POLITECNICO DI TORINO

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATIONS

Master of Science in Nanotechnologies for ICTs

Master Degree Thesis

Adversarial Machine Learning against Real-World Attacks on CNN Object Detectors

Supervisors
Prof. Dr. Luca Benini, Prof. Dr. Guido Masera
Dr. Michele Magno, Moritz Scherer

Candidate
Alessandro OTTAVIANO

ACADEMIC YEAR 2019-2020

Acknowledgements

I would like to thank my mother Liliana and my father Claudio, who dealt my educational journey until now with an all-around and essential support. Their advice and steady presence have been a reference milestone for me, and they will in the years to come.
In particular, my twin brother Marco, a soul mate by name and in fact, with whom words are unnecessary as it suffices for us to speak in silence to trigger emotional contact, and my cousin Elisa, with whom the bond is tight as with a sister, for their constant being around me, even if virtually, as when we were a kid's trio.

A special thank to Moritz, whose technical and organizational help has been fundamental in the development of the project, for his constant patience and availability.

Last, a fragment of gratitude to my lifelong friends Manuel and Valerija, family members and whose bond hasn't been loosen despite my frequent absence, and to my Nanotech colleagues (especially: Alberto, Elena, Giovanni, André, Gabriele, Dario and Ernesto), with whom I spent the last two years sharing life in all its flavors. A more deeper thank goes actually to Elena, dear friend, for having revised my work by providing precious suggestions, and for her tenacious closeness during the thesis experience.

Summary

The past few years have witnessed a growing interest in the analysis of Machine Learning models robustness against **Adversarial Attack** examples, i.e. externally injected modifications (distortion or perturbation ρ) to the input of a Neural Network (NN) that are able to pollute the predicted output correctness. It often happens that the adversarial nature of the perturbation ρ is *imperceptible/incognito* with respect to the clean input. This definition has a double meaning: either **(1)** unperceived modifications that affect the whole set of pixel's intensities of the input image or **(2)** spatially constrained distortions which are not yet constrained in terms of pixels intensity. The attacks typically cause a substantial **drop** in the ability of these models to correctly predict the output given a specific input. The latter capability is quantified by exploiting two evaluation metrics: **(1)** mean Average Precision (**mAP**) of the NN model and **(2)** Patch Success Rate (**SR**). The first aims at measuring the drop in Recall or Precision [1] according to the attack's target, i.e. either **misdetection** or **misclassification**. Misdetection is defined as the perturbation capability to prevent a foreground prediction to be assessed by the NN model, while misclassification is defined as the perturbation capability to change the NN model decision from one class to another. The second, i.e. Patch SR, directly counts the number of false negatives [2] at detection time. In the aforementioned context, several issues and open questions about the actual and effective **security** of modern Machine Learning models employed for different tasks, from Speech Recognition to Computer Vision (CV), are introduced. As a matter of facts, there exists a complementary and expanding field of interest which is getting devoted to craft countermeasures for lowering the attack's strength. This set of techniques, known as **Adversarial Defense**, follows different fashions according to the attack threat-model they aim to defend against. The research community has started a process of hierarchical and methodological organization in order to flatten the ground of reference, thus letting them acquire solid coherent results as well as a robust application tool-set.

Adversarial Attacks
There is not a unique way to identify an adversarial attack. In particular, in the field of Computer Vision which is under analysis in this study, several attacks typologies have been developed so far. Fig. 3.2 provides an high level view of them. The entity of the injected distortion ρ is bounded by a specific **norm** between the input image with and without the adversarial perturbation (x and x' respectively), indicated as ℓ_n (n indicates the type of the norm that is employed):

[1]Recall quantifies the total number of relevant elements the network should be able to retrieve while Precision indicates the number of retrieved and correct elements.

[2]it quantifies the number of relevant elements which are not retrieved after inference.

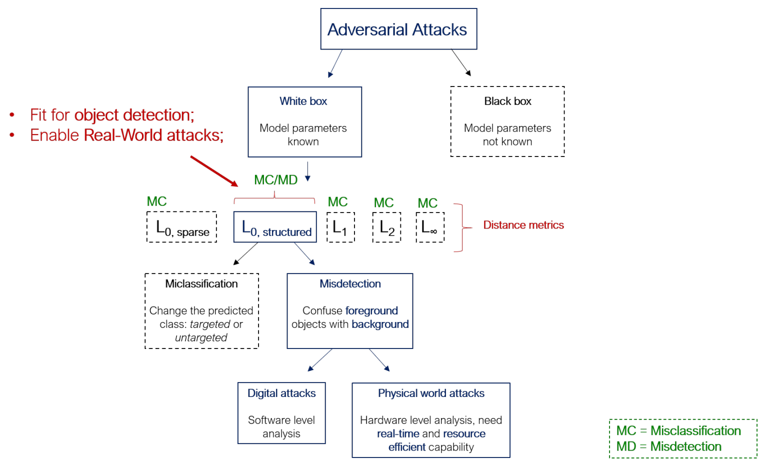
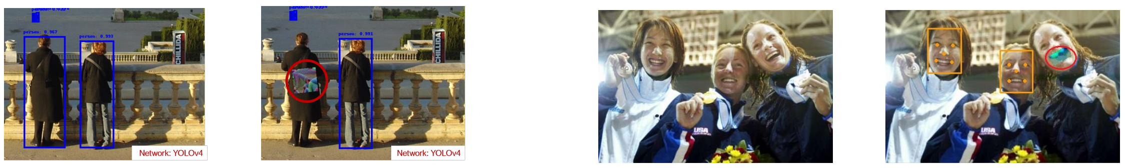

Adversarial Attacks
White box
Model parameters known
Black box
Model parameters not known
• Fit for object detection;
• Enable Real-World attacks;
MC/MD
MC
$L_{0,\text{ sparse}}$
$L_{0,\text{ structured}}$
MC
L_1
MC
L_2
MC
L_∞
Distance metrics
Miclassification
Change the predicted class: targeted or untargeted
Misdetection
Confuse foreground objects with background
Digital attacks
Software level analysis
Physical world attacks
Hardware level analysis, need real-time and resource efficient capability
MC = Misclassification
MD = Misdetection

Figure 1. Adversarial Attacks typologies, high level scheme

$$\text{minimize } \rho \quad \text{s.t.} \quad |x - x'|_n \ \leq \ \rho$$

The norm constraints the adversary ρ by either bounding its **pixel intensity** or its **extension in space**. As already mentioned, the second case does not oblige the pixel intensity to stand within a specified norm-bound and it allows to treat **real-world based** attacks. These find their most suitable model in the **structured ℓ_0 norm** domain [1] [2], which allows to train an **adversarial 'patch'** by applying backpropagation over its pixel set. The training procedure follows the same steps as a standard NN training but it allows to learn the set of pixels which define the perturbation keeping the other network's parameters frozen. The custom loss used during patch training is called **adversarial loss**. It is minimized by employing model's prediction to progressively lower the detection capability of the model itself. The final crafted adversary is applied through **affine transformations** to the target object at testing time and its robustness is evaluated by measuring **Precision-Recall** (`PR`) curves, **Mean Average Precision** (`mAP`) and **Patch SR** as already mentioned. Fig. 2 shows visual examples of inferences in the unperturbed and perturbed domains, performed with `YOLOv4` and `MTCNN` for object and face detection respectively, and highlights the damaging effect labeled as **misdetection**.

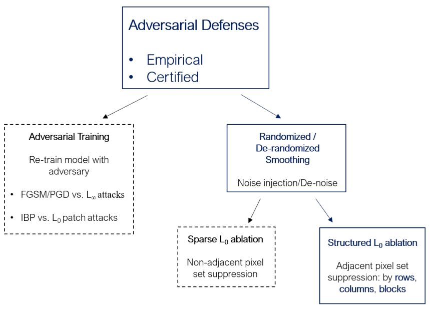


(a) `YOLOv4`, original input (b) `YOLOv4`, perturbed input (c) `MTCNN`, original input (d) `MTCNN`, perturbed input

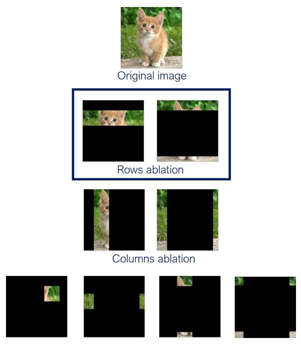
Figure 2. Adversarial patches effect on `YOLOv4` and `MTCNN` detectors.

Adversarial Defenses

Adversarial defenses aim at improving model's robustness by addressing several aspects of the NN domain, from the inner architecture to either training and inference steps. Fig. 3(a) shows an high-level hierarchy.


Adversarial Defenses
• Empirical
• Certified
Adversarial Training
Re-train model with adversary
• FGSM/PGD vs. L_∞ attacks
• IBP vs. L_0 patch attacks
Randomized / De-randomized Smoothing
Noise injection/De-noise
Sparse L_0 ablation
Non-adjacent pixel set suppression
Structured L_0 ablation
Adjacent pixel set suppression: by rows, columns, blocks

(a) Adversarial Defense typologies, high level scheme



(b) Ablation defense full-combinations set

Figure 3. Adversarial Defense domain

The most suited defense against adversarial patch attacks has been identified to be **de-randomized smoothing**, or **ablation** defense, which consists of progressively suppressing input image pixels in a **structured fashion** to generally lower the adversary probability of being sampled at testing time, i.e. 'seen' by the NN model during inference. Fig. 3(b) shows the set of possibilities implied by this kind of defense: ablation by rows, columns or block. The context of the aforementioned defense, in the form described in [3], specifically deals with pure classification domain. Moreover, even though being applied at inference time, de-randomized smoothing needs network **re-training** on ablated samples in order to adapt the model to provide strong performances when a consistent fraction of the input image is suppressed.

Project goals

This project has a double task. **(1)** Evaluate real-world based adversarial attacks effectiveness against object detectors, providing benchmark assessments w.r.t. previous reference works. As already mentioned, this has been done by targeting three families of networks: YOLO, SSD and MTCNN for objects and face detection respectively. **(2)** Select, implement and test an adversarial defense against structured ℓ_0 adversarial attacks in the framework of object detection. The crafted defense is selected within the family of de-randomized smoothing methodologies (ablation defense) and assessed at both **software** and Register Transfer Level (**RTL**) levels. In the following, a granular description of the topics treated in the document's chapters is provided, distinguishing between theoretical background and experimental results.

Theoretical background

A general overview of **traditional** and **convolutional Neural Networks (CNN)** is provided in chapters **1** and **2** respectively. In particular chapter **2** introduces the field of object detection for CV, and distinguishes between two- and **one- stage** detectors, with the latter unlocking **real-time capability**. From there onward, state of the art about adversarial attacks and defenses is addressed in chapters **3** and **5** respectively.

Experimental results

Chapter **4** reports the results obtained by testing the effectiveness of **adversarial patch attacks** targeting **misdetection** against **(a)** state of the art object detectors from the `YOLO` (You Only Look Once), `SSD` (Single Shot Detector) families and **(b)** `MTCNN` (Multitask Cascaded CNN) face detector. Simulations are performed in the framework of **white box digital attacks**, which imply full model's weights knowledge and no extension towards the physical world through patch printing. They led to confirm the already diffused awareness that, in general, a neural network is deceivable even though with heterogeneous degrees of damage according to its architectural structure and complexity. Then, *de-randomized smoothing* by **structured ablation** defense is assessed in chapter **6**. The adopted solution is adjusted to **meet the constraints** introduced in the framework of object detection by addressing the inference phase in the form of an image **pre-processing** step. Benchmark `YOLOv3` is taken as a reference network, re-trained on ablation and its clean performances are checked when **(a)** the perturbation is kept `off`, though the defense is activated (test on unperturbed ablated samples) for different values of **retained fractions** of the input sample at ablation time, either in terms of bounding box regression (spatial localization) and class confidence score predictions and **(b)** the model is attacked by exploiting adversarial patches when the defense is activated. It is shown that the latter can **decrease adversary success rate from 40% to 6%** on the tested dataset (video frames) under attack when switching between the undefended and defended cases respectively. Last, chapter **7** introduces the challenges defined by the new field of Edge Artificial Intelligence (**Edge AI**) with respect to the domain of adversarial Machine Learning. These challenges involve either software and hardware design space exploration to match the constraints of **resource-limited** devices that can not interface with a cloud server. This context involves model compression in terms of internal architecture design and low-precision (quantization), along with the study of efficient, low-power CNN accelerators. In this framework, the pre-processing defense module is implemented as a combinational block in the `RTL` of an existing FPGA-based object detector example flow, simulated and synthesized. The process serves to address the overall resource usage and related **real-time based constraints** w.r.t. multiple-inferences introduced by de-randomized smoothing defense, in order to start the transition from digital to real-world based defenses at **edge computing level**.

