SAPIENZA
UNIVERSITÀ DI ROMA

An adversarial approach to improve Instance Segmentation in Automotive

Dipartipento di Ingegneria Informatica, Automatica e Gestionale Antonio Ruberti

Corso di Laurea Magistrale in Artificial Intelligence and Robotics

Candidate

Damiano Zappia

ID number 1870958

Thesis Advisor

Prof. Fiora Pirri

Academic Year 2019/2020

Thesis defended on 25 May 2021
in front of a Board of Examiners composed by:

Prof. Alessandro De Luca (chairman)
Prof. Fiora Pirri
Prof. Massimo Mecella
Prof. Aristidis Anagnostopoulos
Prof. Andrea Cristofaro
Prof. Francesca Cuomo
Prof. Leonardo Querzoni

An adversarial approach to improve Instance Segmentation in Automotive
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: damianozappia@gmail.com

Abstract

If we think about the future that awaits us, there are certainly lot of technologies, that will revolutionize our life, making reality several things that only few years ago or still today we imagine as science-fiction.
One of these revolutions is for sure authonomous driving. Autonomous driving is a technology that allows vehicles like cars or trucks to self-drive in streets without any (or almost) help from humans.
For this task they combine a series of things like sensors and software, to control and navigate the vehicle. At the basis of this control there are without doubt the vision capabilities, where an accurate analysis of what the vehicle see is crucial to take the appropriate decision. In this context comes the Instance Segmentation, a computer vision technology that is capable of identifying objects , detecting and delineating each distinct object of interest appearing in an image.
Obviously the technology is far from being perfect currently, for a system to be employed in real life driving context, there is the absolutely need to make almost 100% of right actions.
Vision systems still suffer of some problems, like the correct detection of each subject in the scene, or the low level of accuracy when such system runs in real time as its clearly needed when it's used on a vehicle.
The purpose of this thesis is to demonstrate how Instance Segmentation can be further improved allowing the vehicle to count on a strong and reliable algorithm, that can help into choosing the right action while driving. To improve such technique, I decided to use an Adversarial Approach, i.e. to put the Instance Segmentation network to compete with another network, in a Generative Adversarial Network framework, where the two networks are forced to compete and thus to improve each other.
The motivation of this choice is that even if the state of the art methods for instance segmentation actually perform very well, they still suffer in some situations such as crowded scenes or identification of non common objects. The adversarial approach tries to contrast these issues and make instance segmentation more accurate.

Contents

1 Introduction **1**
1.1 Instance Segmentation 1
1.2 Generative Adversarial Networks 3
1.3 Instance Segmentation and GANs in autonomous Driving 3

2 Instance Segmentation and GANs: state of the art **6**
2.1 Instance Segmentation 6
2.1.1 Background 6
2.1.2 Instance Segmentation Techniques 7
2.1.3 State of the Art techniques 9
2.2 Generative Adversarial Networks 14
2.2.1 Overview 14
2.2.2 GANs state of the art techniques 15

3 Preliminaries **21**
3.1 Instance Segmentation with Mask-RCNN 21
3.1.1 Anchor Boxes 23
3.1.2 Intersection over Union and Non Max. Suppression 23
3.1.3 Masks generation 23
3.2 GANs - In detail look 24
3.2.1 What are GANs 24
3.2.2 GANs model 24
3.2.3 Conditional GANs 28
3.3 GAN Mask-RCNN 31

4 Method **32**
4.1 Idea 32
4.2 Method 32
4.2.1 Generator Network 33
4.2.2 Discriminator network 33
4.3 Loss functions 33
4.3.1 GAN loss function 33
4.3.2 Discriminator Loss Function 34
4.3.3 Generator Loss 36
4.3.4 Alternative Loss: Least Square 37
4.4 Model architecture 37

5 Implementation Details **39**
5.1 Development Environment . . . 39
5.2 Dataset Definition . . . 39
5.3 Generator Network details . . . 40
5.3.1 Model loading . . . 40
5.4 Discriminator Network Details . . . 41
5.4.1 Mask discriminator . . . 41
5.4.2 Weights initialization and optimizer . . . 42
5.5 Network parameters . . . 43
5.6 Training Procedure . . . 43
5.6.1 Image processing . . . 43
5.6.2 Training Loop . . . 45
5.7 Further Refinements . . . 46

6 Experimental setup and results **47**
6.1 Experimental setting . . . 47
6.2 Evaluation metrics . . . 48
6.3 Datasets . . . 49
6.3.1 PennFudan dataset . . . 49
6.3.2 CityScapes Dataset . . . 50
6.4 Prepare the baseline . . . 51
6.5 PennFudan dataset Results . . . 52
6.5.1 Quantitative results . . . 52
6.5.2 Qualitative results . . . 53
6.6 CityScapes dataset results . . . 55
6.6.1 Quantitative Results . . . 55
6.6.2 Qualitative results . . . 56

7 Conclusions and future works **59**

A Appendix: relevant code for implementation **60**

Bibliography **68**

Chapter 1

Introduction

Think about a world where traveling in a robotaxy, or asking your car to take you from Rome to Milan while you take a nap or read your emails, would be completely normal.
Even if for some of us it could seem the premise of a sci-fi movie, this is the direction in which we are moving, thanks to the last developments in **Machine Learning**, and especially in **Computer Vision**[1], that allowed the birth of technologies such as autonomous driving.
Autonomous driving is one of those revolutions that will embrace all us in the upcoming future, allowing to live and move in a safer and smarter world, where **vehicles will no longer require human intervention to function**.
To make this revolution possible and support the development of such technology, we need extremely powerful vision algorithms. Algorithms capable of reliable detection and understanding of every single instance inside an image.
At the time of this writing, there are several computer vision algorithms that are capable of giving very good results, but are still far from perfection. According to ASIRT [1], *approximately 1.35 million people die in road crashes each year; on average 3,700 people lose their lives every day on the roads.* Such numbers could be considerably lowered if autonomous driving technology was spread and ready to be largely used, avoiding the vehicle to make errors during the navigation, and preventing a lot of distractions from many peoples while driving.

The purpose of this Master's Thesis work is to use an adversarial approach based on Generative Adversarial Networks (**GANs**), to improve instance level semantic segmentation, in the automotive field. Before to proceed further it's worth to spend some words on what instance segmentation and Generative Adversarial Networks are, and why they are important in Autonomous Driving.

1.1 Instance Segmentation

Instance Segmentation, as stated in the abstract, it **is a computer vision task that requires the prediction of object instances and creation of their pixel-level masks**. This technique extends the object detection standard methods, adding a mask to the identified objects. To better clarify the concept, let's make a visual example in order to understand what Instance Segmentation means and why

[1] Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do.

it is different from standard object detection or semantic segmentation. Standard object detection, as shown in the fig. 1.1, is a computer vision technique that allows to identify and locate objects inside images or video. Each identified object is enclosed in a box region and classified.

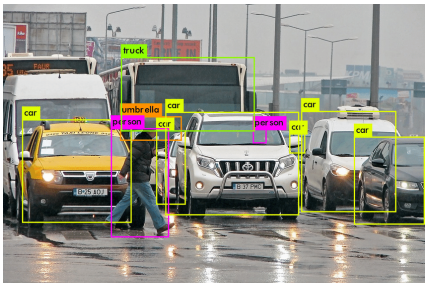
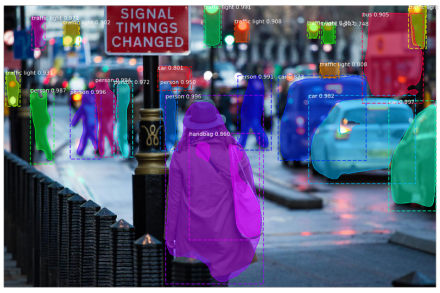


Figure 1.1. Object Detection Example

Image segmentation (either semantic or instance) differs from this approach in the fact that is scans the image at pixel level, "assigning the image" pixel by pixel to a certain class, that is drawing a new image pixel-by-pixel, basically transforming the input image in another more detailed (at information level) one. In fig 1.2 there are examples of this techniques.

(a) semantic segmentation



(b) Instance segmentation

Figure 1.2. Comparison between instance and semantic segmentation

As you can notice, in semantic segmentation each pixel in the image is linked to a particular class label, identified with one specific color. These segments are used then to find and understand interactions between various objects. While in instance segmentation, the difference is that multiple objects of the same class are treated as individual / separate entities, and indeed identified with different colors.
From this you can easily understand that **instance segmentation is a way more difficult task with respect to semantic**, **because each object in the scene has to be well recognized and segmented individually**, and in situations like a crowd of people, while semantic segmentation can easily group them in one region classifying them as *people*, instance segmentation has to identify and divide each single person of the crowd, and **apply a specific and precise mask** on it, i.e. an **ad-hoc pixel level classification**.
Now that this is clarified, we can give an introduction on Generative Adversarial Networks, or better known as **GANs** [2].

1.2 Generative Adversarial Networks

Generative Adversarial Networks are a particular type of neural networks. Before to step into the details of the thesis, it's better to give an overview of them.
They are an approach to generative modeling using deep learning methods.
We define the learning *deep* because the layers that constitute the network are many.
Generative modeling is an **unsupervised learning** technique, that is a learning technique that **does not require any human supervision to work**, consisting in automatically discovering and learning the regularities of patterns in the input data (the data that are provided to the network), such that the model can be used to generate in output new samples that could have been reasonably drawn from the original dataset, i.e. the dataset from where input data are taken.

This approach was presented in 2014 by Ian Goodfellow and other researchers at the University of Montreal in [2], and have been **defined by Yann LeCun** (Director of AI Research at Facebook) **"the most interesting idea in the last 10 years in ML"**. It is a particularly clever approach because frames the problem as a supervised learning problem, with the use of **two sub-models**: the **generator** model that is trained to generate new plausible examples, and the **discriminator** model that is responsible to distinguish the real samples (taken from the dataset), from the fake samples (generated by the generator model). The supervised framing is given by the fact that each of the two sub-models tries to supervise the other, indeed the two **models are trained together in an adversarial zero-sum game**, in which no one of the two networks is going to win, when the discriminator is fooled about half the time, we have obtained a generator that is capable of generating real-like samples.

1.3 Instance Segmentation and GANs in autonomous Driving

Autonomous Driving as already stated, is a technology that allows a vehicle to move without any human help in a real (and likely complex) driving scenario. At nowadays there are not yet commercial applications of such advanced technology, but the research is doing huge steps and we are day by day closer to have such innovation really available.
Of course a car (or a vehicle in general) to be capable of self driving, needs to be supported by a huge set of sensors and actuators to take the proper measurements and actions in each situation, but almost zero uncertainty, since it operates in a potentially dangerous scenario. Thinking about that, we can all likely agree on the fact that **vision capabilities** are on the basis of autonomous driving, and maybe **are the key feature in order to have a self-driving vehicle**.
From this analysis it comes clear that the computer vision algorithm of an autonomous vehicle has to be really powerful, capable of identifying every common object in a driving scenario, and act properly in consequence of every detection.
Here comes the purpose of this thesis' work, a strong algorithm has to be trained on a huge set of different and complete driving scenes, i.e. a dataset of street scenes. The more the dataset is complex, complete and diversified, the more accurate and strong performing the algorithm will be.
The drawback of this requirements, is that **building such dataset is a tremendously hard work**, where thousand of images or videos have to be captured, and annotated. Such work requires countless time and resources and is one of the moti-

vations behind the fact that we still don't have a commercial spread autonomous driving platform.
To face this problem, the purpose of this thesis is to present an approach helping to develop computer vision algorithms that are capable of generating new instance segmentation street scene images, ready to be used in order to implement a self-driving vision algorithm, of course without the need of a human work in the creation of corresponding annotations for each image.

This is possible on the basis of what has been previously introduced. The purpose is to **take a top performing instance segmentation algorithm**, which we will talk about later, **and improve it further** with the use of GANs.
Indeed, even the top performing instance segmentation algorithms, suffer in several situations, in particular crowded street scenes, where the shapes of many people are overlying inside an image, or more in scenes where for instance there are street advertisements on road sides, with a person inside, causing the detection of person by the algorithm that perhaps behave wrongly due to this erroneous detection. Other examples can be again the street advertisements, that are sometimes confused with *bus* or *train* classes, causing of course a **wrong and potentially dangerous behaviour if the algorithm was really implanted on a car**.
GANs are thought as a solution for this problem, because the approach of this work is to use them to improve the behaviour and the performances of the instance segmentation algorithm, trying to fight all the problems such those described above.

The idea is to include the instance segmentation algorithm itself inside a GAN framework, where it will do the part of the generator, i.e. the network responsible of generating new plausible samples, and on the other side we will have a powerful discriminator that tries to distinguish between the predicted results for each input image, and the ground truth [1] results.
This approach will allow the network, to fight all the common errors in instance segmentation, being forced each time it does an error, to correct its behaviour and do a better prediction the next time. But is not all fun and games of course, because the **difficulty of GANs is that they are extremely hard to make to converge**, indeed as stated before, a GAN is based on a game teoretic scenario in which the two networks compete each other trying to prevail.
The fact is that this tendency to prevail is not so rare because it can very often happen that one of the two networks is more powerful than the other, and thus always "wins the game", and if this difference of performance between the two networks is too big, the "looser" network won't be able to correct its behaviour in time and will always go to get worse, meaning that we lost the brittle equilibrium that we was searching, with consequence that the network, in this case the Instance Segmentation network, will get disadvantages instead of advantages from this approach, resulting in a worsening of performances.

With all this in mind, we are ready to understand the next steps of this work, in particular:

- In **Chapter 2** we are going to present the state of the art of Instance Segmentation Methods and Generative Adversarial Networks.
- In **Chapter 3** we will present some preliminaries, i.e. a deepening on the

[1] "Ground truth" may be seen as a conceptual term relative to the knowledge of the truth concerning a specific question. It is the ideal expected result.

techniques that we will need for the development, going in the details of what has been introduced in this chapter.

- In **Chapter 4** the method is presented, together with the idea that supported this work, and the motivation of choices taken for the development.
- In **Chapter 5** we focus on the implementation details, how the code is structured, the training procedure, and an explanation of the steps for the network training.
- In **Chapter 6** instead experiments and results are shown, presenting the datasets used for this work and the achievements obtained.
- Finally, in **Chapter 7** we focus on conclusions and future works.

Chapter 2

Instance Segmentation and GANs: state of the art

Instance segmentation is defined as the technique of simultaneously solving the problem of object detection as well as that of semantic segmentation.
Indeed, if object detection analyzing an image provides the class of each image object, together with the location of the image object inside the scene, semantic segmentation further refines this detection, going to fine label each pixel of the image, assigning it according to the class to which the object enclosing that pixel belong.
The combination of this two strategies gives us instance segmentation, where we have different labels for each object inside the image, even if two object belong to the same class, they are labeled with different pixels.

Generative adversarial networks (GANs) are algorithmic architectures that use two neural networks, putting one against the other (thus the *adversarial*) in order **to generate new, synthetic instances of data that can pass for real data**. They are used widely in image generation, video generation and voice generation. But their application is not only limited in the generation of new samples "from scratch", their structure is in fact so powerful and adaptable that they can be used in the translation of image domain, or in general into the modification of data following some "guidelines" to obtained the desired output, what is called conditional Generative Adversarial Networks.

In the following of this chapter, we are going to see the evolution of the Instance Segmentation, presenting the techniques that are at this time, the best solution in the accomplishment of this task. In the final part of the chapter, we will instead focus on GANs, explaining their origin and the most powerful applications nowadays.

2.1 Instance Segmentation

2.1.1 Background

As was stated, instance segmentation can be defined as the task of finding simultaneous solution to object detection and semantic segmentation, a clear visual example of this can be seen in fig. 2.1
This technique has become one of the most challenging, important and complex topics in the area of Computer Vision research. Instance segmentation at nowadays

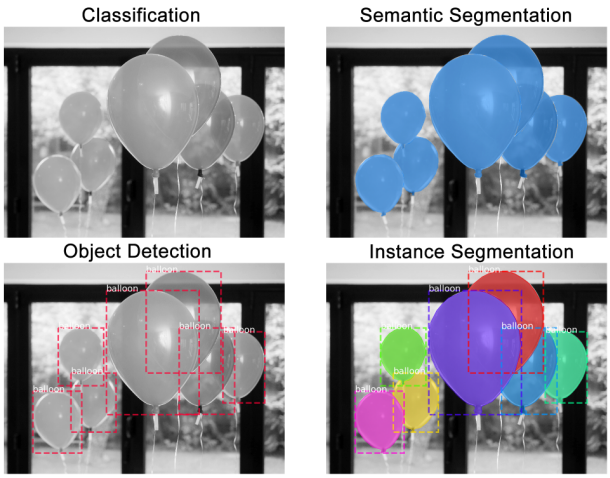


Figure 2.1. Differences between Object detection and Semantic / Instance segmentation

aims at helping the development of fields like robotics, autonomous driving or public surveillance. The born of the technique is dated in concomitance of the advent of Convolutional Neural Networks, and the spreading of Deep Learning, around years 2011-2012, when many instance segmentation frameworks were proposed, with **Mask-RCNN** [15] that is for sure one of the most famous and top performing still today.
Of course to support the development of such frameworks a series of specific datasets were built, first and foremost the *Microsoft's Common Objects in Context* or *COCO dataset* [3], with more than 300k images, and over 2.5 millions of labeled instances, but also datasets like CityScapes [4] or ImageNet [5] and many others.

2.1.2 Instance Segmentation Techniques

Before to introduce what are and what have been the best Instance Segmentation Networks, it's important to spent some words on the techniques at their base.

Approach based on classification of mask proposals

One of the first examples of instance segmentation was shown by Haritan et al. [6]. Their technique first was about generation of mask proposals, and then the classification of these proposals. After that work, the approach with the classification of mask proposals was widely adopted. In fig.2.2 you can see a general framework for this technique, reported by Hafiz et al.[7].

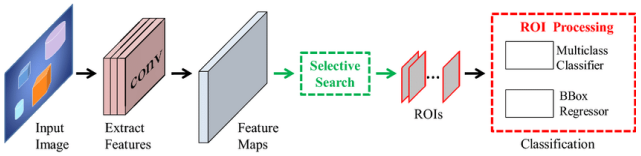


Figure 2.2. general framework for classification of mask proposals

When Deep Learning become popular, this technique was replaced with others

with a more efficient structure, most of them having as a basis the popular *R-CNN* Network [8]. This network is capable of analyzing and image, and extract from it regions proposals, that are then used to compute features using a large convolutional neural network, and classified with a class-specific linear SVMs.
Despite this, all the new techniques of Instance Segmentation based on RCNN and similar, suffered from some problems, like a multistage pipeline for detection, that was slow and difficult to optimize, and also features have to be extracted from each proposal in every image, causing again a slow process. This problems were addressed by *Fast R-CNN* [13] and *Faster R-CNN* [14].

