DEFAULT
            }
            model = fasterrcnn_mobilenet_v3_large_fpn(**params_)

        # training from scratch
        else:
            model = fasterrcnn_mobilenet_v3_large_fpn()

    # if the number of classes different from 91 (the number of classes
of MSCOCO)
    # in case we want to use pretrained weights
    if num_classes != 91:
        in_features =
model.roi_heads.box_predictor.cls_score.in_features
        model.roi_heads.box_predictor = FastRCNNPredictor(in_features,
num_classes)

    return model
```

Faster_RCNN.py

```
import json
from pathlib import Path
import io
import os
from PIL import Image

import torch
from torchvision import transforms
from .detection_models import get_fasterrcnn
import pytorch_lightning as pl
from torchmetrics.detection.mean_ap import MeanAveragePrecision


class Faster_RCNN(pl.LightningModule):

    def __init__(self,
                 cfg,
                 num_classes,
                 backbone,
                 learning_rate,
                 weight_decay,
                 pretrained,
                 pretrained_backbone,
                 ):
        super(Faster_RCNN, self).__init__()

        assert 0 <= learning_rate <= 1, "Learning Rate must be between
0 and 1"
        assert backbone in cfg.DETECTION.BACKBONE, f"Please choose
```

```
backbone from the following: {cfg.DETECTION.BACKBONE}"
        assert num_classes > 0, "Number of classes must be greater than
0."

        self.learning_rate = learning_rate
        self.weight_decay = weight_decay
        self.num_classes = num_classes
        self.save_hyperparameters()

        self.model = get_fasterrcnn(self.num_classes, backbone,
pretrained, pretrained_backbone)

        self.transform = transforms.Compose([
            transforms.Resize(640),
            transforms.ToTensor(),  # convert the image to tensor
            transforms.Normalize(mean=[0.407, 0.457, 0.485],
                                 std=[0.229, 0.224, 0.225])  #
normalize the image using mean ans std
        ])
        self.metric = MeanAveragePrecision()
        self.metric_person = MeanAveragePrecision()
        self.metric_car = MeanAveragePrecision()
        self.metric_tl = MeanAveragePrecision()
        self.metric_ts = MeanAveragePrecision()

    def forward(self, x, *args, **kwargs):
        return self.model(x)

    def training_step(self, train_batch, batch_idx):
        images, targets = train_batch

        targets = [{k: v for k, v in t.items()} for t in targets]

        loss_dict = self.model(images, targets)
        train_loss = sum(loss for loss in loss_dict.values())

        # self.log('train_loss', loss, on_step=True, on_epoch=True,
prog_bar=True, logger=True)

        return train_loss

    def training_epoch_end(self, training_step_outputs):
        epoch_losses = torch.tensor([batch_loss['loss'].item() for
batch_loss in training_step_outputs])
        # epoch_losses = torch.stack(training_step_outputs)
        loss_mean = torch.mean(epoch_losses)
        self.log('training_loss', loss_mean)

    def validation_step(self, val_batch, batch_idx):
        self.model.train()
        images, targets = val_batch

        targets = [{k: v for k, v in t.items()} for t in targets]

        outputs = self.model(images, targets)

        val_loss = sum(loss for loss in outputs.values())
```

```
        return val_loss

    def validation_epoch_end(self, validation_step_outputs):
        epoch_losses = torch.tensor([batch_loss.item() for batch_loss
in validation_step_outputs])
        # epoch_losses = torch.stack(training_step_outputs)
        loss_mean = torch.mean(epoch_losses)
        self.log('val_loss', loss_mean)

    def filter_gt(self, pr, idx_label):
        boxes = pr['boxes'].tolist()
        labels = pr['labels'].tolist()
        pred = [{
            'boxes': [],
            'labels': [],
        }]
        for idx, label in enumerate(labels):
            if label == idx_label:
                pred[0]['boxes'].append(boxes[idx])
                pred[0]['labels'].append(labels[idx])
        if len(pred[0]['boxes']) > 0:
            pred[0]['boxes'] = torch.Tensor(pred[0]['boxes'])
            pred[0]['labels'] = torch.Tensor(pred[0]['labels'])
        return pred

    def filter_prediction(self, pr, idx_label):
        boxes = pr['boxes'].tolist()
        labels = pr['labels'].tolist()
        scores = pr['scores'].tolist()
        pred = [{
            'boxes': [],
            'labels': [],
            'scores': []
        }]
        for idx, label in enumerate(labels):
            if label == idx_label:
                pred[0]['boxes'].append(boxes[idx])
                pred[0]['labels'].append(labels[idx])
                pred[0]['scores'].append(scores[idx])

        pred[0]['boxes'] = torch.Tensor(pred[0]['boxes'])
        pred[0]['labels'] = torch.Tensor(pred[0]['labels'])
        pred[0]['scores'] = torch.Tensor(pred[0]['scores'])
        return pred

    def predict_step(self, batch, batch_idx):
        self.model.eval()
        images, targets = batch
        targets = [{k: v for k, v in t.items()} for t in targets]
        outputs = self.model(images)
        self.metric.update(outputs, targets)
        # Person
        pred_person = self.filter_prediction(outputs[0], 1)
        gt_person = self.filter_gt(targets[0], 1)

        if len(gt_person[0]['boxes']) > 0:
            self.metric_person.update(pred_person, gt_person)
```

```
        # Cars
        pred_car = self.filter_prediction(outputs[0], 2)
        gt_car = self.filter_gt(targets[0], 2)

        if len(gt_car[0]['boxes']) > 0:
            self.metric_car.update(pred_car, gt_car)

        # tl
        pred_tl = self.filter_prediction(outputs[0], 3)
        gt_tl = self.filter_gt(targets[0], 3)

        if len(gt_tl[0]['boxes']) > 0:
            self.metric_tl.update(pred_tl, gt_tl)
        # ts
        pred_ts = self.filter_prediction(outputs[0], 4)
        gt_ts = self.filter_gt(targets[0], 4)

        if len(gt_ts[0]['boxes']) > 0:
            self.metric_ts.update(pred_ts, gt_ts)

        return outputs

    def predict(self, image, confidence_score, device):
        self.model.eval()
        if isinstance(image, (bytes, bytearray)):
            image = Image.open(io.BytesIO(image))
        elif isinstance(image, str):
            assert os.path.exists(image), "This image doesn't exists"
            image = Image.open(image)

        tensor_image = self.transform(image)
        tensor_image = tensor_image.unsqueeze(0)
        tensor_image = tensor_image.to(device)
        model = self.model.to(device)
        yhat = model(tensor_image)
        yhat = yhat[0]
        boxes = yhat['boxes'].tolist()
        labels = yhat['labels'].tolist()
        scores = yhat['scores'].tolist()
        prediction = {
            "boxes": [],
            "scores": [],
            "labels": [],
        }

        for idx, score in enumerate(scores):
            if score > confidence_score:
                prediction['boxes'].append(boxes[idx])
                prediction['scores'].append(scores[idx])
                prediction['labels'].append(labels[idx])

        return prediction

    def configure_optimizers(self):
        optimizer_params = {
            'params': self.model.parameters(),
```

```
            'lr': self.learning_rate,
            'weight_decay': self.weight_decay,
        }
        return torch.optim.Adam(**optimizer_params)
```

segmentation_model.py

```
from torchvision.models.segmentation import DeepLabV3
from torchvision.models.segmentation import deeplabv3_resnet50,
deeplabv3_resnet101, deeplabv3_mobilenet_v3_large
from torchvision.models.segmentation import DeepLabV3_ResNet50_Weights,
DeepLabV3_ResNet101_Weights, \
    DeepLabV3_MobileNet_V3_Large_Weights
from torchvision.models import ResNet50_Weights, ResNet101_Weights,
MobileNet_V3_Large_Weights
from torchvision.models.segmentation.deeplabv3 import DeepLabHead


from torchvision.models.detection import MaskRCNN
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor
from torchvision.models.detection import MaskRCNN
from torchvision.models.detection import maskrcnn_resnet50_fpn,
maskrcnn_resnet50_fpn_v2
from torchvision.models.detection import MaskRCNN_ResNet50_FPN_Weights,
MaskRCNN_ResNet50_FPN_V2_Weights
from torchvision.models.resnet import ResNet50_Weights


def get_deeplab(num_classes: int = 21,
                backbone: str = 'resnet50',
                pretrained: bool = True,
                pretrained_backbone: bool = True) -> DeepLabV3:
    """
    function to return the Deeplabv3+ model
[https://arxiv.org/abs/1706.05587]
    :param num_classes: the number of classes (plus background)
    :param backbone: the backbone we want to use
    :param pretrained: if we want to use pretrained weights or not
    :param pretrained_backbone: if we want to use pretrained backbone
or not
    :return: DeepLabV3 instance
    """