Contents

List of Tables

List of Figures

List of Acronyms

CW	Carlini-Wagner
FGSM	Fast Gradient Sign Method
PGD	Projected Gradient Descend
EBR	Embedded Block RAM
STN	Spatial Transformer Network
NPS	Non-printability score
TV	Total Variation
EoT	Expectation over Transformation
NMS	Non-maximum suppression
IoU	Intersection Over Union
FCN	Fully Convolutional Network
RPN	Region Proposal Network
ROI	Region of interest
R-CNN	Region based CNN
FPN	Feature Pyramid Network
SGD	Stochastic Gradient Descent
GD	Gradient Descent
CV	Computer Vision
AI	Artificial Intelligence
ANN	Artificial Neural Networks
ML	Machine Learning
CNN	Convolutional Neural Network
YOLO	You Only Look Once
SSD	Single-Shot Detector
MTCNN	Multi Task Convolutional Neural Network
SOTA	State of the Art
OD	Object Detection
FPS	Frame per seconds

Part I

Machine Learning for Computer Vision

Chapter 1

An overview on classical (D)NNs

1.1 Background

The huge interest that Artificial Neural Networks (ANN) have brought in the past decades come from an interesting - as well as challenging - task: let a machine to gain some of the **intuition** that is often a human-being exclusive. This is the result of an inverted trend that sees machines as able to perform more quickly and precisely tasks that humans find hard to exploit - it is frequently the case when a series of instructions making up a fairly complex algorithm have to be processed - but less capable when it comes to problems that are **not straightforwardly codifiable in a formal and structured way** [4].
The purpose of Artificial Intelligence (AI) is to try to formalize such an intuition-based and informal knowledge to allow machines solving some common problems like speech, gestures and image recognition.

The key element is the way that kind of knowledge is translated into a formal and machine-friendly interpretation. In this framework, a suitable **data representation** plays a fundamental role in the learning process, for machines as well as humans. In particular, each '*essential piece of information* [4]' that is useful for a machine to properly fulfill its task and gain the sought after knowledge - i.e. mapping an input to an output - is called a **feature**. Two paths have been followed in the past (fig. 1.1):

- **Hard-coding** the outside knowledge by re-shaping and re-formulating data in a way that is efficiently interpretable and mappable by machines. This means **hand-crafting** and preparing features that can be quantitatively econded and thereafter interpreted by machines as the teaching medium [5];
- Let the machine doing the process at the previous bullet point **by itself**, i.e. providing it with the **raw data** only and letting it to extract the features that will be used for predicting the output.

The last observation makes a step forward in making the learning flow more autonomous and human-independent: that is the proper domain of Machine Learning [1]. The **learning structures** able to map the input to the output are called **Neural Networks** (NNs).

[1]It takes the name of **supervised learning**, since not only a set of raw inputs, but also the corresponding solutions to the targeted problem - for each input - must still be provided, in order to calibrate the network judgment capability during the learning phase. **Unsupervised learning**, such as *k-means* clustering, does not need a ground truth, yet it could be used to prepare the reference itself for supervised learning in a previous step.

Features extraction creates a set of intermediate - or **hidden** - steps that are located between the raw input and the mapped output, as the floors in a skyscraper. In particular, intermediate steps could be organized:

- in a way to provide **non-tied and fully connected feature information** when moving from the input to the output, building a 'flatten' architecture.
  That is the case of classical ANN developed before 2012 [6] and referred to as **traditional NNs**.

- in a **hierarchical** way, such that primitive and essential elements of the problem to solve, nearly detached from the context of the input, are isolated first - near the input - while more high-level features are extracted afterwards - near the output - relying on the previous pieces of knowledge (**tied and sparse feature information**).
  That has been the transition phase towards today's State of the Art (SOTA) NNs called Convolutional Neural Networks (CNNs).

The number of hidden steps - the skyscraper's floors - determines the depth of the learning structure (the height of the building).
Deep Neural Networks is merely the name assigned to NNs of both types when a larger number of hidden layers is introduced than the original and classical Machine Learning domain of the early age was used to.

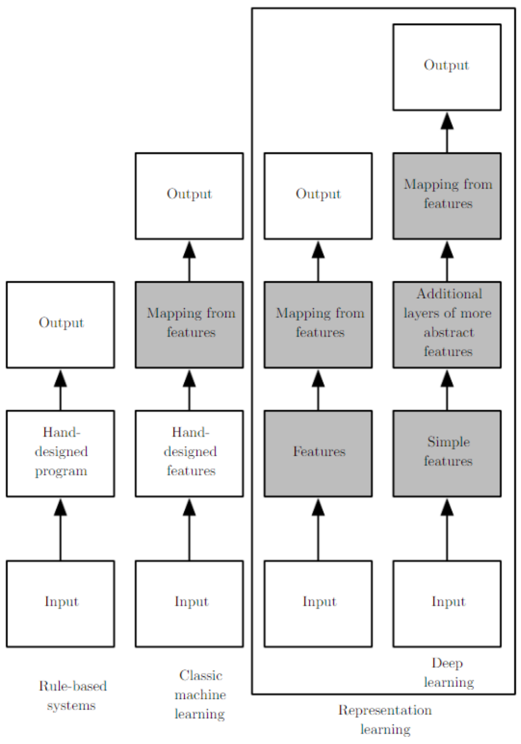


Figure 1.1. Machine Learning (ML) conceptual evolution [4]

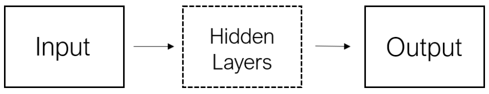
1.2 Traditional NNs building blocks

Traditional ANN that are treated in this document do not involve feedback loops that link some intermediate steps back to the input: they are called **feedforward** ANN, while their counterpart is named **recurrent** [2] (or with feedback) ANN [4]. Therefore, the adjective will be implied even if not explicitly indicated hereafter.

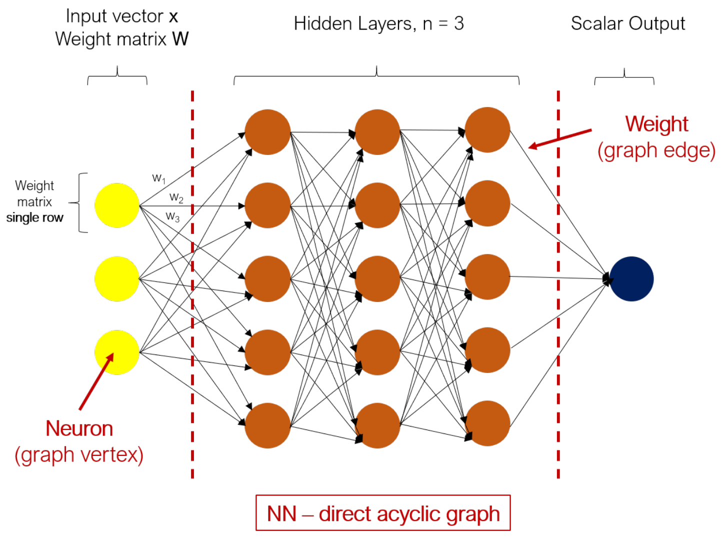
Some steady features exist within a NN internal structure, as well as inside the supervised learning flow - typically called **training phase** - it undertook. They are briefly described below, since the terminology used is recurrent further on in the document.

1.2.1 NN structural composition

The tripartite structure **input - hidden steps - outputs** of a classical NN is reported in fig. 1.2, which represents either system level and layer-level NN structure (with n = 3 hidden steps as an example). Each 'step' is usually referred to as **layer** and the term is adopted here as well.



(a) ANN high-level tripartite structure



(b) ANN detailed fully-connected structure

Figure 1.2. ANN general structure. On the top the system level representation, while on the bottom the layer-level representation.

In the domain of Computer Vision (CV), each entry x_i of the overall vector $\mathbf{x}$ represents **the intensity of a single image pixel**.

[2]As observed in [4], recurrent ANNs play the same role with respect to temporal sequences of data than Convolutional NNs with grids of data, as outlined in chapter 2.

Fig. 1.2 allows to understand the meaning of the term 'Neural Network':

- *Network* because the formal terminology leads to define it as a direct acyclic graph where each input is indeed connected to each output, for every layer;

- *Neural* is inspired to neuroscience, from a conceptual point of view. In fact, each layer is represented as a vector - like the input $\mathbf{x}$ - and its elements play the role of neurons [4].
  Each neuron collects a weighted information from the whole input vector $\mathbf{x}$ according to a specific weight matrix $\mathbf{W}$ and provides a scalar output by taking **(1)** a linear weighted sum followed by **(2)** a non-linear activation.
  The last action is called **firing**. Overall, the process is represented by a chain rule for the overall set of layers:

  $$f(\mathbf{x}) = f^n(\mathbf{W}_n^t \cdot f^{n-1}(\dots f^1(\mathbf{W_1}^t \cdot \mathbf{x} + b_1)) + b_n), \qquad \mathbf{W_j}^t \cdot f_j(\dots) = \sum_i^m w_i^j f(\dots)_i^j$$

  where f^n is the nth layer after activation, $\mathbf{W_n}$ is the nth **weight matrix** and b is the nth bias.

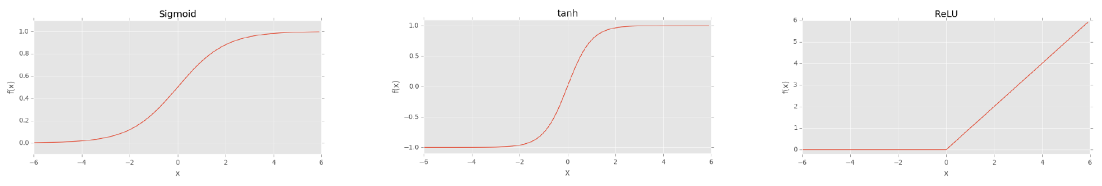
The **firing action introduces the non-linearity** needed by the system in order to be properly taught for solving a particular problem. It is indeed interesting to note, as reported in [4], that **the lack of a non-linearity** would reduce the NN to a single linear transformation, i.e. unable to correctly approximate the majority of non-linear phenomena that happen in the physical world. In a word, the set of n layers would **collapse** to a standalone - and fully linear - one.

The functions used to model the firing behavior are called **activation functions** or **hidden units** [4]. The most intuitive is the Heaviside step. It has the drawback of being non differentiable, a practical obstacle during the learning phase.
Threfore, some commonly employed activations with increasing performance capacity are listed below:

- **sigmoid function (fig. 1.3(a))**: differentiable and continuous everywhere, but not 'zero-centered' [5], as well as asymptotically saturating at $\pm\infty$. Saturation is another unwanted behavior, since it reduces the step-size of the performance gain each neuron is subjected to after every iteration;

- **hyperbolic tangent (fig. 1.3(b))**: similar to the sigmoid, but zero-centered - its value when the input is 0 is exactly zero;

- **Rectified Linear Unit (ReLU) (fig. 1.3(c))**: the well-know **ramp function**, which shares the positive features of its predecessors. Moreover, it **suppresses negative input values** and linearly keeps the positives, preventing saturation.
  Nowadays, it's the **most used activation** function since it outperforms other candidates, and several variants of it have been proposed throughout the years (Leaky ReLU, Exponential Linear Unit (ELU), Parametric ReLU (PReLU);



(a) Sigmoid (b) Hyperbolic tangent (c) ReLu

Figure 1.3. Common activation functions [5]

As a final comment, it is interesting to note that in a traditional NN, as in fig. 1.2, every input sends a piece of information - weighted accordingly - to each output, generating a system that is said to be **fully connected**. This structural feature keeps sealed some useful properties that are instead embraced by Convolutional Neural Network (CNN) topology, as further on specified.