Approach based on detection followed by segmentation

This kind of approach is still today one of the most effective, and is represented in its maximum expression by Mask-RCNN [15], that will be deepen in the later chapters, but consists into a refinement of Faster-RCNN [14] network, where the network analyze the image extracting features, then the feature maps are processed to detect objects and boxes, and these are then processed by a multiclass classifier and a bounding box regressor, obtaining as output the mask of the object. The structure is shown in fig.2.3, taken from Bienias et al. [16]

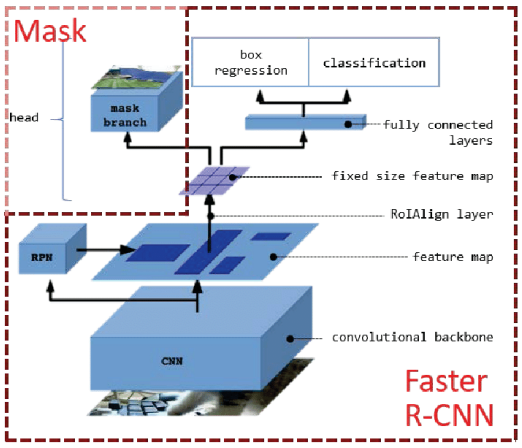


Figure 2.3. The structure of the Mask R-CNN architecture. The model basically extends the Faster-RCNN by adding a mask prediction module as second branch.

These approaches based on detection followed by segmentation are as said particularly effective, even though they suffer from some of the weaknesses presented in the previous kind of approaches, due to their pipeline structure.

Approach based on labeling pixels followed by clustering

A different approach from the ones presented above, is taken from the semantic segmentation strategy, and consists into grouping pixels into object instances, using cluster algorithms, a representative example was presented in [9]; each pixel of the image is labeled as a category. The framework for this technique is shown in fig.2.4 The weaknesses in this case are that the computational resources required are more, because as in semantic segmentation each pixel has to be labeled

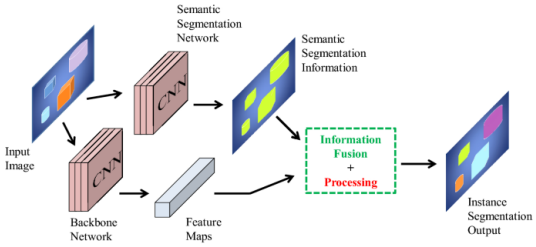


Figure 2.4. The general structure labeling pixels approach

Approach based on dense sliding window method

The last kind of approach is typical of more deep learning based models, which uses massively CNNs, and works by sliding dense widows on the image to extract features, that are then analyzed to obtain feature maps, the feature maps are then further processed and from them objects masks and scores for each mask are obtained, in a framework as the one in fig 2.5

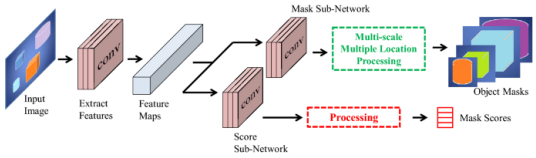


Figure 2.5. General structure of dense sliding windows approach. In this approach dense windows are slided over the input image to extract features, that are processed to obtain masks.

Notably implementations of that approach are for sure DeepMask [10], Instance-FCN [11] and TensorMask [12].

2.1.3 State of the Art techniques

There is currently a large variety of techniques dealing with instance segmentation, but some of them are particularly notable and effective. A common base is to have as backbone a very powerful object detection network, therefore we are going to see a more detailed overview of them.

R-CNN

This network was presented by Girshick et al. [8] in 2014, during the "gold period" of convolutional neural networks, they were among the first to explore and exploit the power of CNNs for instance segmentation.
In fig. 2.6 you can see a graphical representation of the network.

RCNN reached very good results in terms of object detection, but still suffers from some drawbacks, like as said before, the pipelined structure that slows down the process and make it complex, since each individual stage has to be separately trained. Also training is slow because due to the CNN features extraction, object proposal is slow down.
For these reasons other techniques have born, that will be now presented.

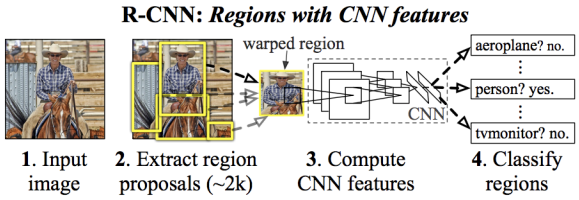


Figure 2.6. RCNN network representation. the procedure is divided in several steps, where first regions are extracted from image, then each region is processed by CNN, and its features are used to do object detection.

Fast RCNN

Fast-RCNN [13], from the same authors of RCNN, was the first network that addressed some problems of RCNN. The approach is similar, but instead of feeding the region proposal to the CNN, the input image is fed to the CNN to generate a convolutional feature map, from which identify regions of proposal and warp them into squares, then by using a RoI (Regions of Interest) pooling layer, the region proposals are reshaped into a fixed size so that it can be fed into a fully connected layer. Finally, from the RoI feature vector, a softmax layer is used to predict the class of the proposed region and also the offset values for the bounding box.
The motivation why Fast-RCNN is actually faster than RCNN, is because there is no more the necessity to fed the 2000 regions proposal to the CNN every time, but the convolution operation is done only once per image, generating a feature map for it.
The result is a training time really a lot faster w.r.t RCNN, around by a factor of 10.

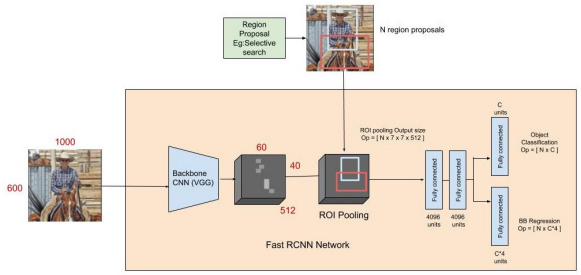


Figure 2.7. Fast-RCNN Network summary. This time instead of feeding the region proposals in CNN as was with RCNN, input image itself is fed into a CNN, used to generate a convolutional feature map from where to use RoI pooling.

Faster-RCNN

Faster-RCNN [14] is a further refinement of Fast-RCNN, in fact, despite being faster than RCNN, Fast-RCNN still relies on external regions proposal, that is basically it's bottleneck. For this reason, Faster RCNN model was proposed; similar to Fast-RCNN the image is provided as input to a convolutional network which provides a convolutional feature map, but instead of using selective search algorithm on the feature map to identify the region proposals, a separate CNN is used to predict the region proposals.

These predicted regions proposals, are reshaped using a RoI pooling layer, that is then used to classify the image withing the proposed region, and predict the offset values for the bounding boxes.
With these improvements, Faster-RCNN is really fast indeed, so fast that it can be used for real time object detection.

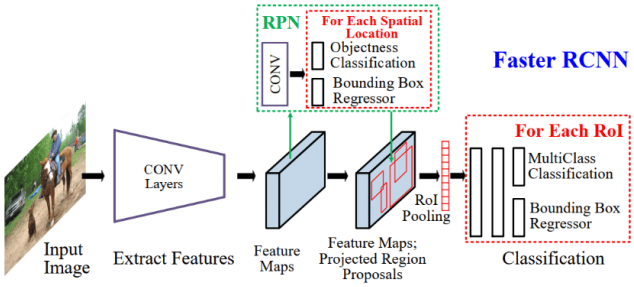


Figure 2.8. Faster-RCNN Network summary. In this case instead of using selective search algorithm on the feature map obtained from image, a separate CNN is used to perform region proposals

Mask-RCNN

Mask-RCNN [15] is presented by the authors as a relatively simple and flexible model for instance segmentation. this is achieved by object detection together with generation of high quality masks. Mask-RCNN is based on the structure of Faster-RCNN, adding a branch for mask predictions in parallel of object detection. This model is easy to train, because add a small computational load to the Faster-RCNN model. The good point of this technique is that is particularly effective. It reached first position in the 3 COCO 2016 challenges: instance-segmentation, detection of bounding boxes, detection of person key points.
Mask-RCNN outperformed the other state of the art models on each COCO task of 2016. Still today is one of the best models for instance segmentation, even of course suffers from some issues, like difficulties into correctly segment instances in crowded situations, a problem that is common to basically all the object detection/segmentation networks. In fig 2.9 is represented the structure of the network.

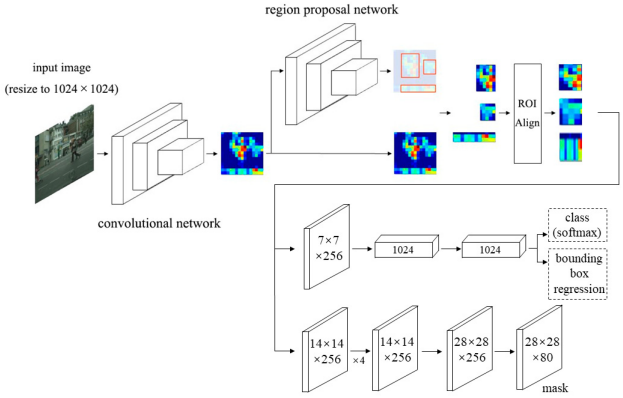


Figure 2.9. Mask-RCNN Network summary. Structure is similar to Faster-RCNN but this time we have a second branch (on the bottom of image) responsible for mask prediction.

YOLO

YOLO [18], which acronym means You Look Only Once, is one of the fastest object detection algorithms, at the basis on many instance segmentation networks that we will see in the next lines.
All the object detection algorithms before YOLO, used regions to localize the object within the image, i.e. the network doesn't look at the entire image but only at those regions that are likely to contain an object. In YOLO instead, a single CNN analyse the whole image predicting the bounding boxes and the class probability for them. This is done by taking the image and split it into a grid of size N $\times$ N.
Whithin the grid, m bounding boxes are taken, and for each bounding box the network outputs a probability and offset values for the boxes. The boxes with the probability over a certain threshold are selected to locate the object within the image. The benefits of YOLO are that it is extremely fast as object detector, but this speed is balanced with a non optimal accuracy, expecially in detecting small objects inside the scene.

PANet

PANet stands for Path Aggregation Network for Instance Segmentation [19], and even if it takes as baseline the Fast-RCNN structure, it enhances the latter a lot. The framework of the network is shown in fig. 2.10

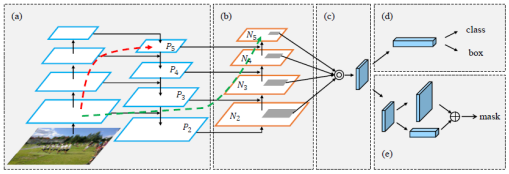


Figure 2.10. PANet framework. Information flow is schematized in steps, where particular attention is given to bottom-up path augmentation. Even in these case there are two branches, one for classes and boxes prediction, and the other for mask prediction.

The key idea proposed in this case, is that the way in which information flows or propagate in deep neural networks is very important , thus this flow of information has to bee improved, in particular the flow of the framework for the instance segmentation task. Following that idea, the authors improved the hierarchy of the deep network by the use of specific signals related to localization in the lower layers. This process, is called bottom-up path-augmentation, and the benefits are the shorter information paths between the lower levels and the features at the top of the deep network. In addition to that, there is also a technique called adaptive feature pooling, thanks to this, the relevant information in every layer of features, is able to flow to the subsequent sub-network responsible for proposals generation. Then there are two branches, one responsible of predicting classes and boxes, the other responsible to predict masks.

YOLACT

YOLACT [20] stands for You Only Look At Coefficients, and is a fast and simple fully convolutional model for instance segmentation. The weakness of many top performing instance segmentation models, is that they are too heavy to run in real time, and even if possible they still run at very low fps rate. YOLACT is designed

just to accomplish this goal, and when presented was the fastest real-time instance segmentation network, being capable of running a 33 fps on an Nvidia Titan XP GPU.
These results are obtained by the execution of the image segmentation task dividing it into two different subtasks, the first is the generation of prototype masks, and the second is the prediction of mask coefficients for each instance mask. Then, the linear combination prototype mask and the mask coefficient gives the instance mask

SOLO

SOLO [21], standing for Segmenting Objects by Location, is a simple and powerful network for instance segmentation. The idea behind this network is resumed in the following quote by the authors:
If two object instances in the input image has exactly the same shape and position, they are the the same instance. Any two different instances either have different position or shape. And as shape is hard to describe in general we approximate shape with size.
That is, instance category is defined by location and size, a totally different approach from all the previous models we have seen. In particular, the location is classified by center position, that is determined dividing the input image into a grid of S $\times$ S cells.
The size is determined by assigning objects of different sizes to different levels of a Feature Pyramid Network (FPN) [49]. Basically, for each pixel, SOLO only needs to perform two simple pixel-level classification problems, as in semantic segmentation.

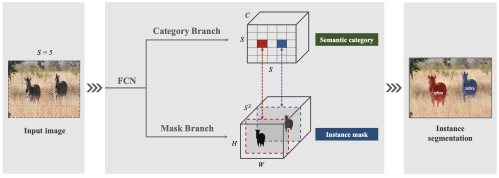


Figure 2.11. SOLO architecture, instance segmentation is divided as two subtasks: category prediction and instance mask generation problems. An input image is divided into a uniform grids. If the center of an object falls into a grid cell, that grid cell is responsible for predicting the semantic category (top) and masks of instances (bottom)

Regarding the masks, each positive grid cell will also generate the corresponsing instance mask, the masks are encoded as the third dimension of a 3D output tensor. At the end there is an instance segmentation result for each grid, because the category prediction and the corresponding mask are naturally associated to their reference gridd cell. In fig.2.11 you can see the network architecture.

CenterMask

CenterMask is a network proposed by Lee et al. [22], is an anchor free instance segmentation network. It works by adding a spatial attention-guided mask branch to the anchor free one stage object detector. As the authors remark:
Plugged into the FCOS object detector, the SAG-Mask branch predicts a segmentation mask on each detected box with the spatial attention map that helps to focus on informative pixels and suppress noise.
The aim of this network was to fill the gap between the good level of accuracy but

low real time performances of Mask-RCNN and the lower level of accuracy but high speed of YOLACT. Centermask is as stated, an anchor free, one stage, instance segmentation network. It has a one stage anchor free object detection branch, called FCOS, to which is added a spatial attention guided mask branch (fig. 2.12), that taking the predicted boxes from FCOS detector, and predicts segmentation masks for each region of interest.

The spatial attention module (SAM) in the SAG-Mask helps the mask branch to

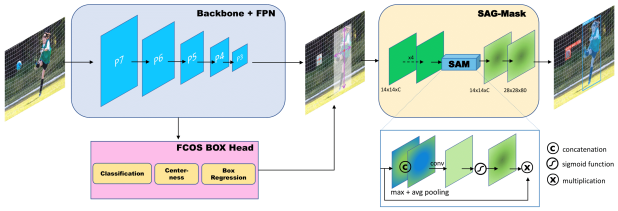


Figure 2.12. CenterMask architecture. P3 to P7 denote the feature map in feature pyramid of backbone network. Using the features from the backbone, FCOS predicts bounding boxes. Spatial Attention-Guided Mask (SAG-Mask) predicts segmentation mask inside of the each detected box with Spatial Attention Module (SAM) helping to focus on the informative pixels but also suppress the noise

focus on meaningful pixels and suppressing uninformative ones. When extracting features on each RoI for mask prediction, each RoI pooling should be assigned considering the RoI scales, and for this reason authors propose a scale-adaptive RoI assignment function that considers the input scale and is therefore a more suitable one-stage object detector.
Also a different backbone is used, that is *VoVNetV2* (based on [23]), having better performances and speed with respect to the Mask-RCNN's backbone ResNet.

2.2 Generative Adversarial Networks

The GANs architecture, as already said, was introduced in 2014 by Goodfellow et al. [2], and **determined a breakthrough in the Machine Learning community**, leading to a series of new possibilities and improvements in lot of tasks. In the following lines we will see the state of the art applications for this kind of network, understanding why they are so powerful and useful.

2.2.1 Overview

As discussed in the previous chapter, GANs are a deep learning typology on networks based on the adversarial approach. Basically a GAN is composed by two neural networks, a generator and a discriminator.
In the standard approach, the **generator is an "unconvolutional" neural network**, that is a network responsible of taking random generated samples of noise, and apply to them inverse convolution operation (or better called TransConv), to reshape and resize this noise, making it as much as possible close and similar to a real sample of data. The **discriminator** on the other hand, **is a convolutional neural network**, responsible of taking in input a data sample, and understand if it is taken from the real data or instead generated from the generator network. To cite the "NIPS 2016 Tutorial" [24] :

"We can think of the generator as being like a counterfeiter, trying to make fake money, and the discriminator as being like police, trying to allow legitimate money, and catch counterfeit money. To succeed in this game, the counterfeiter must learn to make money that is indistinguishable from genuine money, and the generator network must learn to create samples that are drawn from the same distribution as the training data."

As is easy understandable from these words, the task of the discriminator is to become powerful enough to be able to discriminate correctly the samples received, while the generator has to become always better and better to generate fake samples that are indistinguishable from reality.
In this way the two models, i.e. the two networks, are put to compete each other, in a **zero sum adversarial game**.
We say zero sum game because, **when the discriminator correctly classifies an instance, it is rewarded**, and we have no change's application to his weights, **while the generator is penalized** with large updates on it's parameter's weights, because it failed into fool the discriminator. On the other situation, **if the generator is able to fool the discriminator, there is not weights update** for it, **weights** that **are changed into the discriminator** network, that failed into correctly classify the instance.
In the end there should be, **ideally, no network** that **outperforms the other**, arriving to have a generator that able to generate perfect replicas of real samples, resulting in a discriminator that cannot tell the difference between real / fake samples. In this case there is an equilibrium on the GAN, with none of the two networks that receives big updates on the weight. This case, is **an ideal case, very difficult to reach in reality**, especially because **finding an equilibrium between two different networks that compete is far from easy**, but fortunately there is no need to reach such equilibrium to obtain a good generator, the goal of a GAN is indeed to obtain a generator capable of creating admissible samples, and this can happen even when there is not perfect equilibrium.

2.2.2 GANs state of the art techniques

As was stated, GANs are getting ubiquitous in machine learning development in the last period, being used to generate photorealistic images, change facial expressions, create computer game scenes, visualize designs, and even generate awesome-inspiring artworks.
Thus we are going now to review some of the best implementations of such networks.

Generating samples for image datasets

Application was described in the original paper by Goodfellow et al. in 2014 [44], in this case the GAN is used to generate new plausible examples for the given dataset. In particular, in the paper was shown the ability of GANs to generate new samples of the MNIST handwritten digits dataset, the CIFAR-10 small object ohotograph dataset, and the Toronto faces dataset. In fig. 2.13 you can see the results of the GAN generation after the training.