    # Resnet50 as backbone
    if backbone == 'resnet50':
        # if pretrained weights
        if pretrained:
            model_params = {
                'weights': DeepLabV3_ResNet50_Weights.DEFAULT
            }
            model = deeplabv3_resnet50(**model_params)

        # if pretrained backbone
        elif pretrained_backbone:
            model_params = {
```

```
                'weights_backbone': ResNet50_Weights.DEFAULT
            }
            model = deeplabv3_resnet50(**model_params)
        # training from scratch
        else:
            model = deeplabv3_resnet50()

    # Resnet101 as backbone
    if backbone == 'resnet101':
        # if pretrained weights
        if pretrained:
            model_params = {
                'weights': DeepLabV3_ResNet101_Weights.DEFAULT
            }
            model = deeplabv3_resnet101(**model_params)

        # if pretrained backbone
        elif pretrained_backbone:
            model_params = {
                'weights_backbone': ResNet101_Weights.DEFAULT
            }
            model = deeplabv3_resnet101(**model_params)
        # training from scratch
        else:
            model = deeplabv3_resnet101()

    # MobileNet backbone
    if backbone == 'mobilenet':
        # if pretrained weights
        if pretrained:
            model_params = {
                'weights': DeepLabV3_MobileNet_V3_Large_Weights.DEFAULT
            }
            model = deeplabv3_mobilenet_v3_large(**model_params)

        # if pretrained backbone
        elif pretrained_backbone:
            model_params = {
                'weights_backbone': MobileNet_V3_Large_Weights.DEFAULT
            }
            model = deeplabv3_mobilenet_v3_large(**model_params)
        # training from scratch
        else:
            model = deeplabv3_mobilenet_v3_large()

    # if the number of classes different from 91 (the number of classes
of MSCOCO)
    # in case we want to use pretrained weights
    if num_classes != 21:
        model.classifier = DeepLabHead(2048, num_classes)

    return model


def get_maskrcnn(version: str = 'v2',
                 pretrained: bool = True,
                 pretrained_weights: bool = True,
```

```
                num_classes: int = 91) -> MaskRCNN:
    """
    function to return the Mask RCNN model
    :param version: which version we want to use
    :param pretrained: boolean value, if we want to use pretrained
weights on COCO dataset
    :param pretrained_weights: boolean value, if we want to use
pretrained backbone weights
    :param num_classes: the number of classes we want to classify
(Note: classes + background)
    :return:
    """

    assert version in ['v1', 'v2'], 'You have to choose which version
you want to use:\n ' \
                                    'v1 is Mask-RCNN with Resnet50
backbone and FPN network \n' \
                                    'v2 is improved version of Mask-
RCNN with vision transformer.'

    if version == 'v1':
        if pretrained:
            model =
maskrcnn_resnet50_fpn(weights=MaskRCNN_ResNet50_FPN_Weights.DEFAULT)
        elif pretrained_weights:
            model =
maskrcnn_resnet50_fpn(weights_backbone=ResNet50_Weights.DEFAULT)

        else:
            model = maskrcnn_resnet50_fpn()

        if num_classes != 91:
            # get the number of input features for the classifier
            in_features =
model.roi_heads.box_predictor.cls_score.in_features
            # now get the number of input features for the mask
classifier
            in_features_mask =
model.roi_heads.mask_predictor.conv5_mask.in_channels
            hidden_layer = 256
            # replace the pre-trained head with a new one
            model.roi_heads.box_predictor =
FastRCNNPredictor(in_features, num_classes)
            model.roi_heads.mask_predictor =
MaskRCNNPredictor(in_features_mask,

hidden_layer,

num_classes)

    elif version == 'v2':
        if pretrained:
            model =
maskrcnn_resnet50_fpn_v2(weights=MaskRCNN_ResNet50_FPN_V2_Weights.DEFAU
LT)
        elif pretrained_weights:
            model =
```

```
maskrcnn_resnet50_fpn_v2(weights_backbone=ResNet50_Weights.DEFAULT)

        else:
            model = maskrcnn_resnet50_fpn_v2()

        if num_classes != 91:
            # get the number of input features for the classifier
            in_features =
model.roi_heads.box_predictor.cls_score.in_features
            # now get the number of input features for the mask
classifier
            in_features_mask =
model.roi_heads.mask_predictor.conv5_mask.in_channels
            hidden_layer = 256
            # replace the pre-trained head with a new one
            model.roi_heads.box_predictor =
FastRCNNPredictor(in_features, num_classes)
            model.roi_heads.mask_predictor =
MaskRCNNPredictor(in_features_mask,

hidden_layer,

num_classes)

    return model
```

MASKRCNN.py

```
import torch
import torch.nn as nn

import torchvision
import pytorch_lightning as pl

from .segmentation_models import get_maskrcnn
from torchmetrics.detection.mean_ap import MeanAveragePrecision


class Mask_RCNN(pl.LightningModule):

    def __init__(self,
                 cfg,
                 num_classes: int,
                 version: str,
                 learning_rate: float,
                 weight_decay: float,
                 pretrained: bool,
                 pretrained_backbone: bool,
                 ):
        """
        Constructor for the Mask_RCNN class
        :param cfg: yacs configuration that contains all the necessary
information about the available backbones
        :param num_classes: the number of classes we want to classify
for each pixel (with background)
        :param version: the version of the mask rcnn implementation you
```

```
want to use
        :param learning_rate: The learning rate for the network
        :param weight_decay: decay for the regularization
        :param pretrained: bool value for pretrained network
        :param pretrained_backbone: bool value for pretrained backbone
        """
        super(Mask_RCNN, self).__init__()

        assert 0 <= learning_rate <= 1, "Learning Rate must be between
0 and 1"
        assert version in cfg.INSTANCE_SEGMENTATION.VERSION, f"Please
choose version from the following: {cfg.INSTANCE_SEGMENTATION.VERSION}"
        assert num_classes > 0, "Number of classes must be greater than
0."

        self.learning_rate = learning_rate
        self.weight_decay = weight_decay
        self.num_classes = num_classes
        self.save_hyperparameters()

        self.model = get_maskrcnn(version=version,
                                  pretrained=pretrained,

pretrained_weights=pretrained_backbone,
                                  num_classes=self.num_classes)
        self.metric = MeanAveragePrecision(class_metrics=True)

    def forward(self, x):
        return self.model(x)

    def training_step(self, train_batch, batch_idx) -> float:
        images, targets = train_batch

        targets = [{k: v for k, v in t.items()} for t in targets]

        loss_dict = self.model(images, targets)
        losses = sum(loss for loss in loss_dict.values())

        return losses

    def training_epoch_end(self, training_step_outputs) -> dict:
        epoch_losses = torch.tensor([batch_loss['loss'].item() for
batch_loss in training_step_outputs])
        # epoch_losses = torch.stack(training_step_outputs)
        loss_mean = torch.mean(epoch_losses)
        self.log('training_loss', loss_mean)

    def validation_step(self, val_batch, batch_idx) -> float:
        self.model.train()
        images, targets = val_batch
        targets = [{k: v for k, v in t.items()} for t in targets]
        loss_dict = self.model(images, targets)
        va_losses = sum(loss for loss in loss_dict.values())
        return va_losses

    def validation_epoch_end(self, validation_step_outputs) -> dict:
        epoch_losses = torch.tensor([batch_loss.item() for batch_loss
```

```
in validation_step_outputs])
        # epoch_losses = torch.stack(training_step_outputs)
        loss_mean = torch.mean(epoch_losses)
        self.log('val_loss', loss_mean)

    def predict_step(self, batch, batch_idx):
        self.model.eval()
        images, targets = batch
        targets = [{k: v for k, v in t.items()} for t in targets]
        outputs = self.model(images)
        self.metric.update(outputs, targets)


    def configure_optimizers(self):
        optimizer_params = {
            'params': self.model.parameters(),
            'lr': self.learning_rate,
            'weight_decay': self.weight_decay,
        }
        return torch.optim.Adam(**optimizer_params)
```

DeepLab.py

```
# Import Libraries
import torch
import torch.nn as nn

import torchvision
from .segmentation_models import get_deeplab

from torch.utils.data import DataLoader
import pytorch_lightning as pl


class DeepLab(pl.LightningModule):
    """
    DeepLab Class for semantic segmentation task
    """

    def __init__(self,
                 cfg,
                 num_classes: int,
                 backbone: str,
                 learning_rate: float,
                 weight_decay: float,
                 pretrained: bool,
                 pretrained_backbone: bool,
                 ):
        """
        Constructor for the DeepLab class
        :param cfg: yacs configuration that contains all the necessary
information about the available backbones
        :param num_classes: the number of classes we want to classify
for each pixel (with background)
        :param backbone: the name of the backbone we want to use
        :param learning_rate: The learning rate for the network
        :param weight_decay: decay for the regularization
```