1.2.2 The Learning flow

The process of teaching to a machine in the framework of supervised ML shows a common pattern, i.e. a set of fixed key elements that contribute to build up the learning phase, or machine training. They are listed below and schematically sketched in fig. 1.4:

- Training dataset;
- NN model;
- Cost - or Loss - function;
- Optimization method;

These building blocks are managed by a three-step control flow, named **forward pass**, **backward pass** and **optimization step**, that iteratively make use of them to inject the sought after level of knowledge to the machine.

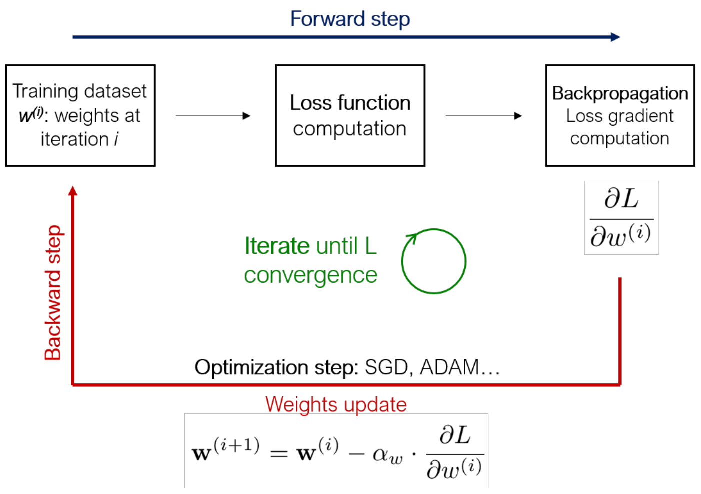

Forward step
Training dataset
w(i): weights at iteration i
Loss function computation
Backpropagation
Loss gradient computation
$\frac{\partial L}{\partial w^{(i)}}$
Iterate until L convergence
Backward step
Optimization step: SGD, ADAM...
Weights update
$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \alpha_w \cdot \frac{\partial L}{\partial w^{(i)}}$

Figure 1.4. NN training process flow - **forward** and **backward** pass

1.2.3 Control: forward pass, backward pass and optimization step

To the extent that an analogy is allowed, a NN model - when dealing with supervised learning - is *equivalent to a student trying to build knowledge around a selected topic by iteratively answering questions coming from outside and receiving a feedback on their correctness slightly after. As time passes, answers accuracy increases due to the trial-and-error strategy.*
Building up on the naive example, and with reference to the scheme in fig. 1.4:

- The action of providing an answer when a question is risen, i.e. of predicting an output given an input, according to the actual level of knowledge is called **forward pass**, or inference. It can be displayed as an arrow pointing to the output from the input;
- The action of comparing the predicted answer with the correct one is named backward pass, or **backpropagation**. It states the entity of the distance - or error - between prediction and ground truth.
- The error itself plays the role of a loss (a penalty, or cost) to be minimized;
- The action of minimizing the loss and distributing the new piece of acquired knowledge back to the entire model (weight update) is referred to as **optimization step**.
  It can be displayed as an arrow pointing back towards the input from the output;

Training dataset

When dealing with supervised learning, the active material upon which the machine knowledge is built is provided by a set of data, each of which bears along the actual solution to the task the machine has to learn and is used to train it accordingly.
In the framework of computer vision, a training dataset consists of a number of input images with information about the objects class and/or geometrical localization if requested. These labels are the reference guideline for the NN model, therefore are also named **ground truth annotations**.

NN model

The choices done around the NN **model architecture** (e.g. the number of hidden layers) affect its performances, and there is no orthodox way, or a prescribed set of rules, to follow in order to fulfill this task, at least when building a model from scratch.
As mentioned in chapter 7, some techniques have been recently developed to **automate** the search for an optimal configuration (`NAS` - Neural Architecture Search) when several constraints have started rising - mostly concerning the model **size** and **inference time** - in the framework of machine learning hardware deployment.

Optimization method and Cost function

The **loss function** (or objective function) quantifies the error of the prediction. It may have several weighted contributions according to the specific problem to be solved, as outlined in chapter 2 with reference to the CV domain.
The cost function has to be minimized to favor the learning step [3]. In order to do that, **the gradient** of the loss function with respect to the NN weights is calculated by **applying the**

[3]Notice that, in this way, the model optimization happens indirectly w.r.t. the cost function optimization. In other words, to solve an optimization problem, another one is addressed [4].

backpropagation (or backprop) chain-rule [7].
This gradient quantifies the sensitivity of the change the loss function undertakes when a corresponding change to the input weights (matrix **W**) and bias (b) is performed. Given a neuron z at (i+1)th layer, its output comes from a two-step process:

1. weighted sum calculation from the activated neuron output from the previous layer:
$$\mathbf{z}^{i+1} = \mathbf{W^i} \cdot f(\mathbf{z}^i) + b$$
where f(·) **is the chosen activation**;
2. Activate the (i+1)th neuron through f(·):
$$a^{i+1} = f(\mathbf{z}^{i+1})$$

Therefore, considering the jth weight and bias in the ith layer as highlighted in fig. 1.5, the gradient loss is expressed as in eq. 1.1:

$$\nabla L = \frac{\partial L}{\partial w_{(i,j)}} w_{(i,j)} + \frac{\partial L(\mathbf{x})}{\partial b_{(i,j)}} b_{(i,j)} \tag{1.1}$$

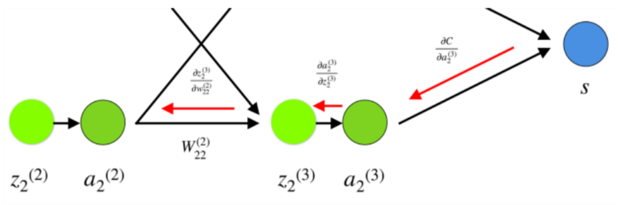

$\frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}}$
$\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}$
$\frac{\partial C}{\partial a_2^{(3)}}$
s
$W_{22}^{(2)}$
$z_2^{(2)}$
$a_2^{(2)}$
$z_2^{(3)}$
$a_2^{(3)}$

Figure 1.5. Backpropagation steps on single entry [8]

The chain-rule allows to compute:

$$\frac{\partial L}{\partial w_{(i,j)}} = \frac{\partial L}{\partial a^{(i+1,j)}} \cdot \frac{\partial a^{(i+1,j)}}{\partial \mathbf{z}^{(i+1,j)}} \cdot \frac{\partial \mathbf{z}^{(i+1,j)}}{\partial w^{(i,j)}} = \frac{\partial L(\mathbf{x})}{\partial a^{(i+1,j)}} \cdot \frac{\partial a^{(i+1,j)}}{\partial \mathbf{z}^{(i+1,j)}} \cdot a^{i,j} \tag{1.2}$$

and

$$\frac{\partial L}{\partial b_{(i,j)}} = \frac{\partial L}{\partial a^{(i+1,j)}} \cdot \frac{\partial a^{(i+1,j)}}{\partial \mathbf{z}^{(i+1,j)}} \cdot \frac{\partial \mathbf{z}^{(i+1,j)}}{\partial b^{(i,j)}} = \frac{\partial L(\mathbf{x})}{\partial a^{(i+1,j)}} \cdot \frac{\partial a^{(i+1,j)}}{\partial \mathbf{z}^{(i+1,j)}} \cdot 1 \tag{1.3}$$

The term
$$\delta^{(i,j)} = \frac{\partial L}{\partial \mathbf{z}^{(i,j)}}$$
is called **local gradient** or error related to neuron jth in layer ith.

The obtained gradients for weights and biases bear the **direction towards which the minimizer [4] is updated**, according to the simplest optimization method upon which all the others

[4]Given a function f(x), a point x* is a **global minimizer of f** if f(x*) ≤ f(x) ∀ x [9]

are built, Gradient Descent (GD). The minimizers are the weights and biases, thus the update rule reads:

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \alpha_w \cdot \frac{\partial L}{\partial w^{(i)}}; \qquad \mathbf{b}^{(i+1)} = \mathbf{b}^{(i)} - \alpha_b \cdot \frac{\partial L}{\partial b^{(i)}} \tag{1.4}$$

where α [5] is called **learning rate**. It quantifies the step taken by the gradient of the loss function. It takes the name of **hyperparameter**, and its optimal choice does influence the performace of the optimization step, i.e. the learning efficiency.
During the training, the hyperparameters are fixed and the learning procedure is performed on a first dataset, called training set. A second dataset is used right after the training to test the effective accuracy of the model after the weights update. The latter process is called **validation**, while the name **validation dataset** is assigned to the corresponding ensemble of input data. Note that training and validation phases, as well as their dataset, are detached and independent during the machine's learning flow.

The aforementioned optimization is termed **gradient-based learning**, since it seeks for the minimum of the cost function, or - equivalently - to the set of weights that minimizes it by moving towards the direction dictated by the gradient and learning rate.
If the used model is linear, the corresponding loss function is convex and the convergence towards the **global minimum** of L is guaranteed:

*When a function **f is convex**, any local minimizer x^* is a global minimizer of f. If, in addition, f is differentiable, then any stationary point x^* is a global minimizer of f [9]*

Conversely, in a NN the non-linearity introduced by the hidden units makes the **loss a non-convex function**. Therefore, its true global minimum can be only approximated when iteratively applying the optimization method by trying to find the nearest **local** one.

Optimization methods are a domain of research by themselves, and some of the most common are briefly overviewed in Appendix A.
From the experience gained with this work and previous academic courses, actual benchmarks optimization algorithms are **Adam** [10] (from adaptive moment estimation), which autonomously tune the learning rate upon convergence and **Stochastic Gradient Descent (SGD) with Nesterov momentum acceleration**, that requires manual learning rate tuning instead.

Regularization

Few words are going to be spent on a crucial concept, i.e. **regularization**. It is defined as the set of techniques that helps the trained model to **generalize its performances** towards unseen targets, i.e. to assess the correct prediction when fed with inputs **it was not trained on** [4] [5]. The set of these inputs is called **testing dataset** [6]. Regularization can exist in several forms:

- **Explicit regularitazion:** [11] defines it as the set of techniques that would introduce regularization by acting on external parameters.

[5]Usually, the bias is included into the weight matrix **W** by adding a column of 1s [5]. This prevents using a separate variable, with a unique learning rate.

[6]Work [11] associates a network with high generalization capability as one having '*small generalization error*', which is the difference between training and validation error. While regularization is effective in improving generalization performances, it's claimed '*it's not necessary nor sufficient*' in controlling the generalization error

  - **Weight decay:** it affects training phase and introduces an additional contribution to the loss function. It addresses directly the weight matrix by taking either ℓ_2 norm (also called weight decay), ℓ_1 norm or a combination of both on its entries.
    In principle, this should improve the accuracy at testing time (avoiding over- and under-fitting [7]) while slightly damaging that at training time;
  - **Architectural regularization:** it involves changing the network architecture at training time by dropping some neuron connections, thus avoiding bounding a single neuron towards a specific input. This reduces the risk of over-fitting. A layer of this kind is called **Dropout layer**;
  - **Data Augmentation:** it consists of a set of techniques targeting input pre-processing before it is fed to the neural network. This helps the model to **generalize towards a larger set of possibilities** and situations.
    Modern NNs for CV employ several data augmentation techniques at pre-processing time, which are gaining growing importance - see [12] as mentioned further down in the text.

- **Implicit regularization:**
  - **Early stopping:** another method implied at training time to prevent overfitting. It sets up a **patience** variable after a certain event is **triggered**, e.g. **(1)** the network performances on the validation dataset (e.g. validation loss) start decreasing **(2)** performances keep themselves constant starting from epoch i with respect to epoch *i-1* **(3)** all of these behaviors happen in average over an ensemble of epochs. After the patience has passed, the training is automatically stopped.
  - **Batch Normalization:** in order to keep the output of each hidden units to have the **same mean μ and variance σ** (thus reducing the so called internal covariance shift [13]), the latter parameters are computed after each activation layer and applied to the activated neuron so that $z_{in}^{i+1} = \frac{f(z^i)-\mu}{\sigma}$.
    μ and σ are said to be running mean and variance, and contribute improving the generalization performances.