Generating photographs of human faces

This application is an evolution of the previous, presented by Karras, et al. in [25], and is actually the demonstration of how powerful such networks are. The generated

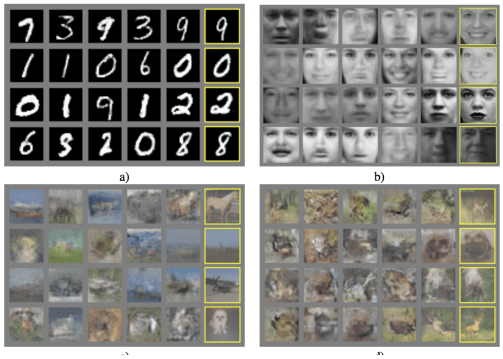


Figure 2.13. Example of generated sample images from GANs compared to the corresponding datasets

samples (fig2.14) are so realistic that is actually impossible to distinguish them from real people, even for a human being.

Figure 2.14. Generated samples from photorealistic GAN generated faces

The same method is used to generate photorealistic images of objects and animals, like cars, horses, cats etc... Also with this approach, in citeanimefaces, the authors show how GANs are capable of generating totally realistic anime faces (i.e. Japanese comic book characters).

Image-to-Image translation

This category involves many applications, and have at the basis the conditional GANs, that is an extention of the classical approach, where in addition to the input given to the generator, that we remember is a random vector from the latent space, is provided an additional input, that has the role to conditionate the data generation. This additional input, could be a class value, such as male or female in the generation of photographs of people, or a digit, in the case of generating images of handwritten digits.
One of the most famous implementation of conditional GANs is for sure the Pix2Pix

network from the paper [27], from which you can see some examples of generated data in fig. 2.15. The particularity of the conditional approach, and in this case in the Pix2Pix GAN, is that the network changes the loss function so that the generated image results both plausible in the context of target domain, and as a translation of the input image.

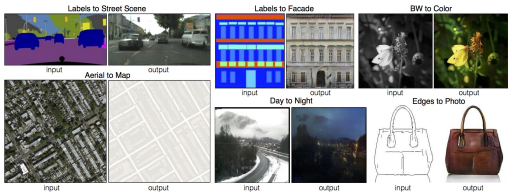


Figure 2.15. Examples of Pix2Pix GAN generated images

Another interesting implementation of Conditional GAN that is worth to mention is the Cycle GAN, introduced in [28] by Zhu et al.
The revolutionary idea here was that, as is easy to imagine, training a model for image-to-image translation requires in most cases a large dataset of paired examples, dataset that can be difficult and expensive to prepare, and in some cases impossible, like when we want to have paired images of photographs and respective paintings of a dead artist. Cycle GAN address this problem, by allowing training of image-to-image translation models without paired examples. How is that possible? The Cycle GAN apply an extension to the standard GAN architecture, adding one extra Generator network and one extra discriminator network, going to train simultaneously two generators and two discriminators.
Thanks to this structure, we have that one generator takes images from the first domain as input, and produce outputs of the second domain, that become input for the second generator, that generates images of the first domain again. The discriminator models are then used to understand how good are these images. With this approach the network is capable of generating plausible instances for both domains, but still not capable of generating translation of images between domains. Here comes the last extension of Cycle GAN in the architecture, the *cycle consistency*, where the idea is that the image output by the first generator is used as input for the second, and the second generator's output should match the original image. this has to be valid even if reverse, the output of the second generator can be given as input to the first, and the output of the first generator should match the input of the second. This idea, that could seem confusing, is taken from machine translation, where as the authors say:
... we exploit the property that translation should be "cycle consistent", in the sense that if we translate, e.g., a sentence from English to French, and then translate it back from French to English, we should arrive back at the original sentence
The CycleGAN encourages cycle consistency by adding an additional loss to measure the difference between the generated output of the second generator and the original image, and the reverse. Graphical example of this network's results is n fig. 2.16.

Text-to-Image translation

This is one of the most astonishing applications of GANs, Zhang et al. in their paper [29], demonstrate the power of GANs in generating realistic images from textual description. To better understand what this means we can look at fig. 2.17.

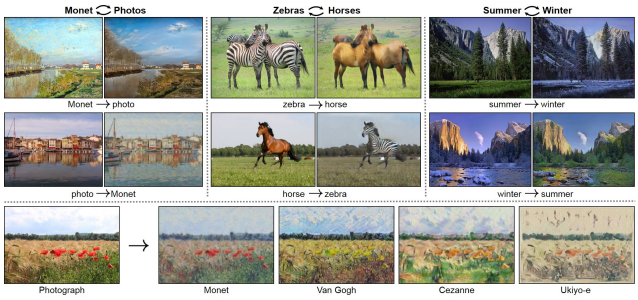


Figure 2.16. Cycle-GAN images examples

Figure 2.17. StackGAN generated samples, first row shows images after stage I while second row are images after stage II

How does it work? Basically the stage one of the network sketches the primitive shape and basic colors of the object based on the given text description, obtaining a low resolution image output that matches the text input description. The second stage then, takes the stage-one's result and generate from that an high resolution image, with, as the authors remarks, "photo realistic details".

Face Aging

Presented by Antipov et al. in [30], it consists in a GAN that is capable to generate photographs of faces with different apparent ages, from younger to older. It is again an application of conditional GANs, and to make possible to apply age effects on a face, it's necessary to create a map from an input image x with label y, to a latent vector z, that has to be constructed. This latent vector, is a variable that is not directly observable during training, but is a kind of "hidden" connection between input and output.
This mapping is obtained using an encoder, that is a neural network that approximate this inverse mapping. The encoder is trained to minimize the Euclidean distances between estimated latent vector z_0 and the ground truth latent vector. But the authors found that although the approximation z_0 result in visually good face reconstructions, the identity of the original image is lost in about 50% of cases. Thus, they proposed a novel "Identity-Preseving" approach to improve this z_0. The solution is to use a face recognition neural network called FR, that embed the input face image x as FR(x) and to embed the reconstructed one x_bar as $FR(x\_bar)$. Then, go to minimize the euclidean distance between these two embeddings rather than directly between x and x_bar, make it possible to maintain the identities of

original face images.
With this same approach is possible to apply other effefct to the subjects in the images, like hair styles, glasses application, and so on and so fourth.

Super Resolution GANs

Presented by Leding et al. in [31], the SRGAN model is capable of generating output images with higher, sometimes very much higher, resolution corresponding to the given input image.
The training of this network is done by taking an high resolution image, downsampling it in a lower resolution, and then give that to the generator, that has to learn how to upsample the received input. Thus the generator has to understand how to transform LR image in HR image, while the discriminator each time has to understand if the received HR image, is a real HR image, or instead is a fake HR image. An example of result for this GAN is in fig. 2.18

Figure 2.18. Super resolution GAN example (left: input image, right: resulting image)

Video Prediction

It could seem strange, but GANs are capable of predicting the future, up to a second of prediction, as it was demonstrated in [32]. The authors propose an approach based on GANs to understanding object motion and scene dynamics, the model in this case has to understand how a certain scene transforms with time.

Figure 2.19. The Deepfake of president Obama, by Alexander Amini, MIT's professor

Deepfake

This application is one of those that made the GANs so popular in the last years, and allows to change the existing scenes in videos to create the new and fake but realistic-looking scenes. One of the most prominent works is presented in [33]. Basically we have as input to the generator network some video or audio (the original data), that has to be "transformed" going to superimpose some fake input, producing fake but realistic output samples. This technology is so advanced nowadays that is very often impossible to detect the fake even for a human, as you can see in fig. 2.19. That specific deepfake was generated by Amini using Canny AI [34].

Text-to-Speech GAN

Not all GANs produce images, we have an example by Yang et al. [35], where they used GANs to produce synthesized speech from text input. The architecture is composed of a conditional feed-forward generator producing raw speech audio, and an ensemble of discriminators which operate on random windows of different sizes. The discriminators analyze the audio both in terms of general realism, as well as how well the audio corresponds to the utterance that should be pronounced.

We have also a notable application in [46], where DeepMind team explored raw waveform generation using GANs composed of a conditional generator for producing raw speech audio and an ensemble of discriminators for analyzing the audio.
In GAN-TTS the input of the generator is a sequence of human speech that present linguistic features and pitch information. The generator in this case has to learn how to convert the linguistic features and pitch information into raw audio.

Chapter 3

Preliminaries

The aim of this work, is to improve instance segmentation methods with adversarial approach.
Before to proceed with the method explanation and the implementation details, that will be reviewed in the next chapters, it's absolutely necessary to explain in the details the theory behind the method.

This chapter will be divided in three subparts. In a first section, more brief, I will introduce the Instance Segmentation method of Mask-RCNN, while in the second part, we will see an accurate analisys of GANs, especially on those related to the method used for this thesis.
In the last section instead, I introduce the paper *GAN Mask R-CNN:Instance semantic segmentation benefits from generativeadversarial networks* by Quang H. Le et al. [39], that is used as idea to develop this thesis'work.

3.1 Instance Segmentation with Mask-RCNN

As was introduced in the previous chapter, **Mask-RCNN is a powerful instance segmentation method**, based on the detection followed by segmentation approach. We are now going to see how does it works, the method at its basis, and why is so powerful.

At the basis of its functioning, we have the RCNN algorithm introduced in chapter 2. RCNN uses a particular technique called *selective search* [17]. Basically, instead of trying to classify a huge number of regions, a fixed number of proposals for each image is established, more specifically, 2000 regions. These regions are generated by the selective search method. The 2000 candidate region proposals are warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output.
Basically the CNN works as a features extractor, and then the extracted features are fed to a SVM that classifies the presence of the object within that candidate region proposal. In addition to that, the algorithm also predicts four values that are the offset values used to increase the precision of the bounding box, this helps in cases like when some instance even if detected is not entirely inside the region proposal, thus the boxes offsets help to adjust the bounding box of the region proposal.

Mask-RCNN further improves Faster-RCNN as we have seen, adding a branch for

the prediction of mask instances. In details, instance segmentation with this method works in the following way.

The model is divided in two parts:

- **Region proposal network** (RPN) to proposes candidate object bounding boxes.
- **Binary mask classifier** to generate mask for every class

In fig. 3.1 is represented the two branches structure, and it's easy to understand the differences between Mask and Faster RCNN.

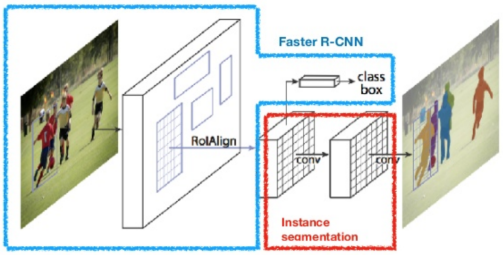


Figure 3.1. Mask-RCNN model, source: [15]

The steps of this model are the following:

1. Taken an image, it is run through the CNN to generate the feature maps
2. Region Proposal Network(RPN) uses a CNN to generate the multiple Region of Interest(RoI) using a lightweight binary classifier. It does this using 9 anchors boxes over the image. The classifier returns object/no-object scores. Non Max suppression (that will be explained in a while) is applied to Anchors with high objectness score
3. The RoI Align network outputs multiple bounding boxes rather than a single definite one and warp them into a fixed dimension
4. Warped features are then fed into fully connected layers to make classification using softmax and boundary box prediction is further refined using the regression model
5. Warped features are also fed into Mask classifier, which consists of two CNN's to output a binary mask for each RoI. Mask Classifier allows the network to generate masks for every class without competition among classes

From fig. 3.1, we can go more in depth to fig 3.2, where the expanded perspective gives a better understanding of the above described steps.
As authors report in the paper, during training the loss is composed by multiple elements for each RoI, and follows the formula:

$$L = L_{cls} + Lbox + Lmask$$

In particular, the loss on mask is the key element in this model, and have the following formulation:

$$L_{mask} = -\frac{1}{m^2} \sum_{1 \leq i,j \leq m} [y_{ij} log \hat{y}_{ij}^k + (1 - y_{ij} log(1 - \hat{y}_{ij}^k)] \tag{3.1}$$

Where y_{ij} is the label of a cell (i, j) in the true mask for the region of size m × m. $\hat{y}_{ij}^k$ s the predicted value of the same cell in the mask learned for the ground-truth class k.

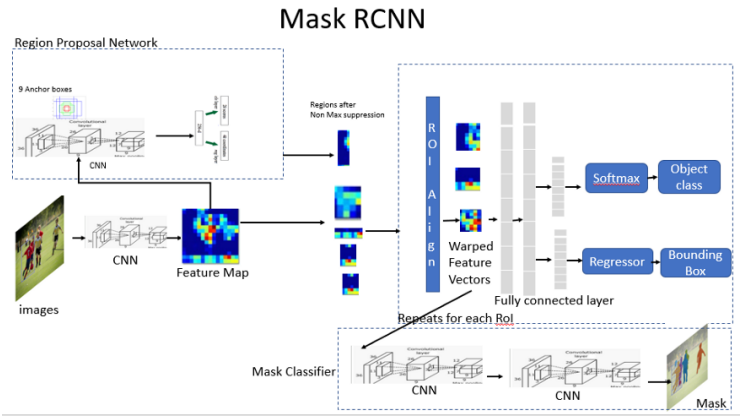


Figure 3.2. Mask-RCNN complete architecture. Image is fed in CNN to generate feature maps, then RPN uses it to generate RoI, that are used by RoI Align Network to output multiple bounding boxes. Warped features are then fed into fully connected layers to make classification, and also fed in Mask classifier to generate masks for every class

Each single element of the model has a key role into determining the model effectiveness, therefore it's useful to have a brief look for each one.

3.1.1 Anchor Boxes

Mask-RCNN uses **anchor boxes** to detects multiple objects inside the scene. These anchor boxes are bounding boxes of a given height and width that are used to make thousands of predictions for each image. Then the final object detection is obtained removing the anchor boxes that are not relevant, such as the background, and those that have low level of confidence score.

3.1.2 Intersection over Union and Non Max. Suppression

Intersection over Union (**IoU**) is a measure that computes the intersection over the union of two bounding boxes, that are the ground truth and the predicted ones. In anchor selection, all anchors that are below 0.5 of IoU are discarded, by **Non Max Suppression**, that has the role to eliminate those boxes, and also to pick for each object just the anchor bounding box with the highest level of confidence, suppressing all the others.

3.1.3 Masks generation

At the second stage of the model, another neural network takes the proposed regions, assign them to several specific areas of the feature map level, and generates object classes, bounding boxes and masks.

This procedure is done without anchors, but thanks to a technique called ROIAlign, that locates the relevant areas of a feature map, and generates a mask for each object at pixel level.

3.2 GANs - In detail look

We have seen in the previous chapter many state of the art techniques that make use of GANs, starting to understand their power and their functioning. Now it's time to have an in detail look at them, explaining how does the GAN framework work, the math behind, and why they can be helpful in this work.

3.2.1 What are GANs

Up to now we introduced GANs, talked about their benefits, but without explaining how and why they are different from the other machine learning paradigms. We will now see the idea of generative models, and how it is different from the supervised learning and discriminative models paradigm.

Supervised vs Unsupervised approach

In a tipical machine learning problem, we have a model that is used to make predictions, and for this we have a training dataset comprised of multiple inputs (X) and corresponding outputs (Y). The model is trained showing examples of inputs and teaching how to predict the right output.
This form of learning is called **supervised** because there is an outcome for each input that the model is expected to match, i.e. its output is compared with that outcome every time, to see if it did god or bad.
Tipically classification and regression tasks are supervised learning problems.

In opposition to that paradigm, we have the **unsupervised** approach, where the model just receives input X but not any output, in this case the model is trained extracting and summarizing the patterns in the input data. Thus the goal of the model is to find *interesting patterns*, to cite Murphy in his book [36], in the data. The difference is therefore that in this second paradigm there is no correction for the model. GANs are exactly an example of this approach.

Discriminative and Generative models

We have explained the supervised approach, and told that classification task is an example of that. Classification is also referred to a **discriminative model**. Such name means that the model must discriminate examples of input variables (images, text, audio, etc..) between classes.

On the other hand, unsupervised models that have to summarize the distribution on input variables finding a pattern, could be able to create (or generate) new examples of that distribution. We have examples of generative models in Latent Dirichlet Allocation (LDA) [37], or Gaussian Mixture Model (GMM) [38].

3.2.2 GANs model

GANs are a model architecture to train generative models. They are an example of unsupervised and generative approach, that will be now explained in detail.

To understand the architecture, we can cite the *Deep Learning* book [41] of Goodfellow et al., were they say the following about GANs:

Generative adversarial networks are based on a game theoretic scenario in which the generator network must compete against an adversary. The generator network directly produces samples. Its adversary, the discriminator network, attempts to distinguish between samples drawn from the training data and samples drawn from the generator

Thus it's necessary to understand the generator and discriminator models individually.

Generator Model

In classical GAN approach, generator model takes a fixed length random vector as input, and generates as output a sample in the target domain, that could be for example the faces dataset that we have seen before in 2. Usually this random vector is drawn from a Gaussian distribution. This means that the input vector is just an array of random numbers, not yet an image or anything similar to the output target. After the training, points in the multidimensional vector space will corresponds to points in the target domain, with a complex representation of the data distribution. The vector space is called **latent space**, or vector space of the **latent variables**. To cite again the [41], *A latent variable is a random variable that we cannot observe directly.* In other words, we can say that a latent space provides a compression or high-level concepts of the observed raw data such as the input data distribution. As regards GANs, the generator model gives a meaning to this latent space, allowing to new points drawn from this space to be provided to the generator model as input and used to generate new and different output examples. The block diagram of generator model is in fig. 3.3.