```
        :param pretrained: bool value for pretrained network
        :param pretrained_backbone: bool value for pretrained backbone
        """
        super(DeepLab, self).__init__()

        assert 0 <= learning_rate <= 1, "Learning Rate must be between
0 and 1"
        assert backbone in cfg.DRIVABLE_AREA.DEEPLAB_BACKBONE, f"Please
choose backbone from the following:
{cfg.DRIVABLE_AREA.DEEPLAB_BACKBONE} "
        assert num_classes > 0, "Number of classes must be greater than
0."

        self.learning_rate = learning_rate
        self.weight_decay = weight_decay
        self.num_classes = num_classes
        self.backbone = backbone

        self.model = get_deeplab(self.num_classes, self.backbone,
pretrained, pretrained_backbone)

        self.criterion = nn.CrossEntropyLoss()

        self.imagenet_stats = [[0.485, 0.456, 0.406], [0.485, 0.456,
0.406]]

    def forward(self, x):
        self.model.eval()
        return self.model(x)

    def training_step(self, train_batch, batch_idx):
        images, targets = train_batch
        outputs = self.model(images)['out']
        train_loss = self.criterion(outputs, targets['mask'])
        return train_loss

    def training_epoch_end(self, training_step_outputs):
        train_loss_mean =
torch.mean(torch.stack(training_step_outputs))
        self.log('training_loss', train_loss_mean.item())

    def validation_step(self, val_batch, batch_idx):
        images, targets = val_batch
        outputs = self.model(images)['out']
        val_loss = self.criterion(outputs, targets['mask'])
        return val_loss

    def validation_epoch_end(self, validation_step_outputs):
        val_loss_mean =
torch.mean(torch.stack(validation_step_outputs))
        self.log('training_loss', val_loss_mean.item())

    def image_transform(self, image):
        reprocess =
torchvision.transforms.Compose([torchvision.transforms.ToTensor(),

torchvision.transforms.Normalize(mean=self.imagenet_stats[0],
```

```
std=self.imagenet_stats[1])])

        return reprocess(image)

    def predict_step(self, image):
        input_tensor = self.image_transform(image).unsqueeze(0)

        # Make the predictions for labels across the image
        self.model.eval()
        with torch.no_grad():
            output = self.model(input_tensor)["out"][0]
            output = output.argmax(0)

        # Return the predictions

        return output.cpu().numpy()

    def configure_optimizers(self):
        optimizer_params = {
            'params': self.model.parameters(),
            'lr': self.learning_rate,
            'weight_decay': self.weight_decay,
        }
        return torch.optim.Adam(**optimizer_params)
```

Dataloaders.py

```
from torch.utils.data import DataLoader


def get_loader(dataset,
               batch_size,
               shuffle,
               collate_fn=None,
               pin_memory=False,
               drop_last=False,
               num_workers=0,
               persistent_workers=False):
    params = {
        'dataset': dataset,
        'batch_size': batch_size,
        'shuffle': shuffle,
        'collate_fn': collate_fn,
        'drop_last': drop_last,
        'pin_memory': pin_memory,
        'num_workers': num_workers,
        'persistent_workers': persistent_workers
    }
    return DataLoader(**params)
```

utils.py

```
# Import Libraries
import numpy as np
```

```
import os
import shutil
from collections import deque
from tqdm import tqdm
from PIL import Image
import matplotlib.pyplot as plt
from matplotlib.path import Path
import torch
from torchvision.io import read_image
from torchvision.ops import masks_to_boxes

from torchvision.utils import draw_bounding_boxes


################################################################
#  Bounding Boxes
################################################################


def extract_bbox_from_mask(mask_path):
    """
    Function to extract bounding boxes from masks
    :param mask_path: str, path for the mask
    :return: Tensor contains the bounding boxes
    """
    mask = read_image(mask_path)  # read the mask as a tensor type
    obj_ids = torch.unique(mask)  # get the unique colors
    obj_ids = obj_ids[:-1]  # get rid of the last color which is 2 (in
my case 2 is the background)

    masks = mask == obj_ids[:, None, None]  # split the color-encoded
mask into a set of boolean masks.
    boxes = masks_to_boxes(masks)  # get the bounding boxes

    return boxes


################################################################
#  Data Format
################################################################

# method to convert xyxy to xywh yolo format
def convert_pascal_to_yolo(xmin, ymin, xmax, ymax, img_w, img_h):
    dw = 1. / (img_w)
    dh = 1. / (img_h)
    x = (xmin + xmax) / 2.0 - 1
    y = (ymin + ymax) / 2.0 - 1
    w = xmax - xmin
    h = ymax - ymin
    x = round(x * dw, 4)
    w = round(w * dw, 4)
    y = round(y * dh, 4)
    h = round(h * dh, 4)
    return (x, y, w, h)


def create_yolo_annotation(bdd):
```

```
    """
    method to convert to create new db from the bdd.db object that have
yolo format
    :param bdd: BDD class instance
    :return: yolo_deque object that has all the annotation in yolo
format
    """
    yolo_deque = deque()
    # YOLO_Format = namedtuple("YOLO", "cls x y w h")
    print("Start converting.")
    for item in tqdm(bdd.db):
        image_path = item['image_path']
        image = Image.open(image_path)
        width, height = image.size
        boxes = item['bboxes']
        classes = item['classes']
        yolo_boxes = []
        for i, box in enumerate(boxes):
            cls = classes[i]
            x, y, w, h = convert_pascal_to_yolo(box[0], box[1], box[2],
box[3], width, height)
            # yolo_format = YOLO_Format(cls, x, y, w, h)
            yolo_boxes.append([cls, x, y, w, h])

        yolo_deque.append({
            'image_name': item['image_path'].split('\\')[-1],
            'image_path': image_path,
            'width': width,
            'height': height,
            'yolo_boxes': yolo_boxes
        })
    print("Finish from converting")
    return yolo_deque


def move_files(yolo_deque, images_folder_destination,
labels_folder_destination):
    """
    method to copy the files from the original source to a new
destination
    :param yolo_deque: deque object, which is the output of the
create_yolo_annotation method
    :param images_folder_destination: the path to the folder where we
want to copy the images to
    :param labels_folder_destination: the path to the folder where we
want to create the labels (txt files)
    :return: None
    """
    assert os.path.isdir(images_folder_destination), "Folder does not
exist!"
    assert os.path.isdir(labels_folder_destination), "Folder does not
exist!"

    print("Start copying files.")
    for item in tqdm(yolo_deque):
        # shutil.copy(item['image_path'],
os.path.join(folder_destination, 'images', stage))
```

```
        # create_annotation_file(item, folder_destination, stage)
        shutil.copy(item['image_path'], images_folder_destination)
        create_annotation_file(item, labels_folder_destination)

    print(f"All images are in {images_folder_destination} and all
labels are in {labels_folder_destination}.")


def create_annotation_file(yolo_item, labels_folder_destination):
    text_file_name = yolo_item['image_name'].replace('.jpg',
'.txt').split('/')[-1]
    file_path = os.path.join(labels_folder_destination, text_file_name)
    file = open(file_path, 'a')
    for line in yolo_item['yolo_boxes']:
        file.write(" ".join(str(item) for item in line))
        file.write('\n')
    file.close()

#############################################################
#  Masks
#############################################################

def to_mask(mask_shape, poly2d):
    """
    function to convert 2D polygon to 2D mask
    :param mask_shape: the shape of the mask we want to return
    :param poly2d: a list of x and y coordinates
    :return: np.array object
    """
    nx, ny = mask_shape[1], mask_shape[0]
    # Create vertex coordinates for each grid cell...
    # (<0,0> is at the top left of the grid in this system)
    x, y = np.meshgrid(np.arange(nx), np.arange(ny))
    x, y = x.flatten(), y.flatten()

    points = np.vstack((x, y)).T

    path = Path(poly2d)
    grid = path.contains_points(points)
    grid = grid.reshape((ny, nx))

    return grid


def bbox_from_instance_mask(mask_path):
    """
    Function to extract bounding boxes from masks
    :param mask_path: str, path for the mask
    :return: Tensor contains the bounding boxes
    """
    mask = read_image(mask_path)  # read the mask as a tensor type
    obj_ids = torch.unique(mask)  # get the unique colors
    obj_ids = obj_ids[1:]  # get rid of the last color which is 2 (in
my case 2 is the background)

    masks = mask == obj_ids[:, None, None]  # split the color-encoded
mask into a set of boolean masks.
```

```
    boxes = masks_to_boxes(masks)  # get the bounding boxes

    return boxes
```

Detect.py

```
from src.models.Detection.Faster_RCNN import Faster_RCNN
from src.models.Segmentation.MaskRCNN import Mask_RCNN
from util import *
import warnings
import argparse
import yaml
import torch
import sys

warnings.filterwarnings("ignore")