A comprehensive and detailed overview of regularization techniques is given in in chapter 7 of [4].

As a **last note**, the training procedure executes in an **iterative manner** until the cost function reaches the convergence. Given a set of input training data, e.g. images for CV, they are **grouped in batches** according to a user-specified **batch size** (its value is usually multiple of 8 depending on computational capabilities).
Each batch defines a portion of teaching material, which is considered for the evaluation of the gradient, and sets up an optimization step. After each of them has been processed and the whole input dataset exploited, **an `epoch` has passed**. The cost function value after one epoch is usually taken by averaging over the set of post-batch optimizations results, for all batches.
The epoch allows to introduce a **measure in terms of training time** and states how many times the system has faced the whole input dataset for learning purposes.

[7]Overfitting: the model produces good performances on the training dataset only, failing to generalize towards other input ensembles. Underfitting: the model performs poorly on the training dataset and thus fails to generalize as well.

1.3 Summary

This chapter briefly outlined the general background behind Neural Networks for system-level understanding.
It highlighted the fundamental processes and elements that are needed to `train` and `test` a neural network model in the framework of feed forward supervised Machine Learning, i.e. **gradient-based learning**.
Key elements in this framework are **(1)** automatized **feature extraction** process along with **(2) training dataset**, **(3) cost function** and **(4) model architecture** choices, which serve as tools for the general **forward** and **backward** passes upon which machine knowledge is built.

Chapter 2

Convolutional Neural Networks

As mentioned in the previous chapter, in a traditional ANN **each neuron receives activated inputs from every neuron of the previous layer**. It is said that the layer is **fully connected** (`FC`). This feature dramatically increases the number of parameters of the final ANN model, slowing down the training time.
In particular, it makes it difficult to treat real-world based problems such as **classification** and **object detection** in the framework of CV.

This chapter initially overviews the **structural composition** of CNNs and their advantages when addressing specific **grid-like data types** such as images. Thereafter, it describes their extension towards CV classification and object detection in the recent years (starting 2012).

2.1 A powerful paradigm for Computer Vision

CNN replace traditional matrix-vector multiplication with the **convolution** operation.
Given two functions f and g, the convolution between them is defined as [14]:

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)\, g(t-\tau) d\tau \tag{2.1}$$

that is the area under the intersection curve between $f(\tau)$ and $g(-\tau + \text{t})\ \forall$ t. Function g is reversed and shifted before calculation, as shown in fig. 2.1.

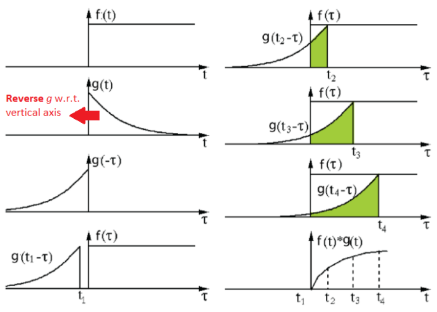


Figure 2.1. Convolution operation between f and g, visual representation [14]

Actually, another operation which does not involve reversing g exists and it is called **cross-correlation**:

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)\, g(t + \tau) d\tau \tag{2.2}$$

The latter is the operation employed by CNN models, although the misleading eponymus 'convolutional' by which they are known. From now on, the term convolution will be used to indicate the actual 'Machine Learning' definition, which implies the mathematical operation of cross-correlation.
As from eq. 2.2, convolution between f and g is an integral in the time domain, yet it becomes an ***element-wise* multiplication between the Fourier transforms of *f* and *g*** in the frequency domain [4].
The latter is the way CNN will be described hereafter, since it encodes highly visual representation in the framework of ML applications, facilitating the final understanding.

Provided the following definitions

- **Function *f***: raw input, grid-like shape. In the CV domain, it is an **image data**, i.e. a 3-D tensor defined by three parameters: image width (`W`), image height (`H`) and number of color channels, typically 3 for `RGB` format and 1 for `grayscale`. Overall, it has dimension `(3 × H × W)` [1]

- **Function *g***: channel **kernel**. Its dimensions are smaller than the input's, which brings the most important gain and source of advantage of CNN with respect to traditional NNs.
The kernel encodes the weights to be learnt and updated through the optimization step, in analogy to a traditional NN. This will be clearer further on in this chapter.

- **Convolution output *f*g***: it usually takes the name of **output feature**. In the domain of CV, it encodes fine details of the input image, such as edges, corners or shapes (blobs).

the primary advantages of CNN, which stand behind its success and SOTA adoption w.r.t. traditional NNs, can be identified as [4] [5]:

1. **Sparse interaction** As already highlighted, the main difference between a CNN and a traditional NN is encoded in the absence of full connections between each layer's neuron. This property is called **sparse interaction** [4] or, equivalently, each layer is said to have **sparse weights**.
With reference to the graphs in fig. 2.2(a), sparsity can be visualized by considering that the kernel **interacts with a small portion of the input tensor**. Kernel size is said to define the **receptive field** of the convolution layer.
Note that though being the connections sparse, neurons in the deep layers of the network still have **indirect interactions** with the input layers.

   The main advantage is a **critical reduction of the model total parameters**, since the number of connections is reduced by the receptive field of the applied kernel.

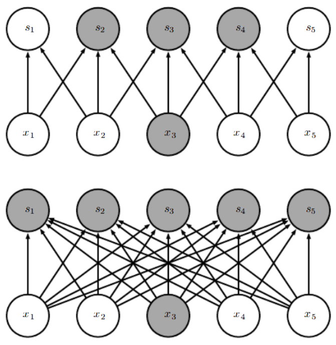
2. **Parameters sharing** In a convolutional network, the kernel slides along the input and performs an operation at each sliding step. In this way, different portions of the output are

[1]This notation is called 'channel first'. The other one 'channel last' reads `(H × W × 3)`

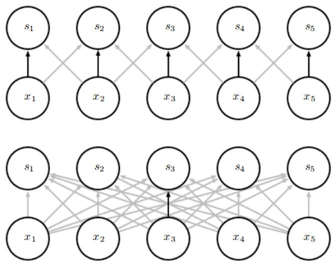
still generated by different portions of the input, as it would be reasonable to imagine, but **against the same applied kernel**, which is 'shared' as the sliding process moves forward. In this framework, kernel entries are called **tied weights**, as represented in fig. 2.2(b) (*Top*), where each bold arrow indicates a multiple contribution coming from the same kernel's element on different input entries. This behaviour is not present in a traditional NN - fig. 2.2(b) (*Bottom*) - where each kernel element does have effect on one input's entry only (single bold arrow);

3. **Equivariant representation, or local invariance against translations** [2] It is the property that allows the extracted features to follow translations concerning the original input's objects.
According to local invariance defintion, **convolving over a translated input is equivalent to translating the convolved input**.

4. **Compositionality** The property of *hierarchical* learning: detailed and context-detached features can be generated early in the network (*shallow* layers), and used to provide more detailed - and complex - ones in the deeper layers.
In the domain of CV, this means that object shapes are addressed from single corners or edges.



(a) Sparse connectivity



(b) Parameters sharing

Figure 2.2. Sparse connectivity and parameters sharing properties. The comparison involves CNN (*Top*) and traditional NNs (*Bottom*) [4]

2.2 Microscopic CNN structural building blocks

A CNN is usually built by **stacking several layers** that manipulate the input and provide, as output, the actual network predictions, whose nature varies according to the task being undertaken.
In this context, some layers provide the sought after convolution, while others are borrowed from traditional NNs architecture, as explained below.

[2] f is said to be equivariant to g if $f(g(\dots)) = g(f(\dots))$ [4]

CONV layer

Given an input grayscale image tensor I of size `(1` $\times$ `H` $\times$ `W)` and a **kernel** K of size `(1` $\times$ K_h $\times$ K_w`)`, with K_h `<` H and K_w `<` W [3] as considered in fig. 2.3(a), the convolution flow is defined by:

1. Align the kernel's center with the first (top-left) element - or pixel - of the input image;

2. Perform **element-wise multiplication** between I and K;

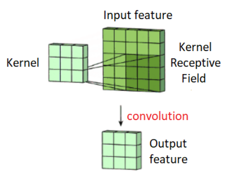
3. Sum the results; the output is the first element of the convolution tensor between I and K;

4. Slide the kernel K by a certain amount (**stride S**, which is chosen by the user himself) along the image width;

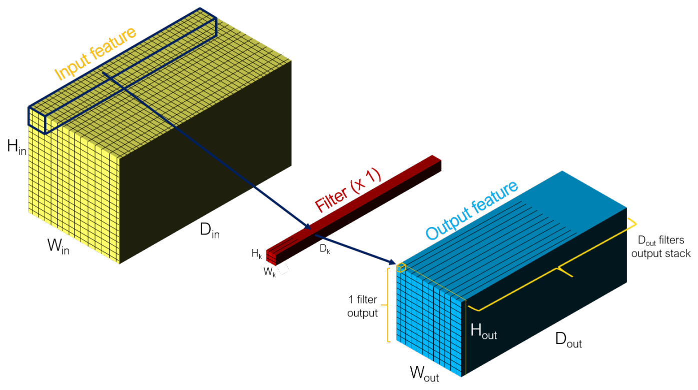
5. Repeat from step 1, until the whole set of input I entry pixels is covered;

In order to preserve the input dimensions after convolution, the latter may be padded (e.g. with zeros) so that pixels at the edges of the input tensor may undertake convolution as well: if not, the operation can't be applied because of the impossibility to perform element-wise multiplication. The value that unlocks this operation takes the name of **padding P**.

If the input tensor has depth D_{in}, a specific number of kernels (here indicated with K_{tot}) is usually exploited. The previous algorithm applies identically for each of them, i.e. it involves input I at i^{th} depth layer with i^{th} kernel K_i, $\forall$ i. The whole operation is called **2D convolution**. The overall structure of size $(K_{tot} \times H_K \times W_K)$ is called **convolution filter**, which is a collection of kernels, as shown in fig. 2.3(b). [4]



(a) Input tensor and kernel reception field [14]



(b) 3D Convolution operation

Figure 2.3. Convolution operation visual representation

[3]The kernel is usually a square matrix whose side is odd to preserve the existence of a center. That side, or **receptive field F**, quantifies the resulting level of sparsity after convolution

[4]Note that for the very first convolution involving the input image, the filter has K_{tot} = 3 for `RGB` (D_{in}=3) format and K_{tot} = 1 for `grayscale` (D_{in}=1), where the filter collapses to a single kernel as described above.

It is worth noticing, as mentioned previously, that the **elements building up each tensor filter are the actual weights that have to be updated using the optimization rule** as well as the first responsible of the learning process.

As a matter of fact, the resulting convoluted tensor has dimensions [5]:

- $W_{out} = (W_{input} - F + 2P)/S + 1$
- $H_{out} = (H_{input} - F + 2P)/S + 1$
- $D_{out} = K_{tot}$

Several types of convolutions have been proposed throughout the years. Among them, two are getting popularity because of their capability to keep model final size compact (chap. 7): **depth-wise** and **pointwise** convolutions [15].
Briefly, they are the result of factorizing a standard convolution process into two operations, the first applying D_K filters of depth 1 $(D_K \times H_K \times W_K)$ and the second D_{out} filters of unit dimensions $(D_{out} \times 1 \times 1)$, providing a **sensible computation footprint reduction**. The operation is called **depthwise separable convolution**.
Separable convolution is the main reason of the growing popularity `MobileNet` family has gained in the recent years in the field of Edge ML.

`POOL` layer
Pooling operation consists of '*replacing the output of a layer with a summary statistics of its neighborood elements*' [4]. The overall process is similar to that of a standard `CONV` layer: a kernel of reduced dimensions with respect to the input tensor slides along its columns and rows by a fixed stride.
The operation that is performed can either be the average of the input pixels intensities in the receptive field of the kernel (**average pooling**), or their maximum (**max pooling**, shown in fig. 2.4).