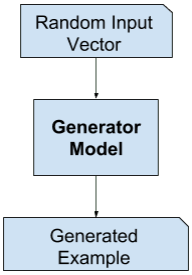


Figure 3.3. GAN generator model block scheme

Discriminator model

The discriminator model is the second element of a GAN, taking inputs from the problem domain, that can be real or generated, and returning a binary prediction that states if the input received is indeed real or was generated.
After the training procedure the discriminator model is discarded, because we are interested in the generator model that has hopefully learned how to generate very realistic samples of the problem domain. The block scheme of discriminator network is in fig. 3.4

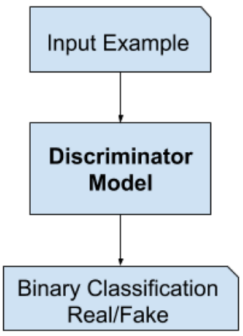


Figure 3.4. GAN generator model block scheme

Generator and Discriminator together

The GANs framework with generator and discriminator network that are connected together, is in fig. 3.5

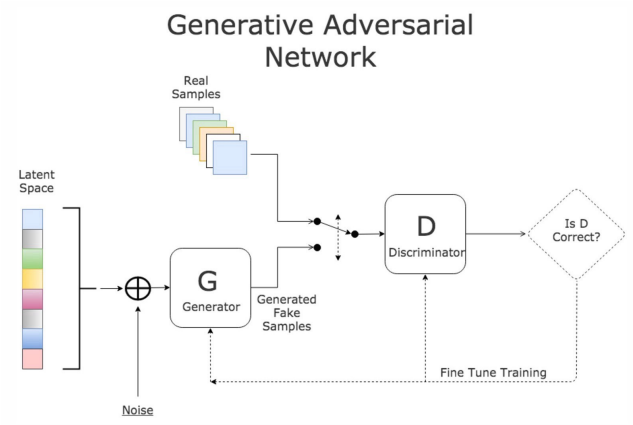


Figure 3.5. GANs framework

The two models, generator and discriminator, compete each other, the simplest way to formulate an adversarial training is a zero sum game, where a function $v(\theta^{(g)}, \theta^{(d)})$, called **payoff function** in game theory, is the payoff of the discriminator, and it's opposite, $-v(\theta^{(g)}, \theta^{(d)})$ is the generator's payoff.
This payoff function, is the expected absolute error of the discriminator, and has the following expression:

$$v(\theta^{(g)}, \theta^{(d)} = E(G, D) = \mathbb{E}_{x \sim p_{data}} log d(x) + \mathbb{E}_{x \sim p_{model}} log(1 - d(x)) \tag{3.2}$$

The function guides the discriminator into learning to correctly classify samples as real or fake. On the same time the generator is guided to fool the discriminator learning to generate realistic samples.
This reasoning means we want to maximize the error with the generator and minimize the error with the discriminator, in formulas:

$$\max_{\mathbf{G}}(\min_{\mathbf{D}} E(G, D)) \tag{3.3}$$

Now, the problem is that in cases where this function is convex, the procedure is guaranteed to converge and is asymptotically consistent. But the reality is that in GANs generator and discriminator are neural networks and the function $max_v(g, d)$ is not convex. This non convergence issue is not easy and is one of the main problems while training GANs, causing underfitting [1]
To better understand the problem, we can take the example from [41]: imagine we have a function $v(a, b) = ab$, where one player controls a with cost function ab and the other player controls b, and has cost function -ab. It's easy to understand that each player goes to reduce it's own cost at the expenses of the other, this the overall value of the function cannot decrease asymptotically, but rather stay in a stable circular orbit.
This means, in other words, that there is no local minima for such a function, but instead a saddle point (fig. 3.6).

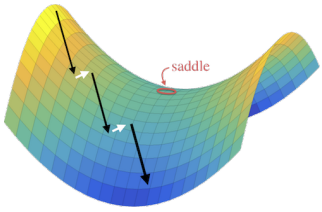


Figure 3.6. Saddle point representation

GANs stabilization remains still an open problem, but there are situations were they perform very well with parameters carefully selected, as in DCGAN [44], or in situations where, as in conditional GANs [45], the generator learn to sample from a distribution $p(x|y)$ rather than $p(x)$, these conditional GANs that have been introduced in previous chapter, deserve a detailed explanation, needed to fully understand the next chapters.

Benefits of GANs

There are a series of reasons why GANs are an extremely interesting, important and used kind of neural network. In [24], Goodfellow highlights a some of them; before to proceed presenting the state of the art techniques that make use of GANs, it's worth to cite some of the most interesting and important approaches:

- **Data Augmentation**: data augmentation is a technique used especially in computer vision, consisting into the creation of new artificial, but absolutely plausible, data from the input domain, having the benefits of obtaining better performing models, increasing the model's skills together with the regularization effect, and reducing the generalization error. Classical techniques of data augmentation are crop, flip, resize and so on. Thanks to GANs, that are capable of generate domain specific data that are similar to the real ones, the data augmentation techniques get lot of improvement.

[1] Underfitting is the case where the model has " not learned enough" from the training data, resulting in low generalization and unreliable predictions

- **Image super resolution**: It consists in the ability to generate high resolution images version of the input images, and is today used in a series of commercial solutions.
- **creating art**: Thanks to GANs is possible to create new artistic images, sketches, paintings and more, giving a series of such art images to the discriminator, and therefore obtaining a generator network capable of literally draw these data from noise, as was explained in the preceding section
- **image to image translation**: this technique is particularly powerful and became popular on the last months, it consists into the ability of GANs to translate images across domains, examples are translation of photos from days to night, or changing the face of a person inside an image, aging it or making it younger.

3.2.3 Conditional GANs

In Chapter 2 several state of the art methods that make use of conditional GANs have been presented, and, to cite the *Conditional Adversarial Nets* paper [45], they say:

> Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information y. y could be any kind of auxiliary information, such as class labels or data from other modalities. We can perform the conditioning by feeding y into the both the discriminator and generator as [an] additional input layer

To better understand their functioning, we will see an overview of the math behind and then focus on the implementation details of some of the state of the art techniques presented before.

Conditional Adversarial Nets

Generative adversarial networks can be extended as we told, feeding both the discriminator and generator with an additional information $\mathbf{y}$, that could be a class label bus also any kind of data.
Once we have this $\mathbf{y}$, in the generator the input noise $p_z(z)$ is combined with y, in a joint hidden representation.
In the discriminator, the input $\mathbf{x}$ and the additional variable $\mathbf{y}$ are presented as inputs of the discriminative function. With this paradigm, the structure of the conditional adversarial network is the one in fig. 3.7
The objective function of the two-players minimax game will transform to:

$$\min_{\mathbf{G}} \max_{\mathbf{D}} V(D,G) = \mathbb{E}_{x\sim p_{data}}[logD(x|y) + \mathbb{E}_{z\sim p_z} log(1 - D(G(x|y)))] \tag{3.4}$$

Famous applications

Among the state of the art methods presented in 2, there ore some that are useful to be deepen in order to understand the powerfulness of cGANs.

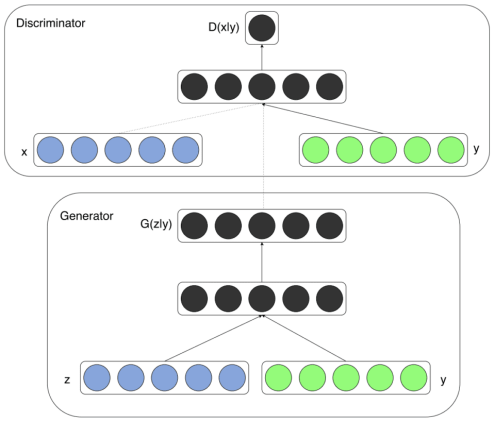


Figure 3.7. Structure of conditional adversarial network

Super Res GANs The loss function that guides this procedure, is called *perceptual loss*, and is a linear combination of two losses:

$$l^{SR} = l_x^{SR} + 10^{-3} l_{Gen} SR$$

To understand the meaning of this formula, we have to focus on his two terms. The first is the content loss, based on pixel-wise differences between the activation layers of a VGG-10 model. The VGG loss is defined as the euclidean distance between the feature representation of a reconstructed image and the reference image. The second term is the adversarial loss, that serves as a check on the generated image. This loss favors the generation ok natural looking images.

Image-to-Image translation GANs

Text to image translation GANs Focusing on the implementation details, the process is particularly complex, indeed one of the main difficulties in such situations is that the network could have no idea in what to start "drawing", and also obviously given a text description for a certain scene, there should be a variety of images that all match the given description.
For this reason, the problem is decomposed in two sub-problems, stage 1 and stage 2 aforementioned. How is possible to make a computer able to generate images from text, since it is not capable of understanding words? The answer is that it can represents words in terms of something understandable, that is called *text embedding*. Basically, differently from classical GAN formulation, there is not just noise taken from the generator, but also an additional information, that is represented as c, it is in this case the phrase used to describe the image. This phrase is used by Generator to take inspiration on how to draw the image, and also by Discriminator to check if the image produced matches that description.
Once the sketch image is generated, we proceed with stage 2, responsible to fill the details. The stage 2 is conditioned both on the text and on the low-resolution image output by the first level and outputs a high-resolution image.
To go further in the details of this network could be very hard to explain in this section, and out of the objective of this chapter, but the implementation details

make use of several interesting concepts of GANs, making clear how many variables and reasoning could be involved in such a "apparently simple" task. The network architecture is summarized in fig. 3.8.

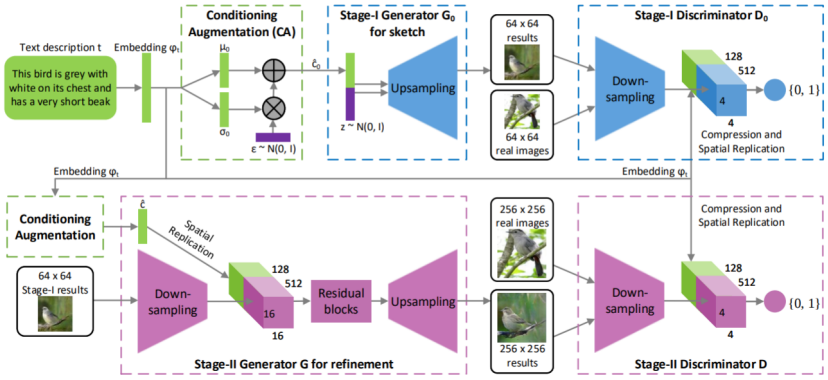


Figure 3.8. StackGAN architecture

DeepFake generation Going a bit into the details, this technique is possible once we have a bunch of images (intended as videos also) of both real subject, the one that has to be transformed, and target subject. Then a convolutional neural network is used to process the image and compress the picture volume (width x height x depth) into the one-dimensional vector. This is what is known as **encoder**, because it encodes information of a picture inside a single vector.
So suppose we have many images of subject A and subject B, and we want to create a deepfake of subject A. We need one encoder, and two decoders, that as it's easy to imagine is responsible to obtain a picture from an encoded single vector. During training, we use the same encoder that encodes all the relevant information from images into a one-dimensional vector. Then one decoder learns how to decode the one-dimensional vector of subject's A images to recreate them and the other one will be used to recreate subject's B from the one-dimensional vector of his images. At this point we have obtained the mapping shown in fig. 3.9

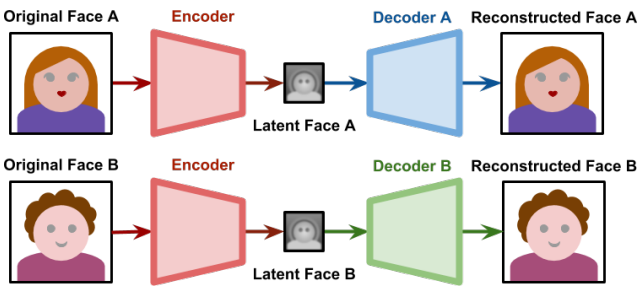


Figure 3.9. Deepfake encoder-decoder structure

After the training, decoder B is applied to latent face A, and viceversa decoder A is applied to latent face B, and is the network is correctly trained, we will obtain surprisingly realistic fake samples of the subjects.

The Deepfake technology is already really advanced, and you probably already faced some implementation of it, even if not realizing that. For instance, is thanks to deepfake is the Paul Walker was present in Fast and Furious 7 movie final scene, after his tragic death in 2013. But all that glitters is not gold, and as was easy to imagine, there are several of non-ethical and bad usages that this GAN technology lead, from creation of fake videos of politicians and institutions to make them say what a malicious person wants. Such situations are of course some cases

from what we all have to be careful and aware.

3.3 GAN Mask-RCNN

This thesis' work has been inspired by the paper *GAN Mask-RCNN: Instance semantic segmentation benefits from generative adversarial networks* [39].
The authors presented an approach for instance segmentation that addresses the problem of predicting masks as a GAN game framework.
Their idea, is to have Mask-RCNN as mask generator, and then optimize this generation using the minmax strategy of GANs, where the discriminator networks tries to to distinguish predicted masks from the real ground truth masks.

Their contribution is articulated in three points:

- New GAN loss to make the Mask-RCNN model trainable in the GAN framework.
- Instead of using images as input for the discriminator, they use feature mapsof the corresponding images
- GAN training on object bounding boxes using Precise RoI Pooling [40], to extract a feature map from the boxes and give it to a box discriminator network.

Thanks to these structure, they obtained performance gain in object detection and in mask segmentation tasks with respect to the standard Mask-RCNN algorithm.

In this work I take inspiration from this idea, designing a more simple GAN framework to improve instance segmentation, as it is explained in next chapter 4

Chapter 4

Method

In the previous chapter instance segmentation model of Mask-RCNN has been introduced, and a detail discussion on how GANs work has been provided, understanding why are they so powerful. With this in mind, it's time to introduce the idea that brought to this thesis' work, explaining the method chosen to develop the model, the math behind it, and the motivation of the choices.

4.1 Idea

Mask-RCNN is a very powerful instance segmentation network, but it **is not perfect**, suffering among the other things in a lack of precision when asked to predict masks and instances in crowded scenes, but also in the prediction of crisp-boundaries for some instances.
The idea behind this thesis is therefore to improve instance segmentation, and in this case Mask-RCNN, with a GAN based approach. The inspiration comes from *GAN Mask-RCNN* paper [39], but here instead of having such complex structure, with multiple discriminators and feature maps processing instead of images, I use a much simpler approach in which in addition to the Mask-RCNN generator there is a simple convolutional neural network, represented by the Discriminator, that with a proper loss and training has the role to improve the mask generations.

We can indeed think of a GAN framework, where the role of generator is exercised by Mask-RCNN, while on the other hand there is a discriminator network, that has to distinguish if for a given image, the resulting masks, come from the real distribution or have been predicted by Mask-RCNN generator. In the next section we are going to see therefore how this is realized, and the structure of this particular type of GAN.

4.2 Method

We said that in this GAN, the **Mask-RCNN model has the role of the generator** network, while the **discriminator network is** designed from scratch and **responsible of prediction for real / generated samples**.
The method is simple, first of all a pretrained Mask-RCNN model is selected, then this model is fine-tuned over the target dataset. Once this operation is completed, the obtained model represents the Generator network of the GAN, it is ready to generate meaningful predictions, and it can be put in competition with the discriminator,

with the purpose of improving mask generation capabilities.

About the discriminator, the main focus of this work is to improve the mask prediction for each received instance, correctly understanding if a given mask comes from the ground truth data or has been generated. The discriminator network is therefore a CNN responsible of returning an outcome representing the probability that the received mask is real, if the outcome is high the network has high confidence on the fact that the mask is real, while on the contrary a low value means the network believes the received mask is generated.
Now we will have an in detail look on the structure of the GAN network, and of the loss functions that guides the learning procedure. In depth details of the generator and discriminator networks are instead given in chapter 5.

4.2.1 Generator Network

The generator network is as mentioned above, Mask-RCNN, in particular a Mask-RCNN model pre-trained on COCO dataset [3]. This means that the model is not starting to learn from scratch, but its weight have already been modified on COCO dataset. This technique allows to start with a model that is already capable of doing predictions, and has just to be *fine tuned* on the desired dataset.
This particular Mask-RCNN model is pretrained on COCO and has ResNet50 [48] as backbone network, with FPN architecture [49].

4.2.2 Discriminator network

The mask discriminator has the role of determining if a given mask is real, coming from ground truth, or fake, predicted by Mask-RCNN generator.
Thus the role of this discriminator is to guide the generator model into improve the mask generation, returning outputs that according to the loss function (explained in the next section), penalize both discriminator and generator causing change of the weights and corresponding improvement.

4.3 Loss functions

In machine learning, the **learning process is guided by a loss function**, for each prediction by the network the loss function computes the loss, that is how much the prediction was wrong with respect to ground truth. This allows the network to understand by receiving a feedback on their error that changes the parameters weights and directs the learning process in the right direction.

In the case of a GAN network, there are multiple losses compatible with each task, and because the learning procedure is highly unstable and difficult to make to converge, no loss is guaranteed to return optimal results. Let's have a look of the general loss function formulation for the GAN and then a focus on the singular loss functions for the discriminator and the generator network.

4.3.1 GAN loss function

The GAN architecture introduced by Goodfellow et al. in [2], has a classical loss formulation that has been explained in chapter 3, called **minimax GAN loss**, and a second formulation that is known as **non-saturating GAN loss**.

minimax GAN Loss

In this formulation, as we saw before, we have the minimax simultaneous optimization of the generator and discriminator model. The min and max terms refer to the minimization of the generator loss and the maximization of the discriminator loss, thus:

$$minmax(D, G)$$

The discriminator tries to maximize the average of the log probabilities of real images and the log of the inverse probability of fake images, with a formulation presented below in the discriminator section.
In this approach the generator learns to generate samples that have a low probability of being fake. The downside is that, to cite the authors of *Generative Adversarial Networks* paper:

"this kind of loss does not provide sufficient gradients for the generator to learn well."

The motivation is soon said, at the beginning of learning process, the discriminator network can reject samples from generator (that is still poor) with high probability because they are very different from real data. This brings to the other loss formulation.

Non-saturating GAN loss

With this loss formulation, while the loss of discriminator remains the same, there is a modification in the generator loss, that consists in the generator that has to maximise the log of discriminator probability tofor generated samples, **instead of minimizing the log of the inverted discriminator**.
In formulas, instead of $min\ log(1 - D(G(z))$ we have $max\ log(D(G(z))$.

The change is really ingenious, indeed if before the generator tried to **minimize the probability of generated samples to be predicted as fake**, here its tries to **maximise the probability of samples to be predicted as real**. The result is an higher gradients update especially in early learning.

4.3.2 Discriminator Loss Function

In the case of this implementation, the first loss function selected is the **Binary Cross Entropy** loss (BCE). This loss function is one of the most simple but yet effective losses when dealing with GANs. The motivation is that the learning procedure is determined by the discriminator network, that in this case is a **binary classifier**, it has indeed to classify if the received sample is real or fake.
Before to give the mathematical formula of this function, it's important to understand the meaning behind its use.
We said we have two discriminators, the reasoning is exactly the same for both: given an input, which is the **probability** of that input to be real (or fake)? Ideally real samples will have a probability 1 (of being reals), and the fake samples have a probability 0 (of being reals of course).
The reasoning is straightforward, the discriminator model will predict for each received input the probability of that input to be real, assigning as we said, a value between 0 and 1. The purpose of the binary cross entropy loss function will be to

assign high value to bad predictions and low values to right predictions.
Thus mathematical formulation is:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p(y_i)) + (1-y_i) log(1-p(y_i)) \tag{4.1}$$

The y_i in the formula is the label, that we assign as explained 1 for reals and 0 for predictions, and $p(y_i)$ is the predicted probability of the received input of being real.