COLOR_MAP = ['blue', 'orange', 'green', 'red', 'purple', 'brown',
'pink', 'gray', 'olive', 'cyan', 'blue']

if __name__ == '__main__':
    # Define the parser
    parser = argparse.ArgumentParser()
    parser.add_argument('--source', type=str, help='the path of the
image or video')
    parser.add_argument('--data', type=str,
default="./data/fasterrcnn.yaml", help='data.yaml path')
    parser.add_argument('--weights', type=str, default=None,
help='train from checkpoint')
    parser.add_argument('--confidence-score', type=float, default=0.5,
help='confidence score used to predict')
    parser.add_argument('--model', type=str, default='fasterrcnn',
choices=['fasterrcnn', 'deeplab', 'maskrcnn', 'yolov5', 'yolov7'],
                        help='the model and task you want to perform')
    parser.add_argument('--save-path', type=str,
default="predicted_image.jpg",
                        help='Path to save the image with bounding
boxes')

    # Fetch the params from the parser
    args = parser.parse_args()

    source = args.source
    confidence_score = args.confidence_score
    weights = args.weights  # Check point to continue training
    save_path = args.save_path
    model_name = args.model  # the name of the model: fastercnn,
maskrcnn, deeplab

    with open(args.data, 'r') as f:
        data = yaml.safe_load(f)  # data from .yaml file

    obj_cls = data['classes']  # the classes we want to work one
    relative_path = data['relative_path']  # relative path to the
dataset
```

```
    # create idx and cls dict
    idx_to_cls = create_cls_dic(obj_cls)

    # check source path
    assert os.path.exists(source), "This image doesn't exists"

    # Load model
    if model_name == 'fasterrcnn':
        try:
            model = Faster_RCNN.load_from_checkpoint(weights)  # Faster
RCNN
            output, fps = detection_predict(model=model, image=source,
confidence_score=confidence_score)
            display(image=source, prediction=output,
save_path=save_path,
                    idx_to_cls=idx_to_cls)  # display result and save
it
        except Exception as e:
            print("Could not load the model weights. Please make sure
you're providing the correct model weights.")
            sys.exit()
    elif model_name == 'maskrcnn':
        try:
            model = Mask_RCNN.load_from_checkpoint(weights)  # Mask
RCNN
            output, fps = detection_predict(model=model, image=source,
confidence_score=confidence_score)
            display(image=source, prediction=output,
save_path=save_path,
                    idx_to_cls=idx_to_cls)  # display result and save
it
        except Exception as e:
            print("Could not load the model weights. Please make sure
you're providing the correct model weights.")
            sys.exit()

    elif model_name.startswith('yolov5'):
        try:
            device = torch.device('cuda' if torch.cuda.is_available()
else 'cpu')
            model = torch.hub.load('./yolov5', 'custom', path=weights,
source='local', device=device, _verbose=False)
            model.conf = confidence_score
        except Exception as e:
            print("Could not load the model weights. Please make sure
you're providing the correct model weights.")
            sys.exit()
        try:
            image = Image.open(source)
            results = model(image)
            results.print()
            results.save(save_path)
        except Exception as e:
            print(e)
            sys.exit()

    elif model_name.startswith('yolov7'):
```

```
        try:
            device = torch.device('cuda' if torch.cuda.is_available()
else 'cpu')
            model = torch.hub.load('../yolov7', 'custom',
path_or_model=weights, source='local')
            model.conf = confidence_score
        except Exception as e:
            print("Could not load the model weights. Please make sure
you're providing the correct model weights.")
            sys.exit()
        try:
            image = Image.open(source)
            results = model(image)
            results.print()
            results.save(save_path)
        except Exception as e:
            print(e)
            sys.exit()


    if model_name == 'fasterrcnn' or model_name == 'maskrcnn':
        # get prediction
        output, fps = detection_predict(model=model, image=source,
confidence_score=confidence_score)
        print(f"Frame per Second: {fps}")  # speed
        display(image=source, prediction=output, save_path=save_path,
                idx_to_cls=idx_to_cls)  # display result and save it
```

prepare.py

```
import os
import yaml
import argparse
from src.config.defaults import cfg
from src.utils.utils import *
from src.dataset.bdd_detetcion import BDDDetection

if __name__ == '__main__':
    # Define the parser
    parser = argparse.ArgumentParser()
    parser.add_argument('--data', type=str, default='', help='')
    parser.add_argument('--yolo_version', type=str, default='yolov5',
choices=['yolov5', 'yolov7'], help='which '

'version of '

'yolo do you '

'want to use')
    parser.add_argument('--dataset_path', type=str, help='The path to
the dataset folder where you want to upload the '
                                                         'data')

    # Fetch the params from the parser
    args = parser.parse_args()
    version = args.yolo_version
```

```
    dataset_path = args.dataset_path

    with open(args.data, 'r') as f:
        data = yaml.safe_load(f)  # data from .yaml file

    obj_cls = data['classes']  # the classes we want to work one
    relative_path = data['relative_path']  # relative path to the
dataset

    ######################################## Datasets
########################################
    bdd_train_params = {
        'cfg': cfg,
        'relative_path': relative_path,
        'stage': 'train',
        'obj_cls': obj_cls,
    }

    bdd_train = BDDDetection(**bdd_train_params)

    bdd_val_params = {
        'cfg': cfg,
        'relative_path': relative_path,
        'stage': 'val',
        'obj_cls': obj_cls,
    }

    bdd_val = BDDDetection(**bdd_val_params)

    bdd_test_params = {
        'cfg': cfg,
        'stage': 'test',
        'relative_path': relative_path,
        'obj_cls': obj_cls,
    }

    bdd_test = BDDDetection(**bdd_test_params)
    print(50*'#')
    print(
        f"We have {len(bdd_train)} training images, {len(bdd_val)}
validation images and {len(bdd_test)} test images.")

    print(50 * '#')
    ######################################## Prepare Data
########################################

    """
    Yolov5 [https://github.com/ultralytics/yolov5] takes the data in
YOLO annotation format
    YOLO format is basically (x, y, w, h), like COCO format but
normalized
    the annotation must be in txt file, each image has it's own
annotation file
    The data should be in a folder and under this folder we should have
two folders: images and labels
    in images we should have three different folders: train, test, val
    same thing for labels
```

```
    """
    if version == 'yolov5':
        """
        Dataset folder's structure for Yolov5 is as follows:
        ├─dataset
        │ ├─images
        │ │ ├─train
        │ │ ├─test
        │ │ ├─val
        │ ├─labels
        │ │ ├─train
        │ │ ├─test
        │ │ ├─val
        """
        if not os.path.exists(os.path.join(dataset_path, 'images')):
            os.path.join(dataset_path, 'images')
            os.path.join(dataset_path, 'images', 'train')
            os.path.join(dataset_path, 'images', 'val')
            os.path.join(dataset_path, 'images', 'test')
        if not os.path.exists(os.path.join(dataset_path, 'labels')):
            os.mkdir(os.path.join(dataset_path, 'labels'))
            os.path.join(dataset_path, 'images', 'train')
            os.path.join(dataset_path, 'images', 'val')
            os.path.join(dataset_path, 'images', 'test')

        # train data
        images_training_path = os.path.join(dataset_path, 'images',
'train')
        labels_training_path = os.path.join(dataset_path, 'labels',
'train')

        # val data
        images_val_path = os.path.join(dataset_path, 'images', 'val')
        labels_val_path = os.path.join(dataset_path, 'labels', 'val')

        # test data
        images_test_path = os.path.join(dataset_path, 'images', 'test')
        labels_test_path = os.path.join(dataset_path, 'labels', 'test')

    elif version == 'yolov7':
        """
        Dataset folder's structure for Yolov7 is as follows:
        ├─dataset
        │ ├─train
        │ │ ├─images
        │ │ ├─labels
        │ ├─test
        │ │ ├─images
        │ │ ├─labels
        │ ├─val
        │ │ ├─images
        │ │ ├─labels
        """
        if not os.path.exists(os.path.join(dataset_path, 'train')):
            os.path.join(dataset_path, 'train')
            os.path.join(dataset_path, 'train', 'images')
            os.path.join(dataset_path, 'train', 'labels')
```

```
        if not os.path.exists(os.path.join(dataset_path, 'val')):
            os.mkdir(os.path.join(dataset_path, 'val'))
            os.path.join(dataset_path, 'val', 'images')
            os.path.join(dataset_path, 'val', 'labels')
        if not os.path.exists(os.path.join(dataset_path, 'test')):
            os.mkdir(os.path.join(dataset_path, 'test'))
            os.path.join(dataset_path, 'test', 'images')
            os.path.join(dataset_path, 'test', 'labels')