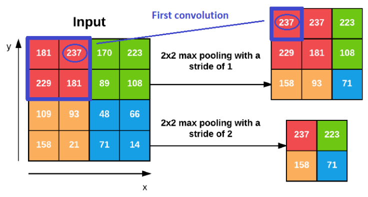


Figure 2.4. Max pooling operation with different strides [5]

It is interesting to note that pooling helps the network model to **build invariance against small input translations**. The reason behind this statement can be understood from fig. 2.4: even though the original set of pixels is slightly translated towards right (that can correspond to an actual movement of an object in a image), **the maximum operation still outputs the same result in both cases**.
This is an advantage when the exact location of some features is less important, for example in a typical pure image classification problem.

Activation, DROPUT, BATCHNORM and FC layers
Other than convolution-based layers (both CONV and POOL), other blocks are usually needed to either avoid overfitting as **explicit and implicit regularizers** (DROPOUT and BATCHNORM) or exploit the required non-linearity via **neuron activation** (benchmark: ReLU layer). They have all been already described in chapter 1.
Finally, one to two FC layers were typically inserted before the model output, although several **benefits exist in having fully convolutional models** and their use is nowadays seldom.

CNN as particular forms of traditional FC NNs
A final comment to underline a quite interesting and peculiar point of view on CNN, as reported in [4]. For each position of the filter with respect to the input tensor during the convolution operation, the operative region is dictated by the filter's receptive field.
The latter can be thought of as a tensor having the same size of the input with a set of **weights (tensor entries) hard-coded to zero**. This property, along with the **slide-and-convolve operation** which keeps the weights constant as the sliding proceeds, is '*similar to imposing a* ***prior probability distribution with infinite strength***' [4] on a traditional, FC, NN model [5], since some operation's features are known *a priori.*

2.3 High-level CNN structure

The elements described in the previous section constitute the 'microscopic' building blocks, or fundamental 'bag of ingredients', when designing a CNN model. Their use has become SOTA in such ML fields like **image classification**, starting from the seminal paper describing AlexNet [6] in 2012 [6].
From then on, research has evolved by introducing several novel modules in order to [12]:

- **Improve network performances**, i.e. model accuracy, for the task it has been trained for;
- **Reduce network inference time**, i.e. the time needed to output a prediction starting from an input feed;
- Control the **model size** to fit and support the '**shift towards the edge**'.
  As a general rule of thumb, the deeper the network, the wider its opportunities to learn robust features in a hierarchical way, the **more accurate** its performance and the larger its size, unless some architectural and methodological strategies are undertaken (chapter 7);

The first step when performing image detection is **features extraction from the input feed**, i.e. the process allowing to derive information that would be manipulated to fulfill the specific task under analysis. These task can be divided into two main classes:

- **Image classification** Given a training dataset with:

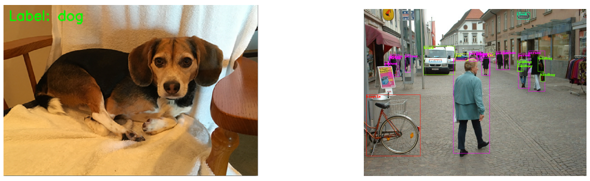
[5]a prior probabilty distribution states the level of knowledge (belief) of an event before its actual measure. It is said to get stronger as long as its entropy, i.e. the uncertainty about prediction's belief, gets smaller.

[6]among AlexNet major contributions: **use of ReLU** activations, extension towards multiple-GPU to **shrink training computation time** and introduction of some regularization techniques to **reduce overfitting**, like data augmentation and DROPOUT layers.

1. a set of input images, each containing one single object;
2. a corresponding set of ground-truth labels, each containing the image category;

the aim of the network is to learn how to **classify** images with a certain probability or classification score, when properly taught on the training dataset (fig. 2.5(a)).
The advent of CNN, as introduced by [6], made it possible to extend image classification from *small and low-resolution* datasets - e.g. **MNIST** [7] [16] and **CIFAR-10** [8] [17] - to *large and high-resolution's* such as **ImageNet** dataset by Stanford University [9][18].

- **2D objects detection**: extension of pure image classification task that introduces **multiple objects** detection, starting from the seminal paper on Region-based CNNs (`R-CNN`, [19]). Within this context ground-truth annotations do **not only** contain **the correct class** for each object, yet also the **set of bounding box coordinates** that locate it in space (fig. 2.5(b)).



(a) Image classification example [5]

(b) Object detection example

Figure 2.5. Convolution operation visual representation

It is usually true that moving towards **modeling a real-world based phenomenon** introduces a series of constraints and challenges.
In the case of 2D object detection, these mostly concern the final layers of the network with a three-state problem:

1. Quantify the network process of discerning an object (foreground) from the background by assigning an `objectness` confidence to some proposed regions of the input image: the process is called **localization**;

2. Quantify four spatial coordinates around the selected objects, whose `objectness` is reasonably high: the process is called **bounding box regression**;

3. Compute classification scores of the localized objects (**class confidence regression**), the only existing process when dealing with simple classification.
   Steps 2. and 3. define the **actual detection phase** since they belong to the executive part of the flow [10];

[7]60000 hand-written digits images at 24×24 resolution

[8]60000 colour images at 32×32 resolution

[9]about 1.4 million images at 256×256, divided into 1000 categories. It is used in one of the most prestigious competitions on image classification, ILSVRC

[10]Henceforth, **'pure classification'** will refer to the process of assigning a single-object image a class category - without localizing - while **'detection'** will refer to an actual interest in the spatial position of multiple objects

Fig. 2.6 provides an high-level view [12] of the actual parts that build up a modern CNN for object detection. The analogy relates with human's spinal chord: **backbone**, **neck** and **head**. The first two provide feature map extraction, the last executes the detection (either simple classification or 2D detection). Each block makes use of the microscopic elements introduced in the previous section and is briefly detailed in the following.

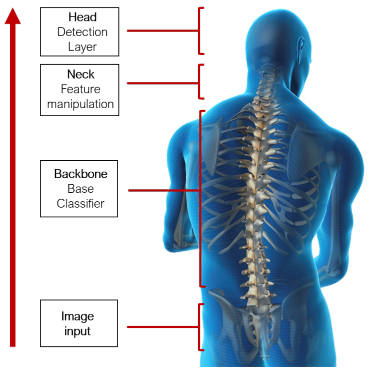


Figure 2.6. Modern CNN structure, human body analogy

Backbone
It is a CNN model typically **pre-trained** [11] on a large image dataset, e.g. `ImageNet`. The last layer, responsible for the predicted score in a simple classification problem, is removed and the network is used to **build the features map**.
Layer's **sequential stacking** is performed by employing the building blocks mentioned in the previous sections. In the recent years, some other features like inception modules (parallel stacking, [20]) and residual connections (feed-forwarded skip connections to reinforce feature learning, [21]) have been introduced to improve the teaching phase robustness.
Given the actual trend of reducing the model size by preserving its accuracy, there exists:

1. **Heavy backbones:** extremely deep CNN to be trained via single- or multi- GPU training. It is worth citing `VGG16`, `ResNet-50`, `ResNet-100`, `GoogleNet`, `Darknet-19` [22] [23] and `Darknet-53` [24] among the most employed [12];

2. **Light backbones:** these are NN models willingly crafted to keep some **efficiency margin** in terms of overall size.
   Among them, it is worth mentioning `SqueezeNet` [25] and the `MobileNet` family [15] [26] [27] which has rapidly become SOTA in the field.
   The architectural features that distinguish them in terms of efficiency/compactness trade-offs are detailed in chapter 7.

[11] a NN is said to be **pre-trained** when its weights have already been taught through gradient-based learning to perform a particular task, usually on a pure classifier's backbone. In this way, detector's weights are not randomly initialized, but training stage can start from an already reasonably well-working initial point, nearer to minimize the cost function.

[12] the final number usually indicates network's depth in terms of layers.

Neck
It comprises a set of techniques to handle the output feature maps. As an example one of the most used, **Feature Pyramid Network (FPN)** [28], allows to improve small objects detection by collecting features from the intermediate layers of the backbone.
In fig. 2.7, the last layer encodes the feature maps to be used for detection. They have low resolution due to the series of up/down sampling after `POOL` and `CONV` operations, but strong **semantic value** since high-level objects, such as effective shapes, are recognized in the deeper hidden units.
Overall, FPN starts a **top-down process** where intermediate feature layers are reconstructed **including residual connections** to keep the resolution as high as possible from the semantic-reach deeper layer.

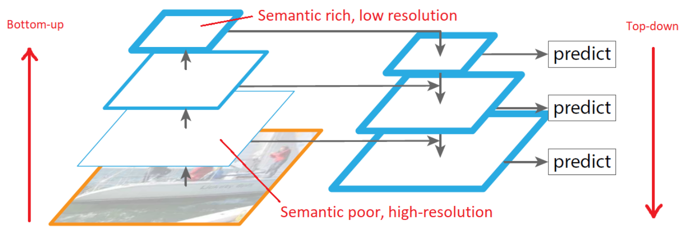


Figure 2.7. Feature Pyramid Network sketch [28]

Head
It is the **detection layer** responsible to predict confidence scores of classes and - when requested by the task - object's bounding boxes.
According to historical development and evolution, the kind of head actually determines the type of object detector:

1. **Two-stage** detectors (or **sparse predictors**): region proposals for localization and model's detection predictions are **detached steps** that happen sequentially and require **fragmenting** the input image before inference.
   This provides high accuracy though not real-time detection. Pioneers in the field, are mostly based on the `Region based CNN (R-CNN)` family, that **made it feasible the transition from pure classification to detection**;

2. **Single-stage** detectors (or **dense predictors**): region proposals extraction and subsequent model's predictions are performed in a **single step, by looking at the input image as a whole entity, at a glance**. This slightly decreases their accuracy, yet making them capable of real-time detection [13].
   They find their pioneers in the `You Only Look Once (YOLO)` (very first) and `Single-Shot Detector (SSD)` families as well.

An interesting picture highlighting the difference between Two- and One- stage detectors is provided in [12] and shown in fig. 2.8.

[13]click here: `https://www.youtube.com/watch?v=Cgxsv1riJhI` for the live comparison of `YOLOv2` and `Faster R-CNN` fed with an input video stream by Joseph Redmon at a TEDx event in 2017.

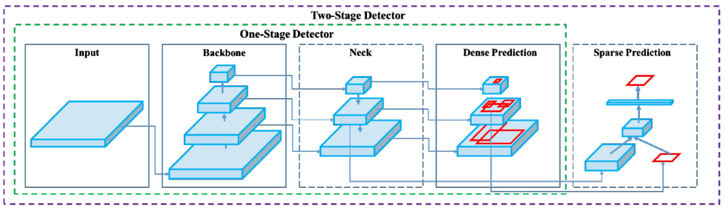


Figure 2.8. One- and Two- stage detectors comparison [12]

Last, with reference to fig. 2.6, it is worth mentioning that, alongside anchor-based models that are the main subjects of this document, several anchor-free detectors have been developed as well.

2.4 The `Head`: detection layer

This section will briefly describe the key differences between the two main heads typologies in terms of classification scores and bounding box predictions computation.
It will serve either to justify the choice of the single-stage family employed in chapter 4 and to highlight the historical evolution in the field, **which is based on borrowed techniques and methodologies that have quickly become cornerstones**.

2.4.1 Two-stage object detectors

Sliding window, Image Pyramids: `Overfeat` approach

The transition from pure classification to detection has been first proposed in 2013 with the `Overfeat` network (2013) [29], its main purpose being the capability to localize mostly single-objects based images by employing multi-scale classification.
The primitive idea is indeed to use:

1. **image pyramid** [14] to get 6 different representations of the original image at different **scales**;
2. a **sliding window approach** to select proper regions of interest (crops); objects with different sizes can be located by the same window thanks to re-scaling;
3. perform both classification and, **after**, bounding box regression for each crop and scale;
4. output final and refined bounding box coordinates by considering the overlap over different crops for all scales.

[14]image processing technique that performs multi-scale sub-sampling of the input feed, varying the original resolution.

Overfeat - Summary

Benefits: Details are left to [29], however, the 2 last `FC` layers are treated through convolution operations, making the network pseudo-fully convolutional. Henceforth, it starts probing the advantages of dealing with fully-convolutional networks: input size independence and computation speed up [30].
Disadvantages: the sliding window and image pyramids **slow down** the whole inference process and improve the overall cost at training time.