Therefore, for each real label we add a log(p(y)), i.e. the log probability of that instance of being real, and for each fake label, we add a log probability for that instance of being not real. But as you can notice the formula includes *log* terms for predictions instead of their stand alone values.
The motivation is that we need to have a non linear mechanism of penalization for the outcomes. This translates in the following reasoning:
We are computing a loss, that is a function responsible of **penalizing bad predictions**. Thus, **if the probability predicted for the real class is 1, the loss should be zero**. While **if the probability of prediction**, again **for real class, is 0**, then the **loss should be very big**.
On the basis of that reasoning, the negative log formulation is the most appropriate, because the log for values comprised in [0,1] is negative and is a non linear curve with almost vertical slope around 0. The resulting loss penalization function assume the behaviour of fig. 4.1.

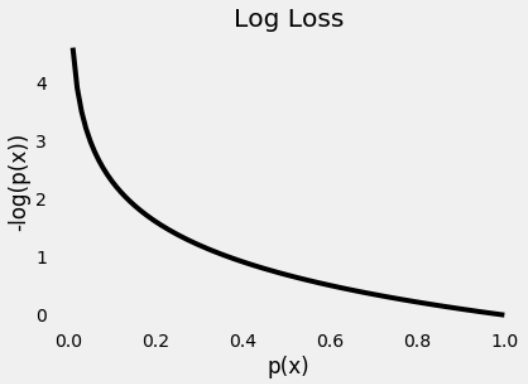


Figure 4.1. plot of the BCE loss function

Discriminator loss function in GAN model

The theory explained, translates in the following formula for discriminator loss in the GAN model:

$$Loss_D = \sum_{n}^{N} -[y_n \cdot log(x_n) + (1-y_n) \cdot log(1-x_n)] + \sum_{m}^{M} -[y_m \cdot log(x_m) + (1-y_m) \cdot log(1-x_m)] \tag{4.2}$$

Where N and M represent the batch dimension respectively for real and fake masks, x are the prediction's outcomes, y are the corresponding labels.

4.3.3 Generator Loss

The generator loss chosen for the development follows the reasoning applied in non-saturating GAN loss, but instead of maximising the log, we have a binary classification problem, where in the loss computation for generator, real and fake labels are flipped, thus the generator has to minimize the cross entropy between the discriminator outcome, and the target result, that is a **True** label. For each generated mask by the Mask-RCNN generator, the discriminator output for that mask is the first term of the BCE loss, while the second term is the true label represented by value 1.
The returned loss tells the generator how well its generation has been *bad classified* by the discriminator, that is the probability of that sample to be real with respect to a true real target (the true label).

Additional term for generator loss

The presented loss function for generator that we described, is the classical loss function that comes with GAN formulation.
The main modification that is added for the purpose of this work, is an additional term to the generator loss function. The motivation is that **we don't just want that the generator learns to fool the discriminator but also to generate masks as much as possible sharp and precise**. For this reason, an additional term is applied to generator's loss, that is a **Mean Absolute Error** loss, or **MAE**. This loss function measures the average magnitude of errors in a set of predictions, without considering their sign. The mathematical formula is simple yet effective:

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y_j}| \tag{4.3}$$

With n the total number of predictions, y_j the expected prediction and $\hat{y}_j$ the actual prediction.
Basically, the loss computes the mean of module difference for a given set of couples targets-predictions.

In the specific case of this work, the **MAE is needed to penalize the difference between the generator prediction for a mask, and the corresponding real mask**. Thanks to this loss addition results are expected to be more realistic and similar to their real counterparts.

Generator loss function in GAN model

The generator loss is implemented in the code following this formula:

$$Loss_G = \sum_{n}^{N} -[y_n \cdot log(x_n) + (1 - y_n) \cdot log(1 - x_n)] + \sum_{i}^{C} |P_{mask_i} - GT_{mask_i}| \tag{4.4}$$

Where in this case N represent the batch dimension for predicted masks, x are the discriminator outcomes for predicted masks, y are the corresponding labels. C is instead the number of matched Ground Truth - Prediction masks that are indicated with P_{mask_i} and GT_{mask_i}.

4.3.4 Alternative Loss: Least Square

In GANs field, the choice of loss function is a hot research topic and many alternate loss functions have been proposed and evaluated. One of the most popular is without doubts the Least Square Loss, also known as **Mean Squared Error** loss. This loss was proposed by Mao et al. in [54]. The idea behind is that when generated images are very different from real images, they should be penalized more with respect to other generated samples more close to reals.
Solution is to have a discriminator that seeks to minimize the sum squared difference between predictions and expected labels for that masks, following the formula:

$$discriminator : min(D(x) - 1)^2 + (D)G))^2$$

And the generator seeks to minimize the sum squared difference between predicted and expected values as though the generated images were real:

$$generator : min(D(G(z)) - 1)^2$$

Basically the procedure is none other than mean squared error between predicted samples and ground truth labels, this loss is also called **L2 loss**, and has the benefit of giving more penalty to larger errors.

Regarding the implementation, the Least Square loss brings to the following formulation for discriminator and generator loss:

Discriminator loss :

$$D.Loss_{MSE} = \sum_{n}^{N}(x_n - y_n)^2 + \sum_{m}^{M}(x_m - y_m)^2 \tag{4.5}$$

Again in the formula x_n and x_m represent the discriminator outcomes either for ground truth or predicted masks, while y_n and y_m are the associated label.

Generator loss :

$$G.Loss_{MSE} = \sum_{n}^{N}(x_n - y_n)^2 + \sum_{i}^{C}|P_{mask_i} - GT_{mask_i}| \tag{4.6}$$

Same as for the discriminator loss, x_n are the discriminator outcomes for predicted masks and y_n are the corresponding labels. P_{masks_i} and GT_{masks_i} are the predicted and ground truth masks.

This loss revealed to be very effective as we will see later in the results chapter.

4.4 Model architecture

In fig. 4.1 you can look at the graphical representation of the described GAN model architecture.

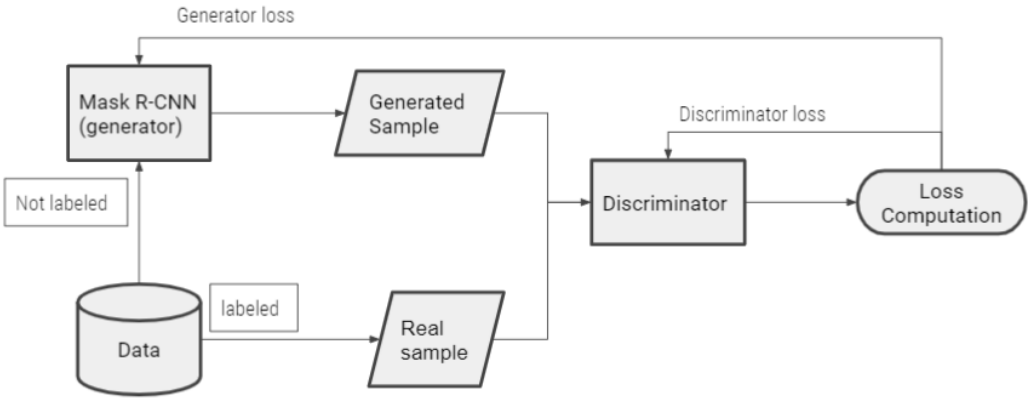


Figure 4.2. GAN model architecture corresponding to the presented method. Inputs are taken from the data in 2 different formats. Indeed for each image, the corresponding masks are given directly to discriminator for classification, while the original image is given to the generator that processes it and returns its generated masks that are fed into the discriminator again for prediction. Then this discriminator prediction both for real and generated masks is used to update, with the proper loss function, the two networks generator and discriminator

Chapter 5

Implementation Details

This chapter focuses on in detail all the implementation procedure, the motivation and the expectation behind each choice, and the training procedure that was followed to implement the method presented in chapter 4.
The following sections will be dedicated each one to a determined network implementation or choices for the training procedure, each of those sections will be provided with code snippets that help to understand what is explained.

5.1 Development Environment

The framework chosen for this thesis work is **PyTorch**, that as is readable from website [51], is *An open source machine learning framework that accelerates the path from research prototyping to production deployment.* Pytorch integrates several libraries and methods that allow to develop machine learning code with easy. In particular, it contains a subpackage of pre-trained machine learning models, called `torchvision.models`, where is possible to find definitions of models for addressing different tasks, including instance segmentation. Thanks to this package the Mask-RCNN pretrained model is extracted, fine tuned on the target dataset, and integrated as model's generator for the GAN architecture.

5.2 Dataset Definition

For the purpose of this implementation the selected dataset had been as aforementioned the **CityScapes dataset** [4], adapted in COCO format, on which we will focus on chap 6.
What matters now is understand how images are processed and used in the code.

Once dataset images are extracted and ordered in a specific folder, the key point is to setup a class that allows to load the images in pytorch framework, ready to be fed in the GAN model for training. For this purpose, pytorch support the easy addition of new datasets that extends the `torch.utils.data.Dataset` class, and implements the methods `__len__` and `__getitem__`.

In the case of this implementation, to add the CityScapes dataset I created a custom class, called `my_CocoDetection`, that basically is similar to the `CocoDetection` class provided by PyTorch in the package `torchvision.datasets` (where all datasets are subclasses of the aforementioned `torch.utils.data.Dataset`), adding the nec-

essary modifications to its methods, to allows the correct loading for the images. The class definition is in Appendix A.1.

As it is visible from the code, **I also added to the class two functions**, `annToRLE` and `annToMask`. The two methods are called inside the `__getitem__` functions, and are **used to elaborate the annotations data for each image in the dataset**, and create the corresponding mask for each element of the image.

To load the dataset once the class is ready, its sufficient to create an object , that i call `cityscapes_dataset`, as element of the class, passing the PATH where images and corresponding annotations files are located.

Last thing to do once the dataset object is created, is to create a dataloader, and more specifically, the 3 dataloaders that are needed for the 3 different tasks:

- **dataset_train**: used to fine tune the Mask-RCNN model before to be used as generator in GAN
- **dataset_train_GAN**: used to train the GAN network
- **dataset_evaluation**: used to evaluate the generator model after the training is complete

5.3 Generator Network details

As was introduced in previous chapter, the generator network is represented by Mask-RCNN model, with ResNet50 backbone network, and FPN architecture. Before to proceed is worth to have a brief look on these networks composing the generator backbone.

ResNet50

ResNet, introduced in [48], stands for Residual Neural Network and is an architecture that makes it possible to train up to hundreds or even thousands of layers and still achieves compelling performance. ResNet50 is a 50 layers network that is very powerful CNN and therefore used as backbone network for several deep learning models.

FPN

Feature Pyramid Network [49] is an architecture that allows features extraction from images with targets that are accuracy and speed. For each received input the network generates multiple feature map layers (multi-scale feature maps) with better quality information than the regular feature pyramid for object detection.

5.3.1 Model loading

We said that the starting model is pretrained on COCO, and selected from the `torchvision.models` subpackage of pytorch. The procedure to do so is the code snippet function in A.2.

Calling the function and passing the right number of classes, the model is loaded and ready to be fine tuned, but not before to setup the optimizer.

optimizer

The optimizer selected for the generator model is the Stochastic Gradient Descent optimizer (**SGD**) [57], with a learning rate of $5 \cdot 10^{-3}$, momentum 0.9 and a weight decay of $5 \cdot 10^{-4}$.
The weight decay decreases the learning rate by a factor $5 \cdot 10^{-4}$ after a fixed number of steps, that in my case is 3.
All that has been described is resumed in code A.3.

5.4 Discriminator Network Details

Differently than Generator network, the discriminator network, has been designed from scratch, and is analyzed now in detail.

5.4.1 Mask discriminator

The **mask discriminator network is a simple Convolutional Neural Network**, that takes in input images at 128×128 resolution, and outputs a single value, that as explained in the previous chapter, is the probability of that input image to be real.
To better understand the model architecture, is good to see first at the code for its implementation (in A.4) and then to explain it step by step.

As you can notice, the architecture is composed by a series of convolutional blocks, each one is comprised by a convolution in 2D with a given kernel size, a batch normalization, and a leaky ReLu.
The final block is a convolution followed by a sigmoid function, that returns an output in the range $[0, 1]$.
Let's see the meaning for each component inside the blocks.

Conv2d

The Conv2D layer is the layer representing the convolution operation, it basically receives in input a tensor of shape $n \times n$ and returns a feature map that is the result of the convolution operation of the input with the kernel, i.e. the features extractor. The kernel dimension together with the stride, that is how much the kernel is slided at each step over the input, and the padding, determines the feature map's dimension. The rule to compute the feature map output dimension is:

$$Out = [(I - K + 2P)/S] + 1$$

Where I is the input dimension, K is the kernel size, P is the padding, and S is the stride. Given an input to the convolutional layer, the more kernels are used to compute the convolution, the more are the features extracted from input. The general rule for deciding the number of kernels in each convolutional layer, is to start with a number appropriate with the dataset complexity, in my case 64 kernels are chosen, and then to generally couble the number of kernel filters at each successive layer, because number of possible combination grow.

BatchNorm2d

Batch normalization standardizes the activations from the prior layer, in order to have zero mean and unit variance. This results in a stabilization of the training process, as was demonstrated in the DCGAN paper [44].

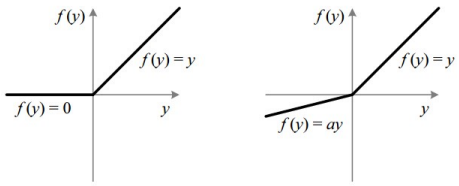


Figure 5.1. ReLu (left) and Leaky ReLu (right) functions

Leaky ReLU

ReLu [42] stands for rectified linear unit, and is used to provide output values that are 0 for negative and zero inputs, while is the value scaled by the slope factor is the input is positive. This function is the best practice in terms of returned results when it comes to convolutional networks.
In GANs however a little modification of ReLu demonstrated to have better results, the **Leaky ReLu** [43]. This function is equal to ReLu except for the fact that negative inputs are no more corresponding to zero output, but are scaled on the basis of the negative slope, that is usually 0.2, as in the case of my implementation. The ReLu and Leaky ReLu are compared in fig. 5.2 for a better understanding.

Sigmoid

The sigmoid layer is the activation function of the last block in the discriminator. The layer has to return an output comprised in the [0, 1] prediction range that represents the probability of a given sample to be real. The role of sigmoid comes here. Motivation is that the output we need has to be confined and so a linear activation function, like a line admitting values from $-\infty$ to $+\infty$ would not be correct. The plot of the sigmoid function is the following in fig. 5.2

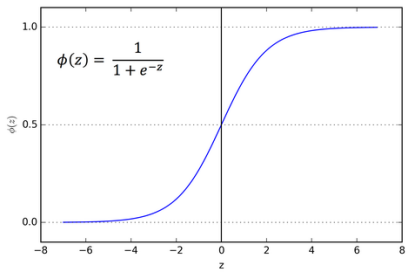


Figure 5.2. Plot of sigmoid function

Sigmoid is one of the most appropriate choices when dealing with predicting probability.

5.4.2 Weights initialization and optimizer

Once the structure for the discriminator is complete, the model can be initialized. For this purpose, the model weights must be initialized and an optimizer is associated to the model.
The weights initialization, is done following the guidelines from [44], where authors specify that all model weights shall be randomly initialized from a Normal distribution

with mean=0, standard deviation=0.02. The function `weights_init`, defined in A.5, is responsible of this.

As regards the optimizer, it is an **Adam** optimizer [56], that stands for **Adaptive Moment Estimation**, with learning rate = $0.2 \cdot 10^{-4}$ and betas hyperparamater equal to $\beta 1 = 0.5$ and $\beta 2 = 0.999$.

The loss function, as explained in chapter 4 is the **Binary Cross Entropy**.

5.5 Network parameters

In table 5.1 you can see the number of parameters for both the generator network and the discriminator network.

model	Total parameters	Trainable parameters
discriminator	11,412,480	11,412,480
generator	43,960,069	43,737,669

Table 5.1. Network parameters for discriminator and generator models

5.6 Training Procedure

The procedure to train the GAN model is divided in steps. In this section we will review each part, explaining the motivations behind each choice and part of the code.

The procedure once the dataset and the models are ready, starts obviously with an iteration over the dataset, for a given number of epochs [1], and after fixed number of epochs during training the model is saved together with the losses for each submodel. Now we will focus in the details of each part.

5.6.1 Image processing

In each epoch of the training, a series of images, that are all the images belonging to the GAN training dataset, are processed by the network. Before to be fed into the discriminator model though, images are processed to extract the right information needed by the network.

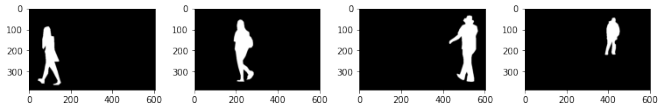
Data extraction: Training starts with image extraction from the dataset. This image is a tensor image, meaning that its data type is a tensor, a type of variable that is similar to numpy array and used by PyTorch for its computation. Once the image is extracted, the corresponding masks, both predicted and reals, have to be retrieved. Thus, first of all I extract the ground truth masks and boxes corresponding to the image, that are extracted directly from the dataset being part of it, and right after that I fed the image to the generator network, the Mask-RCNN model, that gives its prediction about the image, returning all the information that we learned this model gives, thus mask, box, label, score, and this for each predicted instance

[1] An epoch in machine learning means one complete pass of the training dataset through the algorithm

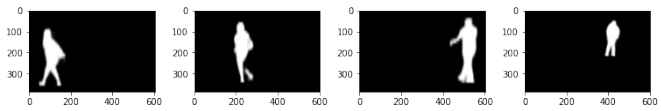
inside the image.
Once we have both the **ground truth** and **predicted** data for the image, they **can be used to prepare the input data for the discriminator**. As was explained before, the mask discriminator expects as input a 128×128 binary mask, that it then elaborates returning the resulting probability for that mask to be real.
However masks returned by the model, and ground truth masks, are not of this size, and rather each mask is just a binary image of the native resolution of the dataset, that is a one channel 2048×1024 image, where the white pixels, those having a 1 value, are those representing the mask for the corresponding instance. Hence as it's easy to guess, there is the need to resize the masks in the right resolution accepted by the discriminator.
Here comes the role of the bounding boxes. **Bounding boxes are** indeed **used to crop the mask images** in the right position in order to retrieve the right portion of the image where the mask is present. Once the image is cropped in the right place, it is resized in the desired resolution and ready to be processed by the discriminator network. The code in A.6 shows all the described steps.