        # train data
        images_training_path = os.path.join(dataset_path, 'train',
'images')
        labels_training_path = os.path.join(dataset_path, 'train',
'labels')
        # val data
        images_val_path = os.path.join(dataset_path, 'val', 'images')
        labels_val_path = os.path.join(dataset_path, 'val', 'labels')
        # test data
        images_test_path = os.path.join(dataset_path, 'test', 'images')
        labels_test_path = os.path.join(dataset_path, 'test', 'labels')

    print(50 * '#')
    # create annotation for training
    yolo_train = create_yolo_annotation(bdd_train)
    # move the data to the specific path
    move_files(yolo_train, images_training_path, labels_training_path)
    print(50 * '#')

    # create annotation for validation
    yolo_val = create_yolo_annotation(bdd_val)
    # move the data to the specific path
    move_files(yolo_val, images_val_path, labels_val_path)
    print(50 * '#')
    # create annotation for test
    yolo_test = create_yolo_annotation(bdd_test)
    # move the data to the specific path
    move_files(yolo_test, images_test_path, labels_test_path)
    print(50 * '#')
```

test.py

```
import sys
import yaml
import argparse
from util import *
from src.models.Detection.Faster_RCNN import Faster_RCNN
from src.models.Segmentation.MaskRCNN import Mask_RCNN
from src.dataset.bdd_detetcion import BDDDetection
from src.dataset.bdd_instance_segmentation_drivable import
BDDInstanceSegmentationDrivable
from src.config.defaults import cfg
from src.utils.DataLoaders import get_loader
from pytorch_lightning import Trainer
import pandas as pd
from pprint import pprint
import warnings
```

```
warnings.filterwarnings("ignore")

if __name__ == '__main__':
    # Define the parser
    parser = argparse.ArgumentParser()
    parser.add_argument('--data', type=str,
default="./data/fasterrcnn.yaml", help='data.yaml path')
    parser.add_argument('--weights', type=str, default=None,
help='train from checkpoint')
    parser.add_argument('--pred', type=str, default='', help='Path to
the predictions folder')
    parser.add_argument('--gt', type=str, default='', help='Path to the
ground truth folder')
    parser.add_argument('--model', type=str, default='fasterrcnn',
choices=['fasterrcnn', 'deeplab', 'maskrcnn', 'yolov5', 'yolov7'],
                        help='the model and task you want to perform')
    parser.add_argument('--save-path', type=str,
default='./mAP_results.csv',
                        help='Path and name of the file you want to
export.')

    # Fetch the params from the parser
    args = parser.parse_args()

    weights = args.weights  # Check point to continue training
    save_path = args.save_path
    model_name = args.model  # the name of the model: fastercnn,
maskrcnn, deeplab

    with open(args.data, 'r') as f:
        data = yaml.safe_load(f)  # data from .yaml file

    obj_cls = data['classes']  # the classes we want to work one
    relative_path = data['relative_path']  # relative path to the
dataset

    if model_name == "fasterrcnn":
        ## Load Model
        try:
            model = Faster_RCNN.load_from_checkpoint(weights)
        except Exception as e:
            print("Could not load the model weights. Please make sure
you're providing the correct model weights.")
            sys.exit()

        bdd_params = {
            'cfg': cfg,
            'stage': 'test',
            'relative_path': relative_path,
            'obj_cls': obj_cls,
        }

        bdd = BDDDetection(**bdd_params)

        dataloader_args = {
            'dataset': bdd,
```

```
            'batch_size': 1,
            'shuffle': False,
            'collate_fn': bdd.collate_fn,
            'pin_memory': True,
            'num_workers': 1
        }
        dataloader = get_loader(**dataloader_args)

        trainer = Trainer(accelerator='gpu', devices=1)
        pred = trainer.predict(model, dataloader)

        print("Start Computing mAP for all classes.")
        mAP = model.metric.compute()

    elif model_name == 'maskrcnn':
        try:
            model = Mask_RCNN.load_from_checkpoint(weights)
        except Exception as e:
            print("Could not load the model weights. Please make sure
you're providing the correct model weights.")
            sys.exit()


        bdd_params = {
            'cfg': cfg,
            'stage': 'test',
            'relative_path': relative_path,
            'obj_cls': obj_cls,
        }

        bdd = BDDInstanceSegmentationDrivable(**bdd_params)

        dataloader_args = {
            'dataset': bdd,
            'batch_size': 1,
            'shuffle': False,
            'collate_fn': bdd.collate_fn,
            'pin_memory': True,
            'num_workers': 1
        }
        dataloader = get_loader(**dataloader_args)

        trainer = Trainer(accelerator='gpu', devices=1)
        pred = trainer.predict(model, dataloader)

        print("Start Computing mAP for all classes.")
        mAP = model.metric.compute()

    elif model_name.startswith('yolo'):
        """
        We evaluate Yolo models the same way
        """
        # get the path where the *.txt file are stored. Those files are
generated from the model
        prediction_path = args.pred
        # path to the ground truth folder, where we have the *.txt
file. Those files are genearted from the prepate.py file
```

```
        gt_path = args.gt

        # check paths
        assert os.path.exists(prediction_path), f"Predictions does not
exists at {prediction_path}"
        assert os.path.exists(gt_path), f"Predictions does not exists
at {gt_path}"

        # Evaluation
        mAP = yolo_evaluation(prediction_path, gt_path)

    # print
    pprint(mAP)
    export_map_json(mAP, json_file_name=f'mAP_{model_name}.json' ,
to_save_path=save_path)
```

train.py

```
import os
import yaml
import argparse

from src.models.Detection.Faster_RCNN import Faster_RCNN
from src.models.Segmentation.MaskRCNN import Mask_RCNN
from src.models.Segmentation.DeepLab import DeepLab
from src.dataset.bdd_detetcion import BDDDetection
from src.dataset.bdd_instance_segmentation import
BDDInstanceSegmentation
from src.dataset.bdd_drivable_segmentation import
BDDDrivableSegmentation
from src.config.defaults import cfg
from src.utils.DataLoaders import get_loader

import torch
from pytorch_lightning import Trainer
from pytorch_lightning.utilities.model_summary import ModelSummary
from pytorch_lightning.callbacks.early_stopping import EarlyStopping
from pytorch_lightning.callbacks import ModelCheckpoint
from pytorch_lightning.loggers import WandbLogger
from pytorch_lightning.loggers import CSVLogger


def get_datasets(model, relative_path, obj_cls):
    if model == 'fasterrcnn':
        # Training dataset
        bdd_train_params = {
            'cfg': cfg,
            'stage': 'train',
            'relative_path': relative_path,
            'obj_cls': obj_cls,
        }

        bdd_train = BDDDetection(**bdd_train_params)

        # Validation dataset
        bdd_val_params = {
```

```
            'cfg': cfg,
            'stage': 'val',
            'relative_path': relative_path,
            'obj_cls': obj_cls,
        }

        bdd_val = BDDDetection(**bdd_val_params)

    elif model == 'deeplab':
        # Training dataset
        bdd_train_params = {
            'cfg': cfg,
            'stage': 'train',
            'relative_path': relative_path,
            'obj_cls': obj_cls,
            'image_size': img_size
        }

        bdd_train = BDDDrivableSegmentation(**bdd_train_params)

        # Validation dataset
        bdd_val_params = {
            'cfg': cfg,
            'stage': 'val',
            'relative_path': relative_path,
            'obj_cls': obj_cls,
            'image_size': img_size
        }

        bdd_val = BDDDrivableSegmentation(**bdd_val_params)

    elif model == 'maskrcnn':
        # Training dataset
        bdd_train_params = {
            'cfg': cfg,
            'stage': 'train',
            'relative_path': relative_path,
            'obj_cls': obj_cls,
            'image_size': img_size
        }

        bdd_train = BDDInstanceSegmentation(**bdd_train_params)

        # Validation dataset
        bdd_val_params = {
            'cfg': cfg,
            'stage': 'val',
            'relative_path': relative_path,
            'obj_cls': obj_cls,
            'image_size': img_size
        }

        bdd_val = BDDInstanceSegmentation(**bdd_val_params)

    return bdd_train, bdd_val
```

```
def get_loaders(bdd_train, batch_size, pin_memory, num_workers):
    train_dataloader_args = {
        'dataset': bdd_train,
        'batch_size': batch_size,
        'shuffle': True,
        'collate_fn': bdd_train.collate_fn,
        'pin_memory': pin_memory,
        'num_workers': num_workers
    }
    train_dataloader = get_loader(**train_dataloader_args)