Region-based approaches

R-CNN family evolution line consists of three main contributions: `Region-based CNN` [19], `Fast R-CNN` [31] and `Faster R-CNN` [32], with the last two correcting some weaknesses of the original work, mostly at inference time, though the general workflow is kept the same.
A synthetic description is provided below, mostly underlining the major achievements and improvements at each steps.

`R-CNN` (2014)
It solves the most important drawback of the approach proposed in [29] by smartly selecting the region proposals over which performing classification and bounding box regression.
About its building-blocks structure:

- A selective search algorithm [33] is applied on the input image to obtain a set of N (here set to 2000) **region proposals**, or crops, with high *`objectness` confidence*, i.e. the probability of dealing with an object - or foreground - rather than background;
- **Each crop** is independently fed to the CNN. Classification scores, as well as bounding box coordinates to fine-tune the outputs localization coming from the selective search, are computed through two `FC` layers;

R-CNN - Summary

Benefit: It reduces training and inference time by searching and selecting a **reduced number** of region proposals instead of employing a **more costly** sliding windows approach;

Disadvantages:

- Inference still takes a large amount of time (about 49 seconds [31] for a single image), since each crops has to be individually fed to the CNN; being N = 2000 proposals, this means that the feature maps extraction process happens **2000 times more** than needed;
- Selective search does not participate directly in the learning process [30] and plays the role of a pre-processing module. It is said that the network is **not end-to-end trainable**.

`Fast R-CNN` (2015)
`R-CNN` drawback no. 1 is solved by **reducing** the actual number of **feature extractions** to one:

1. The input image is fed to the CNN and its feature maps calculated;
2. Selective search algorithm is applied to extract - i.e. to only find - region proposals for object localization;
3. A novel module called **Region of interest (ROI) pooling** is employed to execute ROI cropping on the feature maps according to the results provided by the selective search;
4. Cropped ROIs are fed to a `R-CNN` where classification labels and bounding box coordinates are computed;

`Fast R-CNN` - Summary

Benefit:

- large reduction of training and inference time (about 2.3 seconds [31] for a single image);
- End-to-end training is reached, being each input fed to the network only once;

Disadvantage: dependence on the selective search algorithm still exist at inference time;

`Faster R-CNN` (2016)
In order to integrate the region proposal extraction in the flow at inference time and replace the selective search, the proposal mechanism is split in two halves:

- **Anchor reference generation:** the input image is sampled with a steady stride to generate a set of N_{grid} grid reference points. A fixed number of k boxes with different aspect ratios ($k = 9$ in [32]) is associated to each grid point. Therefore, the maximum number of proposal reads $N_{grid} \cdot k$.
  The use of anchors allows to remove multi-scale images, instead considering anchors pyramids [32], i.e. multi-scaling operated at bounding box coordinates level;
- **Region Proposal Network**: an additional network whose task is to reduce the total number of proposals according to the entity of their *`objectness` score*, i.e. whether they are **foreground** or **background**, keeping those belonging to the first class according to the intersection over union `IoU` with the reference ground-truth boxes (training dataset labels annotations).
  As a matter of facts, its loss for the training stage is composed by two terms:
  1. Binary cross-entropy for addressing the binary class problem of foreground/background distinction;
  2. Bounding box regression for coordinates calculation. It is handled by computing the offset between anchors and predicted boxes coordinates, in the form (x_{center}, y_{center}, `w, h`), where each term is normalized with respect to the corresponding horizontal or vertical image dimension;

The set of N output proposals provided by the Region Proposal Network (RPN) module corresponds to the crops identified by the selective search algorthm in the previous implementations. The set undertakes ROI pooling as well to extract fixed size windows. Each of them is given to a R-CNN in order to calculate the **final classification score predictions** and to **fine-tune the already computed bounding box coordinates**, adding a loss with another two contributions.

Faster R-CNN - Summary

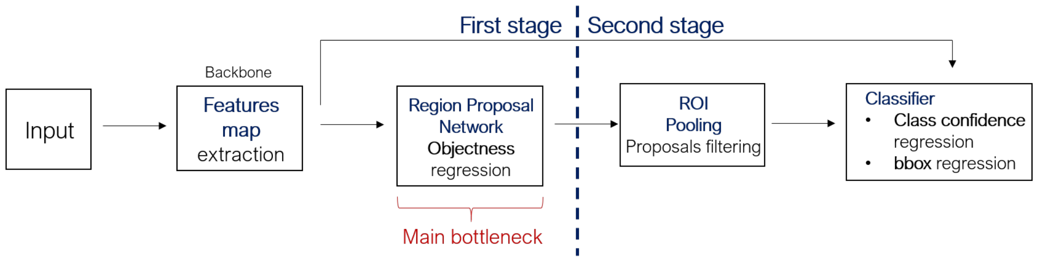
Benefit: **The whole computation is based on gradient learning**. Moreover, the four loss contributions (two for RPN, two for the final detection layer) can be jointly trained increasing training speed. Last, inference speed is increased to 0.142 s for single image w.r.t. Fast R-CNN.

Disadvantage: Overall, inference time is still slow (about 7 FPS) and prevents the network to be used for real-time object detection.
The reason is the network's **modular setup** [30] when dealing with **(1)** region proposals for objectness confidence computation (RPN), **(2)** proposals cropping (ROI pooling) and **(3)** detection (last layer).

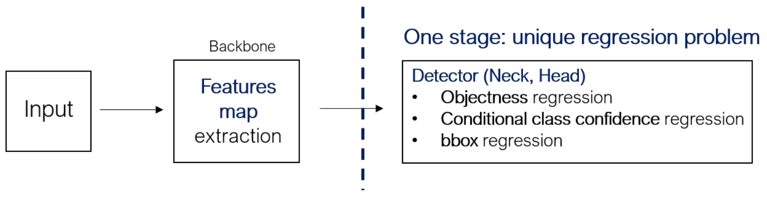
2.4.2 End-to-end object detectors: pursuing *real-time* detection

In order to prevent the bottleneck determined by the multi-step approach of the R-CNN family, fully end-to-end object detectors have been introduced starting from 2016.
Though their inner differences, the general underlying approach is the same: performing objectness confidence, classification confidence and bounding box regression, yet all at once **as a unique regression problem**.
The fundamental take-away is to avoid having a portion of the network to generate N proposals from the input feed - i.e. to first evaluate their objectness and *thereafter* passing them to the detection layer - but **looking at the whole input image once** (hence forth the name YOLO), in a **single-shot** (hence forth the name SSD)[15].
The key difference between the two approaches is sketched in fig. 2.9 and its main advantage is a **huge increase of speed at inference time**.

[15]This approach mimics the **human ability** of deciding the potentiality of x being an object and, in case of affirmative answer, evaluating its category as well as position in space **with a single and quick glance.**



(a) Two-stage object detector high-level view



(b) One-stage object detector high-level view

Figure 2.9. One- and Two- stage detectors system level comparison

In the framework of this document, more attention have been paid towards `YOLO` networks family, making it worth detailing them more deeply. `SSD` topology is instead briefly described in sec. 2.4.2.

`YOLO` family

Several `YOLO` versions have actually been developed up to the fourth generation, released in May 2020 [16].
Up to now, `YOLOv3` is considered SOTA in the field of real-time object detection for CV.

`YOLOv1` (2016)

In order to grasp the whole image context, the following procedure is followed in [22]:

- Given an input image, divide it into $\mathtt{N}_{grid} \times \mathtt{N}_{grid}$ grid cells. Each of these is responsible for detecting one potential object [17];

- For each grid cell, the network predicts B (where B = 2 in the original paper) bounding boxes - each of which brings 5 values: 4 bounding box normalized coordinates and 1 `objectness` confidence score - along with $\mathrm{N}_{classes}$ **conditional** class probabilities.
  The term 'conditional' indicates that the object's probability of belonging to a certain class **is conditioned** to its actual grade of `objectness`, i.e. whether it is an object or not;

[16]An apocryphal `YOLOv5`, released in June 2020 by the Spanish company *Ultralytics LLC*, is not yet considered official, since it is not associated to a scientific publication and it is mostly a re-shaping of `YOLOv3`.

[17]'potential object' means that its grade of *`objectness`* is **not yet known** at detection time as in region-based approaches.

- The output prediction is therefore a tensor of the form (N_{grid}, N_{grid}, B·5 + $N_{classes}$). $N_{classes}$ equals 20 for the PASCAL Visual Objects Classes (VOC) [34] dataset and 80 for Microsoft COCO [18] [35];
- The **final class confidence probability (or score)** for a certain object at inference time is built from objectness and conditional class predictions. It reads [22]:

$$\text{objectness}\cdot \mathrm{P}(\text{class}_i|\text{object}) = \mathrm{P}(\text{object})\cdot \mathrm{IOU}_{gt,box}\cdot \mathrm{P}(\text{class}_i|\text{object}) = \mathrm{P}(\text{class}_i)\cdot \mathrm{IOU}_{gt,box} \tag{2.3}$$

  If P(object) = 0, there is **no intersection** between the ground truth and the predicted box, thus the model confidence of the prediction being an object falls to zero (**background** case).
  Otherwise, if an intersection exists, P(object) = 1 and the **objectness confidence equals $\mathbf{IOU}_{gt,box}$**. In this way, P(object) plays the role of an indicator function.

How the loss function is computed to get one, unique regression procedure is detailed further down in the document.

YOLOv1 - Summary

Benefits:

- The network reaches 45 FPS at inference time [22] while keeping good performances in terms of accuracy on COCO dataset;
- Being able to deal with the whole context of the image, the network is able to distinguish foreground (objectness > 0) from background (objectness $= 0$) better than region-based methods;

Disadvantages:

- Incorrect localization [22]: The number of boxes predicted for each grid cell is low (B = 2), while there exists one conditional class probability output only for each grid cell. This prevents the network to recognize group of objects that are not clearly detached one from the other;
- The model is **not anchor-based** as Faster R-CNN, therefore it is weak in generalizing towards several aspect ratios and shapes;
- Bounding box regression does not take into account multi-scale approach, thus **struggling to generalize towards small objects** that would require high resolution to be detected;

YOLOv2 (2017)

[18] MS COCO plays the role of ImageNet in the domain of pure classification and is rapidly growing as SOTA dataset for object detection. It has about 120 thousands between train and validation annotated images, as well as around 40 thousands available for testing.
VOC is significantly smaller, with about 10 thousands samples fragmented into train, validation and test sets.

YOLOv2 aims at solving localization and missing detection's errors from the first YOLO. In order to do that, some techniques are applied, such as **(1)** BATCHNORM regularization layers that prevent over-fitting and allow to remove DROPOUT final layers, **(2) multi-scale training** that introduces multiple scaled inputs at training time and **(3)** pass-through connections from intermediate layers. The latter two increase the network robustness against small objects.
Nevertheless, the most important novelty concerns bounding box regression.

Fixed anchors (or priors)
Instead of proposing B bounding boxes directly as its predecessor, YOLOv2 borrows anchors-based bounding box regression from Faster R-CNN. Nevertheless, two main changes are introduced:

1. **Grid-cell offset**: Faster R-CNN computes the offset between the proposed bounding box and priors coordinates, following [32]:

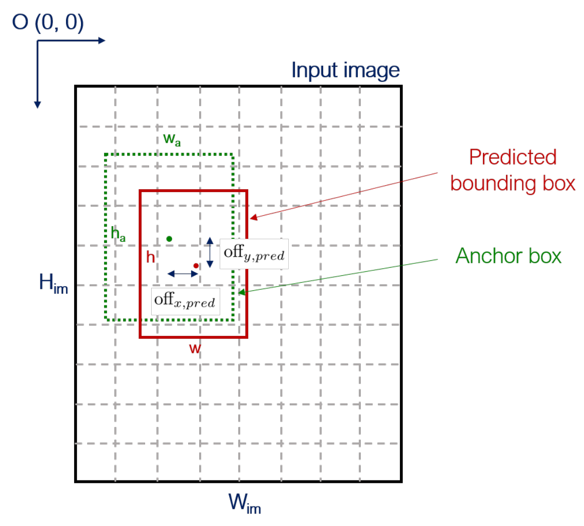
$$\begin{cases} \text{off}_{x,pred} = \dfrac{(x_c^{pred} - x_a)}{w_a}; \\ \text{off}_{y,pred} = \dfrac{(y_c^{pred} - y_a)}{h_a}; \\ \text{off}_{w,pred} = log\left(\dfrac{w}{w_a}\right); \\ \text{off}_{h,pred} = log\left(\dfrac{h}{h_a}\right) \end{cases} \tag{2.4}$$

   where pedices c and a indicate 'center' and 'anchor' respectively and w$_a$, h$_a$ are anchors de-normalization factors.