Matching computation: As it was explained, the training procedure in the GAN model, is carried out on the basis of the discriminator prediction for the received input masks, that are real masks and predicted masks provided in batches, that cause the update of the models weights towards a the desired behaviour, the discriminator that improves at distinguishing the real and fake masks, and the generator learning to generate always better masks.
This procedure doesn't require that the real masks and the predicted masks provided to the discriminator are paired, meaning that there is no need that each one of the real masks has a correspondence in the predicted masks, because in several situations the generator model can wrongly predict a mask where it sholdn't be, or viceversa the generator could miss to predict a mask that instead was to be predicted.
For this reason the training procedure goes with unpaired sets of real and predicted masks, but with an additional code that **checks for matching**.
If a matching between the real mask and the mask predicted by the generator is find during training phase, the loss of generator is modified.
Usually the generator loss is simply the **BCE** or **MSE** between the discriminator's outcome for its prediction, and the true label. **When a matching with real mask is found** however, this **additional information is used to add another term to the loss**, that is the **Mean Absolute Error** between the prediction and the real mask. This value is multiplied by a term λ, whose value is set to 100, as suggested by the authors of [27], and has the role to emphasize the mistake.
The additional term in the loss has the purpose of helping the generator in a better mask prediction, generating masks that are as much as possible similar to the their real counterpart.
The code that computes that matching between predictions and ground truth, make use of a function, called `compute_matches`, and is imported from the Mask-RCNN implementation by [52]. The function takes in input the array of predicted masks and boxes, together with real masks and boxes, the labels of the instances and the predicted scores, and returns three elements, the `gt_match`, that is the list of elements of predictions that corresponds to the element in ground truth, the `pred_match` that is the opposite, indicating for each element in prediction which is the ground truth elements that corresponds to it, and then returns the overlaps (not needed in this work).
In A.7 you can see the code that computes matching for each image, while in fig. 5.3

you can see an example of generated matches between ground truth and predicted masks on an image.



(a) GT masks



(b) predicted masks

Figure 5.3. Resulting masks pairing after matching code is executed

5.6.2 Training Loop

The training loop for the GAN model is carried out training first the discriminator with real batches, then the discriminator again with fake batches, and last the generator is trained.

Discriminator training with real samples The training loop begins training the discriminator with real mask samples. The process consist into, given an image with the corresponding ground truth masks extracted, iterate over the masks and for each mask, as explained, crop the image in the portion indicated by the corresponding box, and send the binary resulting image, resized in 128×128 to the discriminator, that process it and return a probability value. The probability value is saved inside the `output` variable, used to compute the loss. Indeed right after the prediction loss is called, among `output` and `label`, that is a one dimensional tensor with just one value, in this case 1, that represents the **True label**.
The call to the loss function returns the value of the error, that is backpropagated, i.e. value of gradients for Discriminator is computed in the backward pass.

Discriminator training with fake samples The procedure to train the discriminator with predicted (i.e. fake) samples is exactly the same as the reals, except for the fact that in this case the label is a **False label** and thus is a tensor with value 0.

Once all reals and predicted samples for a given image have been processed, the mask discriminator is updated calling `optimizerD.step()`. This function is responsible of updating the parameters, using the **SGD** in this case, as was discussed before.

Generator training Once the discriminator has been trained with real batch and fake batch, and has been updated, the generator is trained. To do so, the fake batch, i.e. the batch of predicted masks generated by this latter, is processed again by the discriminator after it has been updated. The predictions are then used to compute the generator loss, as BCE between the returned discriminator's outcomes and a corresponding number of real labels, remember that **we associate real labels to fake samples this time, because fake labels are real for generator cost function**. In this way the error is computed, and calling `backward()` on it, the

gradients for G are calculated and then updated by calling the optimizer.

The pseudocode for GAN training is the following:

Algorithm 1 GAN Training Loop

```
for epoch in range = 1, 2, ..., N_epochs do
    for k, data in enumerate(data_loader) do
        Extract GT and predictions from image.
        Compute matches between GT and predictions
        Train discriminator with real batch
        Train discriminator with predicted batch
        Update discriminator
        Train generator with predicted batch result by discriminator
        Update generator
    end for
    if epoch % m == 0 do
      Save models and losses
end for
```

5.7 Further Refinements

In addition to all the explained procedure for the training setup, some refinements were added to the process, that are basically a number of best practices useful when training GANs models, and are called **GAN hacks**. These best practices were presented by Soumith Chintala, one of the authors of [44], in the *NIPS 2016* presentation titled *How to train a GAN?*.
The practices that I adopted are:

- **Label smoothing**: As we stated, real labels are represented by value 1 and fake labels are represented by value 0. A better practice is to use soft labels, meaning that true labels have values slightly more or less than 1, in my case randomly taken in range [0.7, 1.2]. This has the advantage of regularizing the model during training. On the same manner the fake labels are comprised in a range [0, 0.3]

- **Noisy labels**: Images received by the discriminator, are labeled with 1 if reals, and with 0 if predicted. A useful approach is to introduce some errors with a low probability in the labels, that is sometimes real images are marked with fake labels and vice versa predicted images are marked as reals. In my case I perform this change with a probability of 5%. The procedure is useful to make the discriminator not so much confident on its predictions and thus avoid loss convergence to zero too quickly.

- **Inputs normalization**: the **input images** provided both by the ground truth and by the generator, **are in a range [0,1]**. Normalizing them means to **transform the range of their values in [-1,1]**

Chapter 6

Experimental setup and results

To validate the described method, a comparison is presented in this chapter, between the baseline Mask-RCNN model fine tuned on the dataset, and the model trained with GAN framework. The experiments and test phase has been divided in multiple trials, and using two different datasets, A detailed explanation is now given for each one.

6.1 Experimental setting

For all the experiments, the baseline model is Mask-RCNN model with ResNet50 backbone and FPN architecture. The network train has been performed with two different setups, the first is a Google Colab environment https://colab.research.google.com/, with a Nvidia Tesla K80 GPU, offering 24GB of memory. The second setup is with a Nvidia GeForce GTX 1080 GPU, with 8GB of memory.

Comparison between this model and the GAN Mask-RCNN model: Before to present the experiments and the obtained results, it is good to see a comparison between the model presented in this thesis and the model by paper in [39].
We remember that our model presents a single generator, with input images size of one channel and resolution 120×128. On the other hand, *GAN Mask-RCNN* model has a more complex structure with a double discriminator configuration, one responsible for masks discrimination and the other one responsible for boxes discrimination.
On the contrary, the proposed model for this work is considerably simpler and necessitates of lower computational power to be trained.
But the most important difference is on the Mask-RCNN backbone network, that is ResNet50 in our case while is ResNet101 in the paper, with more than double number of the parameters.
This means a more complex model that with a batch size of 16 images needs need a very high end and extremely powerful configuration to be trained. We can see a rough estimate of the parameters of the two models in table 6.1, having approximately a model complexity that is halved in our case.

Important Premise: Due to the explained difference and limited resource and training times with respect to other works, and with respect also to the *GAN Mask-RCNN* paper, the results obtained from this configuration have average level of performances that is inevitably lower than the other state of the art results, **despite**

model	Discr. parameters	Gen. parameters	total parameters
presented model	11,412,480	43,737,669	55,150,149
GAN Mask-RCNN	$\sim$ 20,000,000	$\sim$ 90,000,000	$\sim$ 120,000,000

Table 6.1. Number of parameters comparison between the presented model and the GAN Mask-RCNN model in [39]

this they proved that the approach followed for this thesis's work is valid and can lead to concrete gain of performances even on more heavy and longer models to train.

6.2 Evaluation metrics

For the evaluation phase, the classical COCO metrics are reported, i.e. **Average Precision** AP, and **Average Recall** AR. Both are measured over multiple IoU (Intersection over Union) thresholds.
The IoU is measured as Intersection of prediction with ground truth, and Union of prediction and ground truth, with the following formula:

$$IoU = \frac{area(P \cap GT)}{area(P \cup GT)} \tag{6.1}$$

Where in P represents the prediction and GT the ground truth.

We remember also that the formulas for computing Precision and Recall are:

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{6.3}$$

Where TP are the true positives, i.e. the positive elements that have been predicted as positives, FP are the false positives, i.e. the negative elements that have been predicted as positives, and FN are the false negatives, i.e. the predicted negatives elements that were instead positives.
A detection is considered a true positive (TP) only if it satisfies three conditions: confidence score > threshold; the predicted class matches the class of a ground truth; the predicted bounding box has an IoU greater than a threshold (e.g., 0.5) with the ground-truth. The violation of one of these rules implies a false positive.
When the confidence score of a detection that is supposed to detect a ground-truth is lower than the threshold, the detection counts as a false negative (FN).
With this in mind, we can understand the formula of Average Precision, basically it is the precision averaged across all unique recall levels.

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - ri) p_{interp(r_{i+1})} \tag{6.4}$$

Where $r_1, r_2, r_3, ..., r_n$ are the recall levels at which precision is first interpolated.

Similar to average precision, the **Average Recall** is the recall averaged over all IoU $\in [0.5, 1.0]$ and can be computed as two times the area under the recall-IoU curve:

$$AR = 2 \int_{0.5}^{1} recall(o)\ do \tag{6.5}$$

With o that is IoU and *recall(o)* the corresponding recall.

Coming to the experiments, as the COCO guidelines indicate, the evaluation metrics for AP are:

- AP: AP at IoU=.50:.05:.95
- $AP^{IoU=.50}$: AP at IoU=.50
- $AP^{IoU=.75}$: AP at IoU=.75
- AP^{small}: AP for small objects, $area < 32^2$pixels
- AP^{medium}: AP for medium objects, $32^2 < area < 96^2$pixels
- AP^{medium}: AP for large objects, $area > 96^2$pixels

And ass regards average recall, the values indicate:

- $AR^{max=1}$: AR given 1 detection per image
- $AR^{max=10}$: AR given 10 detection per image
- $AR^{max=100}$: AR given 100 detection per image
- AR^{small} : AR for small objects, $area < 32^2$pixels
- AR^{medium} : AR for medium objects, $32^2 < area < 96^2$pixels
- AR^{large} : AR for large objects, $area > 96^2$pixels

6.3 Datasets

For the development of this thesis, I selected two datasets, that are PennFudan [55] and CityScapes [4]. The first is a small dataset containing only one class, but that has been useful to validate the GAN approach and obtain results in small amount of time.
On the other hand CityScapes is one of the most challenging and famous dataset when talking about segmentation, and that suits perfectly with the purpose of this thesis since we are focusing on autonomous driving scenario.
Let's see more in depth the two datasets.

6.3.1 PennFudan dataset

As starting experiment, before to proceed on the more complex dataset that is CityScapes, the approach was tested on **PennFudan** pedestrian dataset [55]. This dataset is comprised of much less images with respect to CityScapes, and contains only one class, that is **pedestrian**. Nevertheless is useful to see how the GAN approach can work on different datasets being as well effective. In detail, the images are 170, with 345 labeled pedestrians. The split is done by taking 75 images for the Mask-RCNN model finetuning, 75 images for the GAN training, and the remaining

20 images for evaluation.
The benefit of this dataset is that, being small, training times are consistently lower and is possible to see the results very quickly.

6.3.2 CityScapes Dataset

Talking about autonomous driving, CityScapes dataset [4] is probably the most appropriate choice. This dataset is composed by 5000 images with high quality annotations, 20000 images with coarse annotations, of 50 different cities in Germany. The dataset is thought for semantic segmentation tasks, images are indeed in the form visible in fig. 6.1

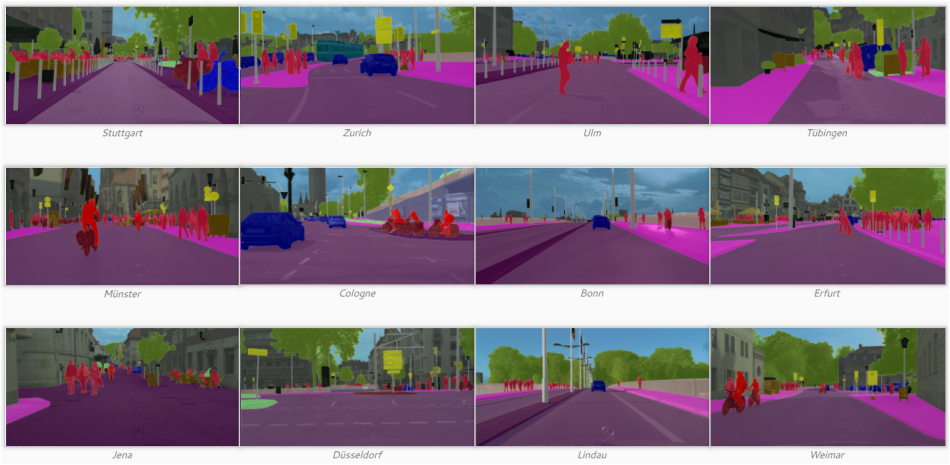


Figure 6.1. Sample images from CityScapes dataset [4]

But the purpose of this thesis is instance, and non semantic, segmentation. Thus the images of the dataset needed to be transformed in COCO format. This is achieved through a python script [47], that allowed to obtain a total of 2975 instance segmentation images, with instances belonging to 9 classes (*BG, person, rider, car, bicycle, motorcycle, truck, train, bus*), with resolution 2048 $\times$ 1024 pixels. The obtained images are in the format visible in fig. 6.2

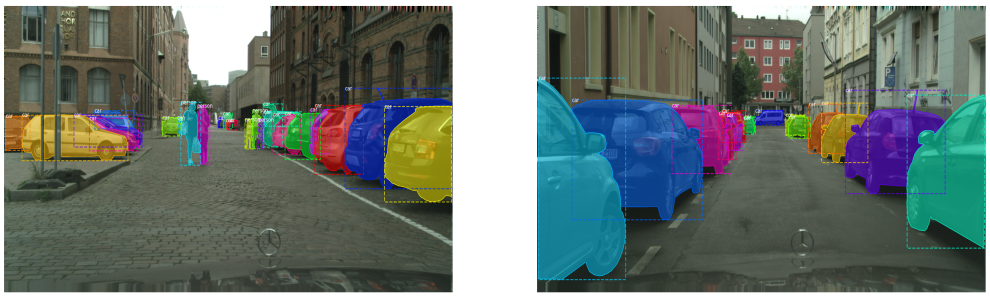


Figure 6.2. CityScapes images format after COCO translation

The resulting dataset after transformation, is split in 3 subsets used for training and validation. In particular, for the first experiment, I selected 300 images for

Mask-RCNN finetuning that bring to Model A, 300 images for GAN training, that are the images on which the Model A is fine tuned to reach model B, whose performances are put in comparison with the GAN model trained starting from model A. Finally, 50 images are selected for evaluation.

6.4 Prepare the baseline

The baseline mask, in each experiment, is prepared by taking the pre-trained Mask-RCNN model with ResNet50 backbone and FPN architecture, and performing fine tuning on the dataset in order to obtain the starting instance segmentation model, that has to been then improved by GAN training.
This starting model, is obtained in each experiment on the same way, that is, calling **dataset A** the first subset of dataset, and **dataset B** the second subset of the dataset, finetuning the Mask-RCNN model first on dataset A, and save it as starting model for GAN training, and then on dataset B. This last model obtained is the Mask-RCNN model that has to be **beaten by GAN training**.
Therefore, GAN training of the Mask-RCNN model over dataset B, has to perform better than the second finetuning of Mask-RCNN model of dataset B, without GAN approach.
Finetuning starts with an SGD optimizer, a learning rate of $5 \cdot 10^{-3}$ and a weight decay of $5 \cdot 10^{-4}$ every 3 epochs. After the training is complete, the baseline model is ready for the GAN training. In the following section we can see the experiments' results.

Model	IoU metric	AP	AP_{50}	AP_{75}	AP_{small}	AP_{medium}	AP_{large}
Base Mask-RCNN	bbox	0.280	0.339	0.339	-1.000	0.118	0.284
GAN Mask-RCNN (BCE)	bbox	0.276	0.438	0.322	-1.000	0.000	0.305
GAN Mask-RCNN (MSE)	bbox	**0.372**	**0.487**	**0.453**	-1.000	**0.700**	**0.366**

Table 6.2. AP bounding boxes Results comparison on PennFudan dataset

Model	IoU metric	AP	AP_{50}	AP_{75}	AP_{small}	AP_{medium}	AP_{large}
Base Mask-RCNN	segm	0.221	0.301	0.251	-1.000	0.030	0.228
GAN Mask-RCNN (BCE)	segm	0.232	0.353	0.253	-1.000	0.000	0.259
GAN Mask-RCNN (MSE)	segm	**0.329**	**0.466**	**0.396**	-1.000	**0.543**	**0.322**

Table 6.3. AP segmentation Results comparison on PennFudan dataset

Model	IoU metric	AR_1	AR_{10}	AR_{100}	AR_{small}	AR_{medium}	AR_{large}
Base Mask-RCNN	bbox	0.280	0.339	0.339	-1.000	0.118	0.284
GAN Mask-RCNN$_{BCE}$	bbox	0.276	**0.438**	0.322	-1.000	0.000	**0.305**
GAN Mask-RCNN$_{MSE}$	bbox	**0.372**	**0.487**	**0.453**	-1.000	**0.700**	**0.366**

Table 6.4. AR bounding boxes Results comparison on PennFudan dataset

Model	IoU metric	AR_1	AR_{10}	AR_{100}	AR_{small}	AR_{medium}	AR_{large}
Base Mask-RCNN	segm	0.237	0.423	0.423	-1.000	0.150	0.437
GAN Mask-RCNN$_{BCE}$	segm	0.170	0.362	0.362	-1.000	0.000	0.382
GAN Mask-RCNN$_{MSE}$	segm	**0.241**	**0.463**	**0.463**	-1.000	**0.750**	**0.447**

Table 6.5. AR segmentation Results comparison on PennFudan dataset

6.5 PennFudan dataset Results

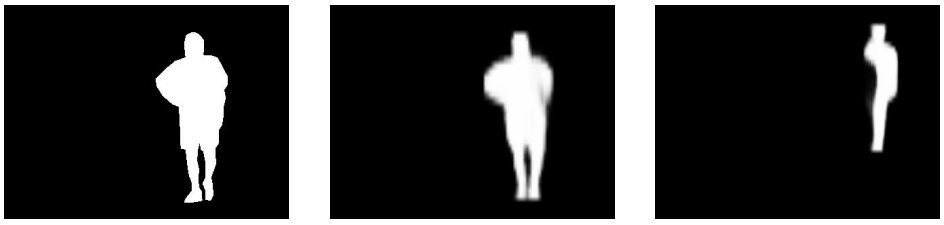
6.5.1 Quantitative results

Average Precision

In table 6.2, you can see the AP results of the base model compared with the GAN model trained with BCE loss and with MSE loss, evaluated on bbox metric. While in table 6.3 the same three models are evaluated on segmentation metric.

From the results is clear that the GAN approach demonstrates to be effective both with BCE loss and with MSE loss, whith this latter case that gives a good performances gain in basically all the AP fields.
A more accurate analysis reveals that the GAN model is particularly effective, and has an overall improvement of AP in average more than 40%. This result even if coming from a relatively easy dataset with just one class, it's important to understand and have a confirmation of the goodness of the adversarial approach.
An important clarification is needed on the AP_{small} field that is -1.0 on all the models, this is due to the lack of small (i.e. comprised in the evaluation metric range) instances in dataset.