    # val dataloader
    val_dataloader_args = {
        'dataset': bdd_val,
        'batch_size': batch_size,
        'shuffle': False,
        'collate_fn': bdd_train.collate_fn,
        'pin_memory': pin_memory,
        'num_workers': num_workers
    }
    val_dataloader = get_loader(**val_dataloader_args)

    return train_dataloader, val_dataloader


def get_model(model_name, num_classes, backbone, lr, version):
    if model_name == 'fasterrcnn':
        faster_rcnn_params = {
            'cfg': cfg,
            'num_classes': num_classes,
            'backbone': backbone,
            'learning_rate': lr,
            'weight_decay': 1e-3,
            'pretrained': True,
            'pretrained_backbone': True,
        }
        model = Faster_RCNN(**faster_rcnn_params)


    elif model_name == 'deeplab':
        deeplab_params = {
            'cfg': cfg,
            'num_classes': num_classes,
            'backbone': backbone,
            'learning_rate': lr,
            'weight_decay': 1e-3,
            'pretrained': True,
            'pretrained_backbone': True,
        }
        model = DeepLab(**deeplab_params)

    elif model_name == 'maskrcnn':
        mask_rcnn_params = {
            'cfg': cfg,
            'num_classes': num_classes,
            'version': version,
            'learning_rate': lr,
```

```
            'weight_decay': 1e-3,
            'pretrained': True,
            'pretrained_backbone': True,
        }
        model = Mask_RCNN(**mask_rcnn_params)

    return model


if __name__ == '__main__':
    # Define the parser
    parser = argparse.ArgumentParser()
    parser.add_argument('--batch-size', type=int, default=16,
help='total batch size for all GPUs')
    parser.add_argument('--img-size', type=int, default=640,
help='train, val image size (pixels)')
    parser.add_argument('--data', type=str,
default="./data/fasterrcnn.yaml", help='data.yaml path')
    parser.add_argument('--weights', type=str, default=None,
help='train from checkpoint')
    parser.add_argument('--backbone', type=str, default='resnet50',
                        help='choose the backbone you want to use -
default: resnet50')
    parser.add_argument('--version', type=str, default='v2',
choices=['v1', 'v2'], help='Version of MaskRCNN')
    parser.add_argument('--lr', type=float, default=1e-5,
help='learning rate')
    parser.add_argument('--total_epochs', type=int, default=100,
help='total_epochs')
    parser.add_argument('--num_workers', type=int, default=4,
help='num_workers')
    parser.add_argument('--pin_memory', type=bool, default=False,
help='pin_memory')
    parser.add_argument('--logger_path', type=str, help='where you want
to log your data')
    parser.add_argument('--checkpoint_path', type=str,
default='./checkpoints',
                        help="Path where you want to checkpoint.")
    parser.add_argument('--name', type=str, default='version1',
help='name of the model you want to save')
    parser.add_argument('--project', type=str, default='Master Thesis',
help='name of the Project to save in wandb')
    parser.add_argument('--model', type=str, default='fasterrcnn',
choices=['fasterrcnn', 'deeplab', 'maskrcnn'],
                        help='the model and task you want to perform')

    # Fetch the params from the parser
    args = parser.parse_args()

    batch_size = args.batch_size  # Batch Size
    img_size = args.img_size  # Image size
    lr = args.lr  # Learning Rate
    weights = args.weights  # Check point to continue training
    backbone = args.backbone  # Check point to continue training
    version = args.version  # version of MaskRCNN you want to use

    total_epochs = args.total_epochs  # number of epochs
```

```
    num_workers = args.num_workers  # number of workers
    pin_memory = args.pin_memory  # pin memory
    model_name = args.model  # the name of the model: fastercnn,
maskrcnn, deeplab
    logger_path = args.logger_path  # where you want to save the logs
    checkpoint_path = args.checkpoint_path  # path to checkpoints
    name = args.name  # name of the projects (version)
    project = args.project  # name of the projects

    with open(args.data, 'r') as f:
        data = yaml.safe_load(f)  # data from .yaml file

    obj_cls = data['classes']  # the classes we want to work one
    relative_path = data['relative_path']  # relative path to the
dataset
    ######################################## Datasets
########################################

    bdd_train, bdd_val = get_datasets(model_name, relative_path,
obj_cls)

    print(50 * '#')
    print(f"Training Images: {len(bdd_train)}. Validation Images:
{len(bdd_val)}.")
    print(50 * '#')

    ######################################## DataLoaders
########################################

    train_dataloader, val_dataloader = get_loaders(bdd_train,
batch_size, pin_memory, num_workers)

    ######################################## Model
########################################

    # check device
    device = torch.device('cpu')
    if torch.cuda.is_available():
        device = torch.device('cuda')

    print(50 * '#')
    print(f"We are using {device}")
    print(50 * '#')

    # check model

    model = get_model(model_name, len(bdd_train.cls_to_idx), backbone,
lr, version)

    print(50 * '#')
    ModelSummary(model)  # print model summary
    print(50 * '#')

    ######################################## Training
########################################

    # Early Stopping
```

```
    early_stop_params = {
        'monitor': "val_loss",
        'patience': 5,
        'verbose': False,
        'mode': "min"
    }

    early_stop_callback = EarlyStopping(**early_stop_params)  # Early
Stopping to avoid overfitting

    # Checkpoint
    checkpoint_params = {
        'monitor': "val_loss",
        'mode': 'min',
        'every_n_train_steps': 0,
        'every_n_epochs': 1,
        'dirpath': checkpoint_path
    }

    checkpoint_callback = ModelCheckpoint(**checkpoint_params)  # Model
check

    # Loggers
    wandb_logger = WandbLogger(name=name, project=project,
log_model='all')
    csv_logger = CSVLogger(save_dir=logger_path, name=name)

    if weights is not None:
        training_params = {
            'resume_from_checkpoint': weights,
            'profiler': "simple",
            "logger": [wandb_logger, csv_logger],
            'accelerator': 'gpu',
            'devices': 1,
            'max_epochs': total_epochs,
            'callbacks': [early_stop_callback, checkpoint_callback],
        }
        fit_params = {
            'model': model,
            'train_dataloaders': train_dataloader,
            'val_dataloaders': val_dataloader,
            'ckpt_path': weights,
        }

    else:
        training_params = {
            'profiler': "simple",
            "logger": [csv_logger, wandb_logger],
            'accelerator': 'gpu',
            'devices': 1,
            'max_epochs': total_epochs,
            'callbacks': [early_stop_callback, checkpoint_callback],
        }
        fit_params = {
            'model': model,
            'train_dataloaders': train_dataloader,
            'val_dataloaders': val_dataloader,
```

```
        }

    trainer = Trainer(**training_params)
    trainer.fit(**fit_params)

    print(f"Model's best weights:
{checkpoint_callback.best_model_path}")
```

test.py

```
import json
import os.path

import matplotlib.pyplot as plt

x = [256, 384, 512, 640, 768, 896, 1024, 1152, 1280, 1408, 1536]

study = {
    'yolov7': {
        'speed': [1.6, 2.6, 3.4, 4.9, 6.3, 8.6],
        'map': [0.19, 0.29, 0.34, 0.36, 0.375, 0.376]
    },
    'yolov7x': {
        'speed': [2.1, 3.4, 5.2, 7.2, 10.2, 14],
        'map': [0.198, 0.3, 0.353, 0.37, 0.38, 0.38]
    },
    'yolov5s': {
        'speed': [1.1, 1.6, 2.2, 2.7, 3.7, 4.9],
        'map': [0.1, 0.17, 0.23, 0.27, 0.3, 0.314]
    },
    'yolov5l': {
        'speed': [2.3, 4, 6.2, 8.7, 12.1, 16.9],
        'map': [0.156, 0.25, 0.316, 0.35, 0.371, 0.378]
    },
    'yolov5x': {
        'speed': [3.5, 6.7, 10.6, 15.2, 20.6, 29.4],
        'map': [0.165, 0.262, 0.328, 0.361, 0.381, 0.387]
    },
    'yolov6': {
        'speed': [0, 0, 0, 3.54, 4.64, 6.26],
        'map': [0, 0, 0, 0.27, 0.28, 0.28]
    }
}

test_order = [
    'b71d7574-e6bc43e9.jpg',
    'c4eb5462-657f5921.jpg',
    'b6818d65-6cfd37c8.jpg',
    'b23493b1-3200de1c.jpg',
    'bbf48d08-f32427fa.jpg',
    'c7648772-cbd18adf.jpg',
    'b5465c6e-b2fb645f.jpg',
    'c4e2eba0-02c66276.jpg',
    'c3593016-1adba20d.jpg',
    'c807cb19-7e09cb11.jpg',
    'bb2c5719-38a69465.jpg',
```

```
    'c50faaad-a8463d3d.jpg',
    'c845b617-34b9e98a.jpg',
    'bba686c2-2e7b039d.jpg',
    'b57dee9d-e5bd3142.jpg',
    'bf1af4ce-dcf62242.jpg',
]