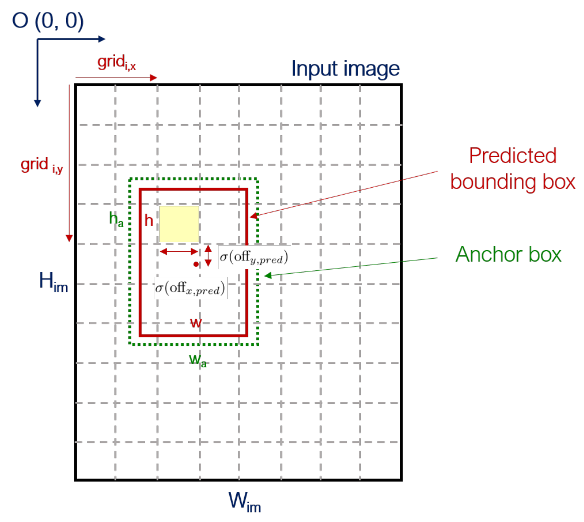
   YOLO authors introduce x- and y- offsets as well, though with respect to each grid-cell center, according to:

$$\begin{cases} \sigma(\text{off}_{x,pred}) = x_c^{pred} - \text{grid}_{i,x}; \\ \sigma(\text{off}_{y,pred}) = y_c^{pred} - \text{grid}_{i,y}; \\ \text{off}_{w,pred} = log\left(\dfrac{w}{w_a}\right); \\ \text{off}_{h,pred} = log\left(\dfrac{h}{h_a}\right); \end{cases} \tag{2.5}$$

   where grid$_{i,x}$ and grid$_{i,y}$ is ith grid center coordinate tuple and $\sigma(\dots)$ is the **sigmoid activation** to constraint the predicted value in the range [0,1].
   As it will be highlighted below, at training time the tuple ($\texttt{off}_{x,pred}$, $\texttt{off}_{y,pred}$, $\texttt{off}_{w,pred}$, $\texttt{off}_{h,pred}$) is compared with ($\texttt{off}_{x,gt}$, $\texttt{off}_{y,gt}$, $\texttt{off}_{w,gt}$, $\texttt{off}_{h,gt}$), where 'gt' means ground truth, to compute the loss for box regression.
   Fig. 2.10 reports a visual scheme of the different ways bounding box regression is performed in two- and one- stage detectors.



(a) One-stage detector, coordinates prediction computation



(b) Two-stage detector, coordinates prediction computation

Figure 2.10. One- and Two- stage detectors bounding box regression notation

2. **Anchor clustering:** fixed priors choice should improve the learning phase as much as it mimics the distribution of object's bounding boxes provided a specific image training dataset.
   `Faster R-CNN` hand-pick anchors [32] [23], while `YOLOv2` performs ***k*-means clustering** [19] over the ground truth bounding box distribution at varying number of priors, i.e. the clusters. The task is to get the smallest number of groups (i.e. clusters/rectangles) the

[19] *k*-means clustering is a ML technique in the domain of unsupervised learning that allows to create clusters

ground truth boxes should be separated into in order to maximize their overlapping area. The set of anchors N_{anch} is associated to each grid cell as in `Faster R-CNN`.

Anchor-level predictions
Anchors introduction allows to **extend the number of actual predicted boxes**. For each prior, hereafter also referred to as **proposals**, 4 bounding box coordinates, 1 `objectness` confidence score and $\mathrm{N}_{classes}$ conditional class probabilities are computed. Therefore, given $\mathrm{N}_{grid} \times \mathrm{N}_{grid}$ grid cells, the output reads $(\mathrm{N}_{grid},\ \mathrm{N}_{grid},\ \mathrm{N}_{anch}\cdot(5 + \mathrm{N}_{classes}))$

`YOLOv2` - Summary

Benefits:
`YOLOv2` manages to handle its predecessor main disadvantages while keeping (and still improving) inference speed.

Disadvantages: It still lacks a multi-scale contribution mechanism to build predictions from different scales, thus **limiting** its overall accuracy.

`YOLOv3` - `YOLOv4` (2018; 2020)

`YOLOv3` comes with several improvements to further increase YOLOv2 accuracy, rather than speed, due to the competition of other family models like `RetinaNet` and SSD. Most of the changes are performed at `backbone` level and detailed below.
Moreover, the newly released `YOLOv4` mainly focus on image pre- and post- processing steps and only slightly modifies `YOLOv3` original `backbone`, while the `head` is kept exactly the same.
Therefore, less attention is dedicated to `YOLOv4`, though its major contributions are briefly mentioned:

- The third version of `YOLO` is a Fully Convolutional Network (FCN), with the already mentioned benefits of input size independence and computation speed up;
- `YOLOv2` had 19 `backbone`'s layers (`Darknet19`[20]) and 11 `head`'s (detection) layers. `YOLOv3`, other than introducing residual connections as in `ResNet` [24], increases networks depth to 106 global layers - half `backbone`, half `head`.
  This further **improves network accuracy** but slows down inference stage with respect to its predecessor to about 30 FPS;
- `YOLOv3` introduces predictions at three scales to improve small object detection. Three intermediate spots along the network are taken and propagated through the output, after proper up-sampling.
  The output shape is the same of `YOLOv2`, $(\mathrm{N}_{grid},\ \mathrm{N}_{grid},\ \mathrm{N}_{anch}\cdot(5 + \mathrm{N}_{classes}))$, with N_{grids}

from scattered data by computing the **Euclidean-norm distance** with a set of **centroids**, which will define the final categories. In this framework, centroids are the anchors themselves and the distance reads [23] d(box, centroid) = 1 - `IoU`(box, centroid), where `IoU` indicates the intersection over union areas.
Their optimal number is typically selected through the **elbow method**, i.e. trading off the curve representing the monitored property when varying the cluster's number.

[20]`Darknet` is the ML framework used to develop `YOLO`, and stands along `TensorFlow`, `PyTorch` and `Caffe`

= 13, 26, 52 for each output. N_{anch} = 3 for YOLOv3 after prior k-means clustering. The total number of predicted boxes thus reads:

$$\text{Pred}_{boxes,\texttt{YOLOv3}} = (13^2 + 26^2 + 52^2) \cdot 3 = 10647$$

against

$$\text{Pred}_{boxes,\texttt{YOLOv2}} = 13^2 \cdot 5 = 845$$

In general, the latter feature improves the network accuracy while reducing its speed, although keeping it faster then the majority of other object detectors.

- YOLOv4 chooses CSPDarknet53 [21] as backbone. Its main novelties concerns the coherent introduction and theoretical reorganization of two sets of techniques [12]: ***bag of freebies*** and ***bag of specials***.
  Both of them aims at improving model performances: the first set affects training cost without degrading inference stage time and mostly deals with image pre-processing for data augmentation, the second focus on enhancing some architectural aspects like receptive field, feature integration (e.g. through FPN addressing network's neck level) and post-processing techniques with anchor-based networks.

SSD (2016)

SSD [37] addresses the problem of Object Detection (OD) in a similar way as YOLO. Its main source of comparison according to timeline is YOLOv1, having been released in the same year. The original publication does employ VGG as the backbone network. Nevertheless, given the effective speed and accuracy performances guaranteed by the detector's head, other backbones have been integrated in the recent years, seeking to increase its efficiency in terms of hardware resources (model size and compactness).
Among them, it is noteworthy to mention SSD with MobileNet family backbones, as well as a compact version of SSD head called SSDLite, proposed in [15] [26]. They are employed in chapter 4 for adversarial attacks experiments. As a matter of facts, hence forth SSD will be used to indicate the architecture's head.

A resource-optimized version of SSD has been proposed in [26] and named SSDLite due to the reduced model's size. It replaces standard CONV layers with more efficient **depth-wise** and **point-wise**, briefly reviewed in chapter 2. This approach reduces the number of MAC [22] operations by a factor of 1/7. SSDLite has been employed in the adversarial attack experimental part.

The main contributions introduced with SSD are summarized below [37] [30]:

[21] Cross Stage Partial Network, a novel feature recently proposed [36] to enhance backbone's compactness towards edge applications. It optimizes gradient reuse during backpropagation.

[22] Short for Multiply and Accumulate, the two main operations performed with convolution.

SSD - Summary

Benefits:

- The network is **fully convolutional** in order to detach its input size from the particular backbone used;
- It provides anchors - or priors - proposals in a way similar to `Faster R-CNN`;
- Conversely with respect to `YOLOv1` and `YOLOv2`, the detection layer (`head`) other than progressively reducing the layer's volume, lowering the resolution, does directly connect intermediate layers to the final output, therefore **providing multi-scale learning during the forward pass**;
- `SSD` is fast and accurate.

The following fig. 2.11 shows some comparison results between two- and one- stage detectors on the `COCO` dataset. In the figure, different circles refer to a **variation in the input size**, since accuracy typically increases when the input resolution gets large.
Performances are indicated through the **mean average precision** metrics [23] as a function of the frames captured per second. As it can be observed, two- stage detectors generally show higher accuracy at the cost of low FPS, i.e. real time capability. Conversely, one- stage detectors allow real-time data detection, yet at the cost of slightly lowering model's capability in terms of accuracy.

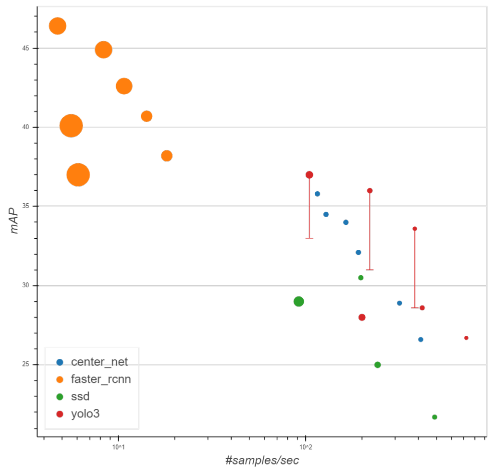


Figure 2.11. Single- and two- stage object detectors `mAP`/`FPS` comparison. Each circle represents the `mAP` of a specific NN model as a function of its real-time capability, measured as frames per second (FPS). Source: `https://gluon-cv.mxnet.io/model_zoo/detection.html`.

[23]A deeper insight about the latter as well as others common evaluation metrics is provided in chapter 4.