Figure 6.3. Original image from dataset



(a) GT mask **(b)** predicted mask 1 **(c)** predicted mask 2

Figure 6.4. Comparison between prediction and GT images

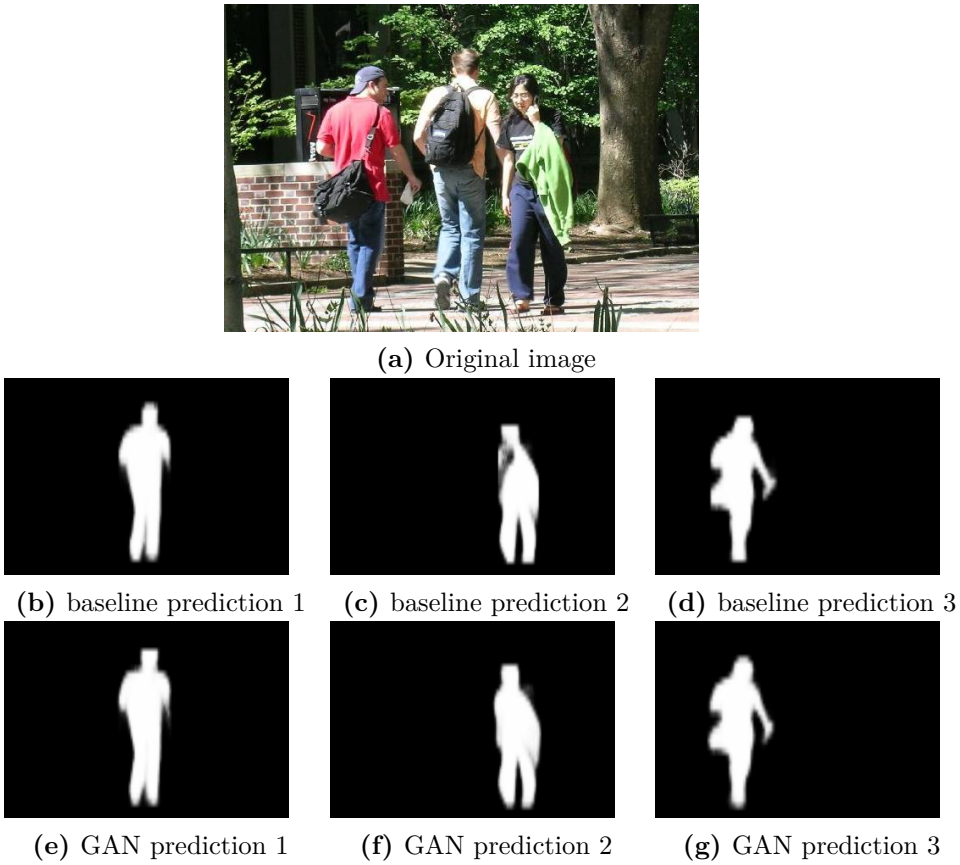
Average Recall

In the same manner of AP, in table 6.4 and table 6.5 you can see a comparison between the base Mask-RCNN model, and the two GAN models with BCE loss and MSE loss, this time on average recall evaluation.
Even in this case results are clear, showing a performance gain of the GAN models, with again the MSE GAN model that outperforms the base model in every field.
The numbers reveal that in particular situations like this where the instances to detect belong just to one class, the effectiveness of the approach is more consistent, but at the same time they provide a good base of confidence for testing on more complex datasets, like we are going to see in a while with CityScapes.

6.5.2 Qualitative results

In the following images you can see some comparison between baseline model predictions and GAN model predictions.
A good thing to remark is that in several situations, the model recognize even instances that are not present in ground truth, as you can see from fig. 6.3 and the corresponding ground truth and predicted masks in fig. 6.4. In that situation, we can see that the model understood correctly how to detect and classify instances, indeed is capable of correctly segmenting instances like the man on the background, whose figure is overposed by the subject in foreground. This is a good result which confirms that model training has led to improvements in the right direction.

Looking instead at figure 6.5, picture (a) is the original image on which models perform mask predictions. The corresponding predicted masks by the models are on the bottom of the image.
On the upper row you can see the prediction of baseline model that returns the 3 masks for subjects in picture (a). In the lower row the same three masks are this time generated by the GAN model.



(a) Original image

(b) baseline prediction 1 **(c)** baseline prediction 2 **(d)** baseline prediction 3

(e) GAN prediction 1 **(f)** GAN prediction 2 **(g)** GAN prediction 3

Figure 6.5. Comparison between baseline prediction (upper row) and GAN model prediction (lower row), for the image in (a)

With an accurate comparison of the pictures, you can notice the difference between the mask generated for the subject in the middle, representing the mask for the girl with the green jacket hanging on her arm. The upper mask results a bit imprecise, having the above part of the body not correctly segmented, with a empty region right under the head.
GAN model solves this issue generating a more correct and well shaped mask, where the whole body is recognized and properly segmented.

Model	IoU metric	AP	AP_{50}	AP_{75}	AP_{small}	AP_{medium}	AP_{large}
Base Mask-RCNN	bbox	0.300	0.537	0.212	0.054	0.360	0.523
GAN Mask-RCNN $_{BCE}$	bbox	0.252	0.413	0.289	**0.074**	0.402	0.495
GAN Mask-RCNN $_{MSE}$	bbox	0.268	0.470	**0.293**	0.068	**0.423**	0.510

Table 6.6. AP bounding boxes Results comparison on CityScapes dataset

Model	IoU metric	AP	AP_{50}	AP_{75}	AP_{small}	AP_{medium}	AP_{large}
Base Mask-RCNN	segm	0.279	0.515	0.318	0.024	0.271	0.571
GAN Mask-RCNN $_{BCE}$	segm	0.215	0.387	0.218	**0.075**	0.293	0.396
GAN Mask-RCNN $_{MSE}$	segm	**0.284**	0.388	**0.351**	0.024	**0.307**	0.425

Table 6.7. AP segmentation Results comparison on CityScapes dataset

Model	IoU metric	AR_1	AR_{10}	AR_{100}	AR_{small}	AR_{medium}	AR_{large}
Base Mask-RCNN	bbox	0.201	0.380	0.392	0.133	0.438	0.609
GAN Mask-RCNN$_{BCE}$	bbox	0.128	0.338	0.352	0.124	**0.485**	0.618
GAN Mask-RCNN$_{MSE}$	bbox	0.154	**0.393**	0.358	0.128	0.480	**0.625**

Table 6.8. AR segmentation Results comparison on CityScapes dataset

Model	IoU metric	AR_1	AR_{10}	AR_{100}	AR_{small}	AR_{medium}	AR_{large}
Base Mask-RCNN	segm	0.200	0.349	0.354	0.053	0.324	0.512
GAN Mask-RCNN$_{BCE}$	segm	0.116	0.267	0.275	0.048	**0.379**	**0.514**
GAN Mask-RCNN$_{MSE}$	segm	0.118	0.264	0.270	**0.058**	0.365	0.502

Table 6.9. AR segmentation Results comparison on CityScapes dataset

6.6 CityScapes dataset results

6.6.1 Quantitative Results

Average Precision Results for AP on bbox metric, are in table 6.6, where you can see a comparison again between base Mask-RCNN model, GAN model with BCE loss and GAN model with MSE loss. In table 6.7 instead there are the results of the same models evaluated on segmentation metric.

This time the situation is different from the previous cases, indeed the GAN models are capable of doing better than the baseline but not in every field. This can be due to the much bigger complexity of the dataset, that results in the more arduous task of segmentation by the models.
What is interesting is that despite being focused on mask prediction improvement, the proposed method also results in a bounding box metric improvement. This can be explained thinking at the GAN approach, indeed at each wrong prediction by the generator network, weights are updated, and this cause not only an improvement in masks creation but this movement of weights in the right direction causes also an improvement for the bounding box prediction.

Coming to AP results for segmentation in table 6.7, is noticeable how the GAN model is capable of doing better than the Mask-RCNN baseline respectively in AP, with a 1.79% increase, AP_{75} having a good 10,37% increasing, AP_{small} and AP_{medium}, being also very close to baseline performances in the other fields. Particularly interesting is the result for AP_{small}, where the performance gain is very consistent and means that the GAN generator model is capable of better segment small object, with an higher precision than what baseline can do. This observation is valid also for the AP_{medium} metric, that with a 13.28% of perfomance improvement confirm the better precision in segmenting objects of medium size.

Average Recall The situation is similar coming to average recall, in table 6.8 and table 6.9 the results show that again GAN model is capable of outperforming the baseline model in several fields, demonstrating the effectiveness of the used approach. Focusing on the results, we can see that again the GAN model improves also the bounding boxes creation, but the most relevant results are of course from segmentation metric in table 6.9. Here we have an improvement with respect to the baseline model, in the AR_{small}, AR_{medium} and AR_{large} fields, with an improvement respectively of 9.43%, 16.97% and 2.62%. Even in this case GAN model performances in other fields are close to the baseline.
These results confirm the validity of approach, the model is able to perform very good in relation of the stats obtained with training, that we remember even if are not high due to resources limits, confirm the right direction of GAN training.

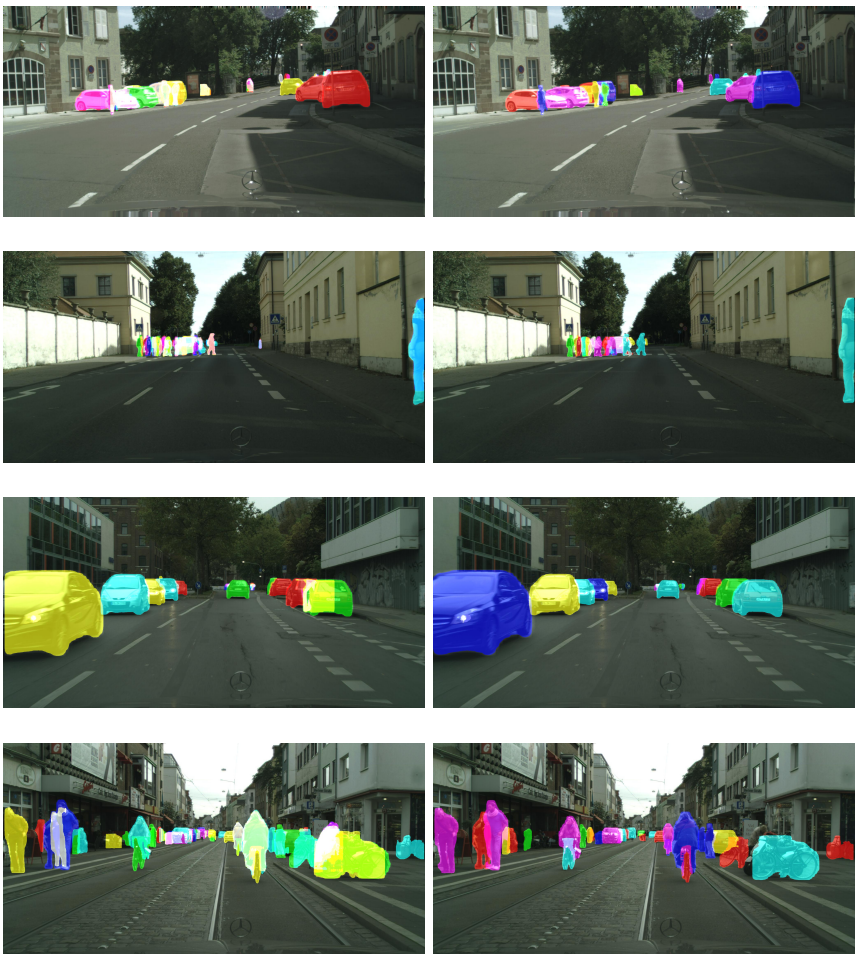
On the basis of what we can see in tables, we can hypothesize that with more data and the right hardware support, these results can reflect with the same trends on higher overall values, proving in that case that the adversarial approach is indeed capable of improving the baseline instance segmentation model right in that situations where it suffers more, like lack of precision in small/medium size objects, and difficulty in the segmentation of crowds.

Also, what emerged from the training experiments is that the approach is more valid when used to improve an already particularly performing starting model. In the sense that if the basic model is able to generate sensible masks, while presenting various inaccuracies and smudges, then the GAN approach is able to improve it in a consistent way. Conversely, if the starting model has very low performance, the approach in question does not prove to be as valid. This confirms the expectations and proves that an adversary type approach is valid for the improvement of instance segmentation models just when they are already able to have good performances, in fact going to perfect all those cases in which the basic models suffer from the classics. problems described in the previous chapters.

6.6.2 Qualitative results

In fig. 6.6 you can see a qualitative results comparison between instance segmentation masks generated by the baseline model, lying on the left column, and the same masks generated by the GAN model, lying on the right column.

Results show a performance gain in favor of the latter. For example in the upper pair of images, where on the left side we have a generated instance segmentation mask image by the baseline model, while on the right side the same prediction is done by the GAN model. Looking at the pictures with attention, we can notice that the left image presents more artifacts than the right one, with shapes that are badly



baseline masks GAN masks

Figure 6.6. Comparison between baseline model (left column) and GAN model (right column) on masks prediction in cityscapes

segmented, like the person on left in front of the car, where the mask is not correctly shaped and confuse with car on the back. On the right image the same person is this time perfectly segmented and distinguished by the car on the back. The same improvement is notable in the masks of the other cars and people again on the left of the scene.

The same observations apply in the other images. In particular on second pair of images, we have a comparison in segmentation for a crowd of people. On the left column there is a wrong mask prediction in several subjects, where the mask for each person presents more errors and lack of precision than the corresponding mask generated by the GAN model. Right column picture indeed shows a better mask prediction with crisp boundaries and right shapes for all the subjects. This confirms the results seen in quantitative inspection, because in the scene the majority of subjects is in the distance, implying small shapes to be detected, where as we already seen, the adversarial approach is more clever in the segmentation.
Coming to the third pair of images, we have a series of cars again with shapes overlapping. On the left picture, the baseline model performs well but doing again some mistakes, like the car on right side of the scene, that present a wrong yellow mask applied. The situation is again better on the right picture, where each car is correctly segmented and there are not glaring errors.
Lastly, the fourth pair of images, shows a really complex street situation where there are many people both at a short distance and at a great distance. Such situation is inevitably difficult for whatever model, but with an accurate analysis we notice that left picture has less precision in the people segmentation, like the man in left side of the scene, that has a double mask blue and white because it's not correctly recognized, or the bicycles in the right side, with the yellow one that is confused with a man on the back, and the cyan bicycle in the extreme right that even if segmented without huge errors still has a lack of precision in the shape.
All these errors are partially or totally solved by the GAN model image of right column, where shape of the man is this time perfectly recognized, and bicycle masks are undoubtedly better with a much more detail and crisp boundaries.

Chapter 7

Conclusions and future works

The work presented demonstrated how a simple GAN framework, with the right choice of the losses can be effective into improving and refine instance segmentation tasks.
The idea at the base is to use Mask-RCNN instance segmentation method as generator module in a GAN network, where paired with a convolutional neural network discriminator, it is capable of improving its performances in the instance segmentation masks generation. This is possible thanks to a GAN loss formulation where the two models are involved in a zero-sum game where each one has to minimize what the other model want to maximize, causing a good competition that brings to performance improvements.
The realized GAN is capable of outperforming the standard Mask-RCNN model and if trained properly with the right resources can become a reference model in instance segmentation, confirming the power of Generative Adversarial Networks not only in classical images generation tasks but also as instrument to improve already existing models and as a solution to a wide range of problems.

As future works, one of the greatest positive remarks of the architecture, is that is absolutely model independent, thus it can be translated and implemented with ease in other current or future state of the art instance segmentation methods.
Also the architecture and the functioning of this GAN model allow to use it in other segmentation tasks, like semantic segmentation. Furthermore, a further improvement that can be done is the addition of a more complex discriminator network that trained on the object detection task is capable of recognizing the detected instances by the generator and penalize it whenever a detection is wrong.

Improvements in this direction can lead to obtain, with the right implementation choices, very high performing networks, that with the constant evolution of algorithms and hardware, would result in the near future to a chronic and widespread diffusion of autonomous driving, at a commercial level and accessible to all, a situation from which we could be less distant than what we imagine.