#7204a601-5bed616c.jpg
#04629c39-1d564d8a.jpg

results_rtx5000 = {
    'yolov7': {
        'speed': 12.8,
        'map': 0.36,
    },
    'yolov7x': {
        'speed': 14.3,
        'map': 0.37,
    },
    'yolov5s': {
        'speed': 11,
        'map': 0.27,
    },
    'yolov5l': {
        'speed': 19,
        'map': 0.35,
    },
    'yolov5x': {
        'speed': 24,
        'map': 0.361,
    },
    'yolov6': {
        'speed': 8,
        'map': 0.28,
    },
    'fasterrcnn': {
        'speed': 86,
        'map': 0.31,
    },
}

results_a500 = {
    'yolov7': {
        'speed': 9.3,
        'map': 0.36
    },
    'yolov7x': {
        'speed': 10.5,
        'map': 0.37
    },
    'yolov5s': {
        'speed': 11.5,
        'map': 0.27
    },
    'yolov5l': {
        'speed': 15.6,
        'map': 0.31
```

```
    },
    'yolov5x': {
        'speed': 18,
        'map': 0.361
    },
    'yolov6': {
        'speed': 7.4,
        'map': 0.27
    },
    'fasterrcnn': {
        'speed': 26.19,
        'map': 0.31
    }
}

colors = ['orange', 'blue', 'green', 'gray', 'olive', 'brown', 'red']


def plot_scatter(results, title, save_path, xlabel='speed',
ylabel='COCO AP'):
    plt.figure(figsize=(10, 7))
    for i, algo in enumerate(results):
        x = results[algo]['speed']
        y = results[algo]['map']
        plt.scatter(x, y, 200, label=algo, cmap=plt.cm.coolwarm)

    plt.legend()
    plt.xlabel(xlabel, size=20)
    plt.ylabel(ylabel, size=20)
    plt.title(title, size=20)
    plt.grid()
    plt.savefig(save_path)
    plt.show()


def plot_study(results_dic, save_path):
    plt.figure(figsize=(10, 7))
    for alog in results_dic:
        speed = results_dic[alog]['speed']
        map = [item * 100 for item in results_dic[alog]['map']]
        plt.plot(speed, map, '.-', label=alog, linewidth=2,
markersize=15)
    plt.legend()
    plt.xlabel('speed (img/ms)', size=20)
    plt.ylabel('COCO AP', size=20)
    plt.title("Inference time in ms vs COCO AP for different image
sizes", size=20)
    plt.grid()
    plt.savefig(save_path)
    plt.show()


def plot_barcharts(models, key, xlabel, ylabel, title, save_path):
    fig, ax = plt.subplots()

    x = []
    y = []
```

```
    for model in models:
        x.append(model)
        y.append(models[model][key] * 100)

    # Save the chart so we can loop through the bars below.
    bars = ax.bar(x, y, color=colors)

    # Axis formatting.
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)
    ax.spines['bottom'].set_color('#DDDDDD')
    ax.tick_params(bottom=False, left=False)
    ax.set_axisbelow(True)
    ax.yaxis.grid(True, color='#EEEEEE')
    ax.xaxis.grid(False)

    # Add text annotations to the top of the bars.
    bar_color = bars[0].get_facecolor()
    for i, bar in enumerate(bars):
        ax.text(
            bar.get_x() + bar.get_width() / 2,
            bar.get_height() + 0.3,
            round(bar.get_height(), 1),
            horizontalalignment='center',
            color=colors[i],
            weight='bold'
        )

    # Add labels and a title. Note the use of `labelpad` and `pad` to
add some
    # extra space between the text and the tick labels.
    ax.set_xlabel(xlabel, labelpad=15, color='#333333')
    ax.set_ylabel(ylabel, labelpad=15, color='#333333')
    ax.set_title(title, pad=15, color='#333333', weight='bold')

    fig.tight_layout()
    plt.savefig(save_path)


path_to_save = 'doc/images'
plot_scatter(results_a500,
             title='Inference time in ms vs COCO AP on A500 GPU',
             save_path=os.path.join(path_to_save,
'map_speed_a500.jpg'))
plot_scatter(results_rtx5000,
             title='Inference time in ms vs COCO AP on RTX5000 GPU',
             save_path=os.path.join(path_to_save,
'map_speed_rtx5000.jpg'))
plot_study(study, save_path=os.path.join(path_to_save, 'study.jpg'))
```

util.py

```
# Import Libraries
import os
import io
```

```
import random
import time
import sys
import json
from pprint import pprint
import numpy as np
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
from PIL import Image
import cv2
import torch
from torchvision import transforms
from torchmetrics.detection.mean_ap import MeanAveragePrecision
import warnings

warnings.filterwarnings("ignore")

# Colors list
color_map = [[0, 255, 0], [0, 0, 255], [255, 0, 0], [0, 255, 255],
[255, 255, 0], [255, 0, 255], [80, 70, 180],
             [250, 80, 190], [245, 145, 50], [70, 150, 250], [50, 190,
190]]


def export_map_json(mAP, json_file_name='mAP.json' , to_save_path='.')
-> None:
    """
    Function used to exported the mean average precision results to a
json file
    :param mAP: dictionary contains the mAP results
    :param json_file_name: the name of the json file
    :param to_save_path: the path where you want to save the json file
    :return: return None
    """
    with open(os.path.join(to_save_path, json_file_name), "w") as
outfile:
        json.dump(mAP, outfile)

    print(f"mAP exported to {os.path.join(to_save_path,
json_file_name)}.")



def create_cls_dic(objs) -> dict:
    """
    Function to create dictionary to map from index to class
    :param objs: list of object you want to classify (including
background)
    :return: dictionary
    """
    idx_to_cls = {}
    i = 0
    for obj in objs:
        idx_to_cls[i] = obj
        i += 1
```

```
    return idx_to_cls


def image_transform(image) -> torch.Tensor:
    """
    Function to apply image transform before feed it to the model
    :param image: image of Image.Image or numpy.array type
    :return: torch.Tensor type
    """
    t = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225], )
    ])
    return t(image)


def detection_predict(model, image, confidence_score=0.5, device=None)
-> tuple:
    """
    method to predict the bounding boxes and classes using faster rcnn
model
    :param model: the model you want to use (Faster RCNN, Mask RCNN)
    :param image: the image, it can be path to an image or array of
bytes
    :param confidence_score: confidence score used to predict
    :param device: the device (gpu, cpu)
    :return: tuple contains dictionary of the prediction and float
number
    """

    # check image type
    if isinstance(image, (bytes, bytearray)):
        image = Image.open(io.BytesIO(image))  # byte array

    elif isinstance(image, str):
        assert os.path.exists(image), "This image doesn't exists"
        try:
            image = Image.open(image)  # path to an image
        except IOError:
            print("An exception occurred, make sure you have selected
an image type.")
            sys.exit()

    elif isinstance(image, Image.Image) or isinstance(image,
np.ndarray):
        # if it is Image.Image or ndarray then pass
        pass
    else:
        # else raise an exception
        raise Exception("You must input: bytes, Image type, path for an
image, or numpy array.")
        sys.exit()

    # check device
    if device is None:
```

```
        device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')

    tensor_image = image_transform(np.array(image))  # convert image to
tensor and apply some transform
    tensor_image = tensor_image.unsqueeze(0)  # add batch dimension
    model.eval()  # put model on evaluation mode
    tensor_image = tensor_image.to(device)  # move tensor to device
    model = model.to(device)  # move model to device
    start_time = time.time()  # to measure the speed of inference
    prediction = model(tensor_image)  # predict the output
    end_time = (time.time() - start_time)  # time in sec
    fps = 1 / end_time
    prediction = prediction[0]  # remove the batch dim

    # convert the predictions to lists
    boxes = prediction['boxes'].tolist()
    labels = prediction['labels'].tolist()
    scores = prediction['scores'].tolist()

    output = {
        'boxes': [],
        'labels': [],
        'scores': [],
    }

    predict_masks = False
    if 'masks' in prediction.keys():
        masks =
(prediction['masks']>0.5).squeeze().detach().cpu().numpy()
        predict_masks = True
        output['masks'] = []

    # Filter the prediction based on the confidence score threshold
    for idx, score in enumerate(scores):
        # check score
        if score > confidence_score:
            output['boxes'].append(boxes[idx])
            output['scores'].append(scores[idx])
            output['labels'].append(labels[idx])
            if predict_masks:
                output['masks'].append(masks[idx])

    return output, fps


def get_coloured_mask(mask):
    """
    random_colour_masks
      parameters:
        - image - predicted masks
      method:
        - the masks of each predicted object is given random colour for
visualization
    """
    colours = color_map
    r = np.zeros_like(mask).astype(np.uint8)
```