Appendix A

Appendix: relevant code for implementation

In this appendix you can find the most relevant code snippets.

```
class my_CocoDetection(VisionDataset):
    """‘MS Coco Detection <https://cocodataset.org/#detection-2016>‘_ Dataset.

    Args:
        root (string): Root directory where images are downloaded to.
        annFile (string): Path to json annotation file.
        transform (callable, optional): A function/transform that takes in an PIL image
            and returns a transformed version. E.g, ‘‘transforms.ToTensor‘‘
        target_transform (callable, optional): A function/transform that takes in the
            target and transforms it.
        transforms (callable, optional): A function/transform that takes input sample and its target as entry
            and returns a transformed version.
    """

    def __init__(self, root, annFile, transforms = None):
        self.root = root
        self.annFile = annFile
        self.transforms = transforms

        from pycocotools.coco import COCO
        self.coco = COCO(annFile)
        self.ids = list(sorted(self.coco.imgs.keys()))

    def __getitem__(self, index: int) -> Tuple[Any, Any]:
        """
        Args:
            index (int): Index

        Returns:
            tuple: Tuple (image, target). target is the object returned by ‘‘coco.loadAnns‘‘.
        """
        coco = self.coco
        img_id = self.ids[index]
        ann_ids = coco.getAnnIds(imgIds=img_id)
        target = coco.loadAnns(ann_ids)
```

```
36
37          path = coco.loadImgs(img_id)[0]['file_name']
38          img = Image.open(os.path.join(self.root, path)).convert('RGB'
    )
39
40          # # resize the image to lower resolution
41          # img = img.resize((1024, 512))
42
43          width = img.size[0]
44          heigth = img.size[1]
45
46          boxes = []
47          masks = []
48          class_ids = []
49          image_id = index
50          iscrowd_list = []
51          areas = []
52
53          for elem in target:
54
55            if(len(elem['bbox'])<4):
56              print("!!!WARNING!! - found non 4 boxes for element")
57              print("skipping to next image...")
58              #continue
59
60            # sort the box coordinates in the right order
61            xmin = elem['bbox'][0]
62            xmax = xmin + elem['bbox'][2 ]
63            ymin = elem['bbox'][1]
64            ymax = ymin + elem['bbox'][3]
65
66
67            boxes.append([xmin, ymin, xmax, ymax])
68
69            mask = self.annToMask(elem['segmentation'], heigth, width)
70
71            masks.append(mask)
72            class_ids.append(elem['category_id'])
73            #image_ids.append(elem['image_id'])
74            iscrowd_list.append(elem['iscrowd'])
75            # compute area and append to list
76            areas.append(elem['area']) # np.int(elem['area']/4))
77
78          # convert everything into a torch.Tensor
79          boxes = torch.as_tensor(boxes, dtype=torch.float32)
80          masks = torch.as_tensor(masks, dtype=torch.uint8)
81          class_ids = torch.as_tensor(class_ids, dtype=torch.int64)
82          image_id = torch.tensor(image_id)
83          iscrowd_list = torch.as_tensor(iscrowd_list, dtype=torch.
    int64)
84          areas = torch.as_tensor(areas, dtype=torch.float32)
85
86          # changed the name inside brackets because this format is
    required
87          # in the train_one_epoch method
88          gt_instances = {}
89          gt_instances['boxes'] = boxes
90          gt_instances['masks'] = masks
91          gt_instances['labels'] = class_ids
92          gt_instances['image_id'] = image_id
93          gt_instances['iscrowd'] = iscrowd_list
94          gt_instances['area'] = areas
95
96
```

```
        if self.transforms is not None:
            img, gt_instances = self.transforms(img, gt_instances)

        return img, gt_instances


    def __len__(self) -> int:
        return len(self.ids)


    def annToRLE(self, ann, height, width):
        """
        Convert annotation which can be polygons, uncompressed RLE to RLE.
        :return: binary mask (numpy 2D array)
        """
        segm = ann #ann['segmentation']
        if isinstance(segm, list):
            # polygon -- a single object might consist of multiple parts
            # we merge all parts into one mask rle code
            rles = maskUtils.frPyObjects(segm, height, width)
            rle = maskUtils.merge(rles)
        elif isinstance(segm['counts'], list):
            # uncompressed RLE
            rle = maskUtils.frPyObjects(segm, height, width)
        else:
            # rle
            rle = ann['segmentation']
        return rle

    def annToMask(self, ann, height, width):
        """
        Convert annotation which can be polygons, uncompressed RLE, or RLE to binary mask.
        :return: binary mask (numpy 2D array)
        """
        rle = self.annToRLE(ann, height, width)
        m = maskUtils.decode(rle)
        return m
```

Listing A.1. code for defining dataset

```

import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor


def get_instance_segmentation_model(num_classes):
    # load an instance segmentation model pre-trained on COCO
    model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)

    # get the number of input features for the classifier
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    # replace the pre-trained head with a new one
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    # 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
    # and replace the mask predictor with a new one
    model.roi_heads.mask_predictor = MaskRCNNPredictor(
    in_features_mask,
                                                       hidden_layer,
                                                       num_classes)

    return model
```

Listing A.2. code for loading model

```
model = get_model_instance_segmentation(num_classes)

# move model to the right device
model.to(device)

# construct an optimizer
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005,
                            momentum=0.9, weight_decay=0.0005)
# and a learning rate scheduler
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                               step_size=3,
                                               gamma=0.1)
```

Listing A.3. code for optimizer and learning rate setup

```
class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(

            # input is (nc) x 128 x 128
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),

            # input is (nc) x 64 x 64
            nn.Conv2d(ndf, ndf*2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf) x 32 x 32
            nn.Conv2d(ndf*2, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*2) x 16 x 16
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*4) x 8 x 8
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*8) x 4 x 4

            nn.Conv2d(ndf * 8, ndf * 16, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 16),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*16) x 2 x 2

            nn.Conv2d(ndf * 16, 1, 2, 1, 0, bias=False),
            nn.Sigmoid()
        )

```

```
    def forward(self, input):
        return self.main(input)
```

Listing A.4. code for discriminator model

```
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)
```

Listing A.5. weights init for discriminator

```
for epoch in range(epoch, num_epochs+epoch):
    # For each batch in the dataloader
    for k, data in enumerate(data_loader_test, 0):

        # reset errors at each image iteration
        errD_masks = 0
        errG_tot = 0
        # get the image
        tensor_image = data[0][0]
        # get the corresponding gt boxes
        gt_boxes = data[1][0]["boxes"].cpu().numpy()
        # get the corresponding gt masks
        gt_masks = data[1][0]["masks"]
        tensor_gt_labels = data[1][0]["labels"]
        ## -----------------------
        # generate mask predictions using the Mask-RCNN model
        model.eval() # put the model in evaluation mode
        with torch.no_grad():
          prediction = model([tensor_image.to(device)])
        ## IMPORTANT restore the model in training mode
        model.train()
        # -------
        pred_masks = prediction[0]["masks"] # tensor predicted masks
        # save predictions and extract each corresponding boxes
        pred_boxes = prediction[0]["boxes"].cpu().numpy()
        # save predicted scores and labels
        pred_labels = prediction[0]["labels"].cpu().numpy()
        pred_scores = prediction[0]["scores"].cpu().numpy()
```

Listing A.6. data extraction from dataset images

```
# ################# Compute Matches ########################
# convert image tensor in PIL image
image = Image.fromarray(tensor_image.mul(255).permute(1, 2, 0).byte()
    .numpy())
# get the corresponding gt numpy masks
gt_numpy_masks = data[1][0]["masks"].mul(255).permute(1, 2, 0).cpu().
    numpy()
## -----------------------
pred_numpy_masks = torch.squeeze(prediction[0]["masks"], dim=1).
    permute(1, 2, 0).cpu().numpy()
## --------------------------------
gt_match, pred_match, overlaps = compute_matches(gt_boxes,
    tensor_gt_labels, gt_numpy_masks,
                                                  pred_boxes,
    pred_labels, pred_scores, pred_numpy_masks)
valid_gt_match = [] # initialize list before every iteration to avoid
     errors
valid_gt_elems = []
```

```
index = 0 # initialize i before every iteration to avoid errors
elem = None # initialize i before every iteration to avoid errors
for index, elem in enumerate(gt_match):
  if np.int(elem) != -1:
    valid_gt_match.append(np.int(elem))
    valid_gt_elems.append(index)
```

Listing A.7. code for computing matches between GT and prediction

List of figures

- Figure 1.1, source: https://www.researchgate.net/figure/Object-detection-in-a-dense-scene_fig4_329217107
- Figure 1.2a, source: https://medium.com/swlh/object-detection-and-instance-segmentation-a-detailed-overview-94ca109274f2
- Figure 1.2b, source: https://medium.com/swlh/object-detection-and-instance-segmentation-a-detailed-overview-94ca109274f2
- Figure 2.1, source: https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46
- Figure 2.2, source: https://www.researchgate.net/figure/General-framework-for-classification-of-mask-proposals-techniques_fig5_342670444
- Figure 2.3, source: https://www.researchgate.net/figure/The-structure-of-the-Mask-R-CNN-architecture_fig2_337795870
- Figure 2.4, source: https://media.springernature.com/lw685/springer-static/image/art%3A10.1007%2Fs13735-020-00195-x/MediaObjects/13735_2020_195_Fig4_HTML.png
- Figure 2.5, source: https://link.springer.com/article/10.1007/s13735-020-00195-x
- Figure 2.6, source: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
- Figure 2.7, source: https://towardsdatascience.com/fast-r-cnn-for-object-detection-a-technical-summary-a0ff94faa022
- Figure 2.8, source: https://www.researchgate.net/figure/High-level-diagram-of-Faster-R-CNN-16-for-generic-object-detection-2-Inception-v2-The_fig3_334987612
- Figure 2.9, source: https://www.mdpi.com/1424-8220/20/4/1010/htm
- Figure 2.10, source: https://becominghuman.ai/reading-panet-path-aggregation-network-1st-place-in-coco-2017-challenge-instance-segmentation-fe4c985cad1b
- Figure 2.11, source: https://towardsdatascience.com/single-stage-instance-segmentation-a-review-1eeb66e0cc49

- Figure 2.12, source: https://github.com/youngwanLEE/CenterMask
- Figure 2.13, source: https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/
- Figure 2.14, source: https://towardsdatascience.com/an-end-to-end-introduction-to-gans-bf253f1fa52f
- Figure 2.15, source: https://phillipi.github.io/pix2pix/
- Figure 2.16, source: https://junyanz.github.io/CycleGAN/
- Figure 2.17, source: https://medium.com/@mrgarg.rajat/implementing-stackgan-using-keras-a0a1b381125e
- Figure 2.18, source: https://heartbeat.fritz.ai/image-super-resolution-using-generative-adversarial-networks-86283cd29d28
- Figure 2.19, source: https://images.app.goo.gl/FvNAhj9RLFGMU22j7
- Figure 3.2, source: https://towardsdatascience.com/computer-vision-instance-segmentation-with-mask-r-cnn-7983502fcad1
- Figure 3.3, source: https://medium.com/@attyuttam/generative-adversarial-networks-using-pytorch-3ad31cc61ac8
- Figure 3.4, source: https://medium.com/@attyuttam/generative-adversarial-networks-using-pytorch-3ad31cc61ac8
- Figure 3.6, source: https://medium.com/ai-society/gans-from-scratch-1-a-deep-introduction-with-code-in-pytorch-and-tensorflow-cb03cdcdba0f
- Figure 3.5, source: https://www.cs.umd.edu/~tomg/projects/stable_gans/
- Figure 3.7, source: https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/
- Figure 3.8, source: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789136678/6/ch06lvl1sec48/architecture-of-stackgan
- Figure 3.9, source: https://www.hudi.it/2019/09/05/deepfake-perche-ne-sentirete-parlare-e-come-difendersi/
- Figure 4.1, source: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a
- Figure 5.1, source: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
- Figure 5.2, source: https://dev.to/jemaloqiu/learn-julia-12-logistic-regression-403g
- Figure 6.2, source: https://github.com/TillBeemelmanns/cityscapes-to-coco-conversion

Bibliography

[1] Association for safe international road travel: https://www.asirt.org/

[2] Goodfellow, Ian and Pouget-Abadie, Jean and Mirza, Mehdi and Xu, Bing and Warde-Farley, David and Ozair, Sherjil and Courville, Aaron and Bengio, Y.. (2014). Generative Adversarial Networks. Advances in Neural Information Processing Systems. 3. 10.1145/3422622.

[3] Lin, Tsung-Yi & Maire, Michael & Belongie, Serge & Hays, James & Perona, Pietro & Ramanan, Deva & Dollár, Piotr & Zitnick, C.. (2014). Microsoft COCO: Common Objects in Context.

[4] Cordts M, Omran M, Ramos S, Rehfeld T, Enzweiler M, Benenson R, Franke U, Roth S, Schiele B The Cityscapes Dataset for Semantic Urban Scene Understanding. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 27-30 June 2016 2016. pp 3213-3223. doi:10.1109/CVPR.2016.350

[5] Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC, Fei-Fei L (2015) ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision 115 (3):211-252. doi:10.1007/s11263-015-0816-y

[6] Hariharan B, Arbeláez P, Girshick R, Malik J Simultaneous detection and segmentation. In: European Conference on Computer Vision, 2014. Springer, pp 297-312

[7] Hafiz, Abdul, Bhat, Ghulam. (2020). A survey on instance segmentation: state of the art. International Journal of Multimedia Information Retrieval. 9. 10.1007/s13735-020-00195-x.

[8] Girshick R, Donahue J, Darrell T, Malik J 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

[9] Bai M, Urtasun R Deep Watershed Transform for Instance Segmentation. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 21-26 July 2017 2017. pp 2858- 2866. doi:10.1109/CVPR.2017.305

[10] Pinheiro PO, Collobert R, Dollar P (2015) Learning to Segment Object Candidates.1990–1998

[11] Dai J, He K, Li Y, Ren S, Sun J Instance-sensitive fully convolutional networks. In: European Conference on Computer Vision, 2016. Springer, pp 534-549

[12] Chen X, Girshick R, He K, Dollár P (2019) TensorMask: A Foundation for Dense Object Segmentation. arXiv preprint arXiv:190312174

[13] Girshick R Fast R-CNN. In: 2015 IEEE International Conference on Computer Vision (ICCV), 7-13 Dec. 2015 2015. pp 1440-1448. doi:10.1109/ICCV.2015.169

[14] Ren S, He K, Girshick R, Sun J (2017) Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (6):1137-1149. doi:10.1109/TPAMI.2016.2577031

[15] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2980-2988, doi: 10.1109/ICCV.2017.322.

[16] Bienias, Lukasz and n, Juanjo and Nielsen, Line and Alstrøm, Tommy. (2019). Insights Into The Behaviour Of Multi-Task Deep Neural Networks For Medical Image Segmentation. 1-6. 10.1109/MLSP.2019.8918753.

[17] Sande KEAvd, Uijlings JRR, Gevers T, Smeulders AWM Segmentation as selective search for object recognition. In: 2011 International Conference on Computer Vision, 6-13 Nov. 2011 2011. pp 1879- 1886. doi:10.1109/ICCV.2011.6126456

[18] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[19] Liu S, Qi L, Qin H, Shi J, Jia J Path aggregation network for instance segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018. pp 8759-8768

[20] Bolya D, Zhou C, Xiao F, Lee YJ (2019) YOLACT: Real-time Instance Segmentation. arXiv preprint arXiv:190402689

[21] Wang, Xinlong and Kong, Tao and Shen, Chunhua and Jiang, Yuning and Li, Lei. (2019). SOLO: Segmenting Objects by Locations.

[22] Y. Lee and J. Park, "CenterMask: Real-Time Anchor-Free Instance Segmentation," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 13903-13912, doi: 10.1109/CVPR42600.2020.01392.

[23] Youngwan Lee, Joong-won Hwang, Sangrok Lee, Yuseok Bae, and Jongyoul Park. An energy and gpu-computation efficient backbone network for real-time object detection. In CVPR Workshops, pages 0–0, 2019

[24] Goodfellow, Ian. (2016). NIPS 2016 Tutorial: Generative Adversarial Networks.

[25] Karras, Tero & Aila, Timo & Laine, Samuli & Lehtinen, Jaakko. (2017). Progressive Growing of GANs for Improved Quality, Stability, and Variation.

[26] Jin, Yanghua & Zhang, Jiakai & Li, Minjun & Tian, Yingtao & Zhu, Huachun & Fang, Zhihao. (2017). Towards the Automatic Anime Characters Creation with Generative Adversarial Networks.

[27] Isola, Phillip & Zhu, Jun-Yan & Zhou, Tinghui & Efros, Alexei. (2017). Image-to-Image Translation with Conditional Adversarial Networks. 5967-5976. 10.1109/CVPR.2017.632.

[28] Zhu, Jun-Yan & Park, Taesung & Isola, Phillip & Efros, Alexei. (2017). Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. 2242-2251. 10.1109/ICCV.2017.244.

[29] Zhang, Han & Xu, Tao & Li, Hongsheng & Zhang, Shaoting & Huang, Xiaolei & Wang, Xiaogang & Metaxas, Dimitris. (2016). StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks.

[30] G. Antipov, M. Baccouche and J. Dugelay, "Face aging with conditional generative adversarial networks," 2017 IEEE International Conference on Image Processing (ICIP), 2017, pp. 2089-2093, doi: 10.1109/ICIP.2017.8296650.

[31] Ledig, Christian & Theis, Lucas & Huszar, Ferenc & Caballero, Jose & Cunningham, Andrew & Acosta, Alejandro & Aitken, Andrew & Tejani, Alykhan & Totz, Johannes & Wang, Zehan & Shi, Wenzhe. (2017). Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. 105-114. 10.1109/CVPR.2017.19.

[32] Vondrick, Carl & Pirsiavash, Hamed & Torralba, Antonio. (2016). Generating Videos with Scene Dynamics.

[33] Liu, Ming-Yu & Huang, Xun & Yu, Jiahui & Wang, Ting-Chun & Mallya, Arun. (2020). Generative Adversarial Networks for Image and Video Synthesis: Algorithms and Applications.

[34] Canny AI: https://www.cannyai.com/

[35] Yang, Shan & Xie, Lei & Chen, Xiao & Lou, Xiaoyan & Huang, Dong-Yan & Li, Haizhou. (2017). Statistical Parametric Speech Synthesis Using Generative Adversarial Networks Under A Multi-task Learning Framework. 10.1109/ASRU.2017.8269003.

[36] Murphy, K. P. (2012). Machine learning: a probabilistic perspective. Cambridge, MA: MIT Press.

[37] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. J. Mach. Learn. Res. 3, null (3/1/2003), 993–1022.

[38] Reynolds D. (2009) Gaussian Mixture Models. In: Li S.Z., Jain A. (eds) Encyclopedia of Biometrics. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-73003-5_196

[39] Le, Quang & Youcef-Toumi, Kamal & Tsetserukou, Dzmitry & Jahanian, Ali. (2020). GAN Mask R-CNN:Instance semantic segmentation benefits from generativeadversarial networks.

[40] Borui Jiang, Ruixuan Luo, Jiayuan Mao, Tete Xiao, and Yuning Jiang. Acquisition of localization confidence for accurate object detection, 2018

[41] @bookGoodfellow-et-al-2016, title=Deep Learning, author=Ian Goodfellow and Yoshua Bengio and Aaron Courville, publisher=MIT Press, note=http://www.deeplearningbook.org, year=2016

[42] Nair, Vinod & Hinton, Geoffrey. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair. Proceedings of ICML. 27. 807-814.

[43] Maas, Andrew L.. "Rectifier Nonlinearities Improve Neural Network Acoustic Models." (2013).

[44] Radford, Alec & Metz, Luke & Chintala, Soumith. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.

[45] Mirza, Mehdi & Osindero, Simon. (2014). Conditional Generative Adversarial Nets.

[46] Bińkowski, Mikołaj & Donahue, Jeff & Dieleman, Sander & Clark, Aidan & Elsen, Erich & Casagrande, Norman & Cobo, Luis. (2019). High Fidelity Speech Synthesis with Adversarial Networks.

[47] https://github.com/TillBeemelmanns/cityscapes-to-coco-conversion

[48] He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Deep Residual Learning for Image Recognition. 770-778. 10.1109/CVPR.2016.90.

[49] Lin, Tsung-Yi & Dollár, Piotr & Girshick, Ross & He, Kaiming & Hariharan, Bharath & Belongie, Serge. (2016). Feature Pyramid Networks for Object Detection.

[50] Daniel Godoy, Understanding binary cross-entropy / log loss: a visual explanation, https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

[51] https://pytorch.org/

[52] Waleed Abdulla, matterport_maskrcnn_2017, url: https://github.com/matterport/Mask_RCNN

[53] Potdar, Kedar & Pai, Chinmay & Akolkar, Sukrut. (2018). A Convolutional Neural Network based Live Object Recognition System as Blind Aid. 10.13140/RG.2.2.34494.54085.

[54] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang and S. P. Smolley, "Least Squares Generative Adversarial Networks," 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2813-2821, doi: 10.1109/ICCV.2017.304.

[55] Penn-Fudan Database for Pedestrian Detection and Segmentation : https://www.cis.upenn.edu/~jshi/ped_html/#pub1

[56] Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

[57] Ruder, Sebastian. (2016). An overview of gradient descent optimization algorithms.