```
    g = np.zeros_like(mask).astype(np.uint8)
    b = np.zeros_like(mask).astype(np.uint8)
    r[mask == 1], g[mask == 1], b[mask == 1] =
colours[random.randrange(0, 10)]
    coloured_mask = np.stack([r, g, b], axis=2)
    return coloured_mask


def display(image, prediction, save_path, idx_to_cls, rect_th=2,
text_size=0.5, text_th=2):
    """
    method to display image with bounding boxes and masks
    :param image: Image or numpy array image
    :param prediction: the dictionary that contains the preidction
    :param save_path: the path where to save the image with bounding
boxes
    :param idx_to_cls: mapping from idx to classes
    :param rect_th: thickness of bounding boxes
    :param text_size: size of text
    :param text_th: thickness of text
    :return:
    """
    if isinstance(image, Image.Image):
        image = cv2.imread(np.array(image))
    if isinstance(image, np.ndarray):
        image = cv2.imread(image)
    elif isinstance(image, str):
        assert os.path.exists(image), "This image doesn't exists"
        try:
            image = cv2.imread(image)
            # image = Image.open(image)
        except IOError:
            print("An exception occurred, make sure you have selected
an image type.")
            sys.exit()
    else:
        raise Exception("Prediction must have: boxes, scores and
labels.")
        sys.exit()

    if all(label in prediction.keys() for label in ['boxes', 'scores',
'labels']):
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        for idx, box in enumerate(prediction['boxes']):
            if 'masks' in prediction.keys():
                rgb_mask = get_coloured_mask(prediction["masks"][idx])
                image = cv2.addWeighted(image, 1, rgb_mask, 0.5, 0)
            pt1 = (int(box[0]), int(box[1]))
            pt2 = (int(box[2]), int(box[3]))
            cv2.rectangle(image, pt1, pt2,
color=color_map[prediction['labels'][idx]], thickness=rect_th)
            cv2.putText(image, idx_to_cls[prediction['labels'][idx]],
pt1, cv2.FONT_HERSHEY_SIMPLEX, text_size,
                        color_map[prediction['labels'][idx]],
                        thickness=text_th)

        plt.figure(figsize=(10, 10))
```

```
        plt.imshow(image)
        plt.xticks([])
        plt.yticks([])
        plt.savefig(save_path)
        plt.show()
        print(f"Image saved to {save_path}")

    else:
        raise Exception("You must input: Image type, path for an image,
or numpy array.")
        sys.exit()


def inference_video(model, video_source, idx_to_cls,
confidence_score=0.5, device=None, save_name='video_inference'):
    if isinstance(video_source, str):
        assert os.path.exists(video_source), f"Video does not exist in
{video_source}"
    else:
        raise Exception("You must input video path")
        sys.exit()

    # check device
    if device is None:
        device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')

    model = model.to(device)
    model.eval()

    cap = cv2.VideoCapture(video_source)
    if not cap.isOpened():
        raise Exception("Error while trying to read video. Please check
path again")
        sys.exit()

    idx_to_cls
    cls_to_idx = {}
    for idx in idx_to_cls:
        cls_to_idx[idx_to_cls[idx]] = idx

    COLORS = np.random.uniform(0, 255, size=(len(idx_to_cls), 3))

    # define codec and create VideoWriter object
    out = cv2.VideoWriter(f"inference_outputs/videos/{save_name}.mp4",
                          cv2.VideoWriter_fourcc(*'mp4v'), 30)

    frame_count = 0  # to count total frames
    total_fps = 0  # to get the final frames per second

    # read until end of video
    while cap.isOpened():
        # capture each frame of the video
        ret, frame = cap.read()
        if ret:
            image = frame.copy()
            tensor_image = image_transform(image).to(device)
```

```
            tensor_image = tensor_image.unsqueeze(0)
            start_time = time.time()
            with torch.no_grad():
                # get predictions for the current frame
                outputs = model(tensor_image)
            end_time = time.time()
            # get the current fps
            fps = 1 / (end_time - start_time)
            # add `fps` to `total_fps`
            total_fps += fps
            # increment frame count
            frame_count += 1

            # load all detection to CPU for further operations
            outputs = [{k: v.to('cpu') for k, v in t.items()} for t in
outputs]
            outputs = outputs[0]
            # carry further only if there are detected boxes
            if len(outputs['boxes']) != 0:
                boxes = outputs['boxes'].data.numpy()
                scores = outputs['scores'].data.numpy()
                # filter out boxes according to `detection_threshold`
                boxes = boxes[scores >=
confidence_score].astype(np.int32)
                draw_boxes = boxes.copy()
                # get all the predicited class names
                pred_classes = [idx_to_cls[i] for i in
outputs[0]['labels'].cpu().numpy()]
                # draw the bounding boxes and write the class name on
top of it
                for j, box in enumerate(draw_boxes):
                    class_name = pred_classes[j]
                    color = COLORS[cls_to_idx[class_name]]
                    cv2.rectangle(frame,
                                  (int(box[0]), int(box[1])),
                                  (int(box[2]), int(box[3])),
                                  color, 2)
                    cv2.putText(frame, class_name,
                                (int(box[0]), int(box[1] - 5)),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, color,
                                2, lineType=cv2.LINE_AA)
                cv2.putText(frame, f"{fps:.1f} FPS",
                            (15, 25),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0),
                            2, lineType=cv2.LINE_AA)

                cv2.imshow('image', frame)
                out.write(frame)
                # press `q` to exit
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break

        else:
            break

    # release VideoCapture()
    cap.release()
```

```
    # close all frames and video windows
    cv2.destroyAllWindows()

    # calculate and print the average FPS
    avg_fps = total_fps / frame_count
    print(f"Average FPS: {avg_fps:.3f}")


# YOLO evaluation

def yolo_to_pascal(x, y, w, h, width=1280, height=720) -> tuple:
    """
    Function to convert YOLO format (xywh) to Pascal voc format (xyxy)
    :param x: float number, x center
    :param y: float number, y center
    :param w: float width of the bounding box
    :param h: float height of the bounding box
    :param width: float width of the image
    :param height: float width of the image
    :return: tuple contains xmin, ymin, xmax, ymax
    """
    xmax = int((x * width) + (w * width) / 2.0)
    xmin = int((x * width) - (w * width) / 2.0)
    ymax = int((y * height) + (h * height) / 2.0)
    ymin = int((y * height) - (h * height) / 2.0)
    return (xmin, ymin, xmax, ymax)


def prepare_gt(file_path) -> dict:
    """
    Function to prepare th ground truth annotation from the file to use
them in evaluation
    :param file_path: the path for the ground truth annotations
    :return: dictionary contains the bounding boxes and labels
    """
    output = {
        "boxes": [],
        "labels": [],
    }
    with open(file_path, 'r') as f:
        for line in f:
            line = line.split()
            label = int(line[0])
            box = [float(b) for b in line[1:]]
            xmin, ymin, xmax, ymax = yolo_to_pascal(box[0], box[1],
box[2], box[3])
            output['boxes'].append([xmin, ymin, xmax, ymax])
            output['labels'].append(label)

    output['boxes'] = torch.tensor(output['boxes'])
    output['labels'] = torch.tensor(output['labels'])

    return output


def prepare_prediction(file_path) -> dict:
    """
```

```
    Function to prepare the predicted bounding boxes to use them in
evaluation
    :param file_path: the path for the prediction files
    :return: dictionary contains the bounding boxes, scores and labels
    """
    output = {
        "boxes": [],
        "scores": [],
        "labels": [],
    }
    with open(file_path, 'r') as f:
        for line in f:
            line = line.split()
            label = int(line[0])
            box = [float(b) for b in line[1:-1]]
            score = float(line[-1])
            xmin, ymin, xmax, ymax = yolo_to_pascal(box[0], box[1],
box[2], box[3])
            output['boxes'].append([xmin, ymin, xmax, ymax])
            output['scores'].append(score)
            output['labels'].append(label)

    output['boxes'] = torch.tensor(output['boxes'])
    output['scores'] = torch.tensor(output['scores'])
    output['labels'] = torch.tensor(output['labels'])
    return output


def yolo_evaluation(prediction_path, gt_path) -> dict:
    """
    Function to evaluate the performance of the YOLO algorithm
    :param prediction_path: the path for the prediction folder that
contains the prediction files
    :param gt_path: the path for the ground truth folder that contains
the ground truth files
    :return: dictionary contains the mean average precision
    """
    assert os.path.exists(prediction_path), f"{prediction_path} folder
not found"
    assert os.path.exists(gt_path), f"{gt_path} folder not found"

    pred_files = os.listdir(prediction_path)

    metric = MeanAveragePrecision(box_format='xyxy',
class_metrics=True)

    for file in tqdm(pred_files):
        pre = prepare_prediction(os.path.join(prediction_path, file))
        gt = prepare_gt(os.path.join(gt_path, file))
        metric.update([pre], [gt])

    mAP = metric.compute()

    return mAP
```