2.4.3 Training stage: Loss contributions for anchor-based detectors

In this section the main cost function contributions for an anchor-based object detector will be briefly described.
As mentioned in the previous sections, detection is treated has a unique regression problem, with three components:

- **Confidence score regression** to teach the model to correctly predict the `objectness` confidence of a prediction;
- **Bounding box regression**, responsible for box parameter's learning (typically x_c, y_c, w, h), i.e. the actual spatial detection;
- **Classification problem**, i.e. assign the detected box a correct class label;

which can be synthesized as:

$$\mathcal{L}_{tot} = \mathcal{L}_{conf} + \mathcal{L}_{BB} + \mathcal{L}_{class} \tag{2.6}$$

Bounding box regression

The typical approach has been to define the ℓ_n-norm [24] between ground-truth and predicted spatial parameters. Each parameter is assumed in the form of relative coordinates with respect to the set of reference anchors and/or grid cells, according to the network model (as an example, refer to eq. 2.4 or 2.5) [22] [38] [19]:

$$\mathcal{L}_{BB} \propto \sum_H \sum_W \sum_K \ell_n((\text{off}_x^{pred} - \text{off}_x^{gt}), (\text{off}_y^{pred} - \text{off}_y^{gt}), (\text{off}_w^{pred} - \text{off}_w^{gt}), (\text{off}_h^{pred} - \text{off}_h^{gt})) \tag{2.7}$$

where H, W and K indicates feature's height, width and depth (number of applied filters) respectively.
Nevertheless, some subtleties happen when considering ℓ_n norm which have been dealt with ever since `YOLOv1`, producing an evolution in the actual approach.
One of the most recent results (2019) in the field has been considering the Intersection Over Union (IoU) between ground truth and predicted boxes. Among the notable methods:

- **Generalized IoU** (`GIoU`): it evolves the IoU-based bounding box regression loss, of the form [33] [39]:

  $$\mathcal{L}_{BB,IoU} = 1 - \text{IoU(gt, pred)} = 1 - \frac{|B_{gt} \cap B_{pred}|}{|B_{gt} \cup B_{pred}|} \tag{2.8}$$

  where B is the tuple indicating bounding boxes parameters, by adding a penalization term that takes care of non-overlapping boxes, which witness vanishing `backprop` gradients slowing down regression:

  $$\mathcal{L}_{BB,GIoU} = 1 - \text{IoU(gt, pred)} + \frac{C - |B_{gt} \cup B_{pred}|}{|C|} \tag{2.9}$$

[24] ℓ_n norm is addressed with more detail in chapter 3. As for now, it is worth citing the Euclidean norm (also called norm-2 or squared distance, i.e. ℓ_2) as one of the most employed for bounding box regression.

C is the smallest box covering both B_{gt} and B_{pred}. Note that C reduces to $B_{gt} \cup B_{pred}$ when B_{gt} and B_{pred} are overlapping, otherwise $C > B_{gt} \cup B_{pred}$.

- **Complete IoU** (`CIoU`): `GIoU` often collapses to simple `IoU` because the penalization term is often near to zero.
  A novel approach recently proposed (December 2019) redesigns the penalty term in eq. 2.9 either by **minimizing the normalized distance (ℓ_2 norm)** d between ground truth and predicted boxes (DIoU) and considering their **aspect ratio consistency** [39]:

$$\mathcal{L}_{BB,DIoU} = 1 - \text{IoU(gt, pred)} + \frac{d^2(B_{gt}, B_{pred})}{C^2}; \qquad \mathcal{L}_{BB,CIoU} = \mathcal{L}_{BB,DIoU} + \alpha\nu \quad (2.10)$$

  where α is a tunable parameter and ν measures the aspect ratio consistency [39].
  This leads to a double advantage: regression speed up by distance minimization and a non-vanishing gradient in the case of box inclusion, where otherwise `GIoU` loss would have collapsed to `IoU`'s and vanished, being the intersection over union the same if $B_{gt} \subseteq B_{pred}$. Of course, `DIoU` and `CIoU` cover the non-overlapping case as well.

  `YOLOv3` employs `GIoU`, while `YOLOv4` witness the better results provided by `CIoU`.

Confidence score regression and Classification

The **first** contribution term considers the `IoU` between ground truth and predicted bounding boxes and calculates the distance with the corresponding network output. `IoU` encodes the reference information about the objectness confidence of a prediction. Besides, some tunable parameters are usually introduced to **enhance or smooth the penalization** provided by the loss when dealing with large/small objects or with foreground/background predictions, as e.g. in [38] [22].

The **second** comes from a pure classification problem (where indeed is the only one computed) and deals with class labels predictions. Two cases can be distinguished [40]:

1. **Single-label (SL) classification**: each prediction may belong to one class only, i.e. the set of classes is independent. In this case, the loss contribution reads:

$$\mathcal{L}_{class,SL} = -\sum_{i}^{C} c_{i,gt} \log(\texttt{Softmax}(c_{i,pred})); \qquad \texttt{Softmax}(c_{i,pred}) = \frac{e^{c_{i,pred}}}{\sum_j^C c_{j,pred}} \quad (2.11)$$

   where $c_{i,gt}$ and $c_{i,pred}$ indicates ground truth and network class scores predictions respectively, `Softmax`(x) is the softmax activation; the overall loss is called **Categorical Cross-Entropy loss**, or **`Softmax` loss**.
   `Softmax` guarantees that the predicted classification probabilities for the whole set of classes, given a certain proposal, lie within the range [0,1] and sum up to 1, thus compounding together class predictions.

2. **Multi-label (ML) classification**: when an object can belong to multiple classes, as in the case of `Google OpenImages` [25] dataset [41] where each macro-class - such as 'Person' -

[25]`OpenImages` is a wide, multi-task dataset that counts over 30 millions of annotated images and around 600 labels that can serve for several CV tasks, such as pure classification and image segmentation.

is fragmented into sub-categories - such as 'Man' and 'Woman' - score predictions should be detached, meaning that their sum can overcome 1. The problem is treated as a binary classification within each class - if the set of classes is represented with an array, each element can be either 1 or 0, and more elements can be 1 at the same time.
To treat each prediction independently, `Softmax` is replaced by `Sigmoid` activation:

$$\mathcal{L}_{class,ML} = -\sum_{i}^{C} c_{i,gt} \log(\texttt{Sigmoid}(c_{i,pred})); \qquad \texttt{Sigmoid}(c_{i,pred}) = \frac{1}{1+e^{-c_{i,pred}}} \tag{2.12}$$

`Sigmoid` constraints each prediction to lie within [0,1] without compounding the overall ensemble values. Loss in eq. 2.12 is named **Binary Cross-Entropy**.

Multi-label classification has been adopted by `YOLOv3`.

2.4.4 Inference stage: prediction's post-processing

Post-processing is a term that indicates the sequence of processes applied to the ensemble of bounding box proposals (and associated class confidence scores) predicted by the network. As mentioned in [12], post-processing only exists in the domain of anchor-based models where the concept of 'region proposal', being it sparse or dense, arises directly from the previously computed and fixed anchors.
It typically consists of two procedures:

1. **Confidence score filtering**: The ensemble of proposals is filtered by hard-coding a confidence threshold for their `objectness` score, therefore removing boxes that are less likely to be objects; after filtering, the **maximum conditional class probability** among all classes is retained for a certain proposal - in the single label framework - and class confidence score is computed as in eq. 2.3.

2. **Non-Maximum suppression (NMS)**: it aims at **removing redundant proposals**, i.e. bounding boxes survived after confidence score filtering and pointing at detecting the same objects. Usually, their `IoU` is filtered according to a NMS threshold value when comparing with the highest confidence score's proposal.
   Recent advances such as `GIoU`, `DIoU` and `CIoU` can be applied to this purpose as well.

2.5 Summary

This chapter outlined the powerful paradigm introduced by Convolutional Neural Networks in the framework of Computer Vision.
In particular, it described **microscopic** (`CONV`, `MAXPOOL`, `RELU`, `DROPOUT`) and **high-level** (`Backbone`, `Neck` and `Head` detection layers) building blocks, before setting them in the context of **pure classification** and **object detection** fields.
The latter domain is investigated in detail by distinguishing between **two- and single- stage detectors** with their most representatives models, i.e. `R-CNN`, `YOLO` and `SSD` families. This comparison has the aim of reproducing the steps that have lead to **unlock real-time detection** without accuracy loss by exploiting a unique regression problem.
Last, **bounding box**, **objectness** and **classification loss contributions** are reviewed for object detection training.

Part II

Adversarial Attacks

Chapter 3

Theory and Principles

Adversarial attacks can be defined as "*Inputs specifically designed to force a Machine Learning system to produce erroneous outputs*" [42]. The forcing nature typically consists of considering imperceptible alteration of the input - e.g. an image in the field of CV - that however affects the final system's prediction, **by properly training and crafting the alteration (distortion) itself**.

This chapter aims at briefly introducing why this topic has gained attention by the research community in the recent years as well as reporting a walkthrough SOTA literature works.

3.1 Motivation

An attacker is typically defined through a **threat model** [42]. It embeds the set of **perturbation's features** defined by:

1. **Goal**, or attack main purpose;
2. **Capabilities**, or adversarial approach to reach its task;
3. **Knowledge**, or adversary's accessibility towards the attacked system;

They are briefly described in the following.

Threat model goal, capabilities and knowledge

It is important to define the exact **goal** of the threat, i.e. which kind of damage it intends to bring to the NN model under attack. In the field of object detection, it deals with either **misclassification** or **misdetection** as already described and, for the first category, either with **targeted** or **un-targeted** attacks. They will be both introduced further down in this chapter during SOTA walkthrough.

Threat **capabilities** assessment concerns the way it can elaborate an assault.
Usually, it is worth reducing to a feasible and **non-trivial attack**, i.e. one that **does not distort completely the input semantic leading to a predictable decision change**. This implies the set of ℓ_n norm-based attacks, discussed further down in the next section, to be **constrained in intensity** and only changing the input set by a small amount according to the corresponding **distance metrics**.

Note that this holds true even when the adversarial pixels are **not constrained in intensity, but a bound exists anyway, e.g. in terms of spatial location**: this contributes to define a perturbed sample that slightly differs from the original, **being the majority of pixels unchanged**, and therefore preventing to assess a **predictable decision incorrectness** due to an input consistent change. It is the case of structured ℓ_0 norm - or patch - attacks, the main category analysed in this document.
Roughly speaking, an attack needs **inconspicuousness** [43] and should edit the input example so that a human could perform the particular task correctly as well, yet leading a machine to produce an error.

The last listed feature, adversarial **knowledge**, refers to the model's accessibility from the attacker. Typically, the regime addressed in this work is named **white-box attack** with **(1)** full weights, **(2)** pre-processing and **(3)** post-processing available to the adversary as media to strengthen its damaging capabilities.
White-box counterpart, called **black-box** attacks, are not addressed in the framework of this document. Nevertheless, appendix B outlines the attempt with adversarial patch **ensemble training** according to [2] in the fashion of increasing the attack transferability towards several models. This leads to less strong results than expected from literature comparison.
In the following paragraph, some examples of real-world based applications of adversarial attacks are presented. They have the task of stressing the potential damage that an attacker can cause to a NN model integrated in the physical world routine.

Real-world applications

It has initially been thought that adversarial attacks were not suited to model physical world scenarios. This applies to a restricted subset of adversaries, in particular those constrained in intensity but able to access each pixel of the input feed, a feature that makes the attack quite **unrealistic**.
Starting from [44] 'adversarial patch', as detailed below, has paved the way towards physical-realizable and realistic attacks which could cause harm to sensible applications like:

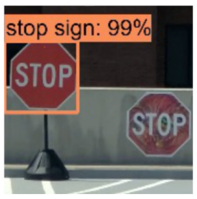
- **Autonomous driving systems** in the form of ADAS [1] systems: the most common fashion is **traffic sign distortion** to misclassify or misdetect them defining instability in terms of security.
  The same fashion is extendable towards pedestrian/car/cyclists hiding from the NN model responsible to detect objects, with the well-intuitable consequences. The latter exceeds into the field of intra-class variety (*Person* class), a more challenging task than traffic sign fooling;

- **Surveillance cameras** for people and face detection: the possible harm is people localization hiding from the camera in terms of object detection, or either face **dodging** and **impersonation** with respect to misclassification ([43]).
  Note that targeting the entire person body - which is usually treated as a class among other objects - and a single face leads to a difference in the overall technical approach, even if the qualitative effects are the same (see sec. 4);

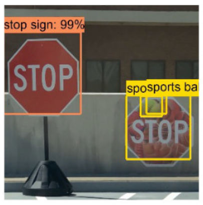
Fig. 3.1 shows an example of stop sign attack by an adversary. As it can be seen, the non-perturbed stop sign is correctly detected and classified, while its attacked version, even though

[1]Autonomous Driving Systems

still resembling a traffic sign for human beings, is misdetected and misclassified by a NN model, as in figs. 3.1(a) and 3.1(b) respectively.



(a) Misdetection



(b) Misclassification

Figure 3.1. Stop sign misdetection and misclassification. The stop signal squared in orange is the non-perturbed one and it is correctly detected and classified. The other one is, instead, subjected to the adversary attack [45].

The subsequent second part of the chapter has the aim of describing how the research field of Adversarial Attacks has evolved through time either in terms of methodologies and capabilities.

3.2 SOTA walk-through: Adversarial Attacks typologies

The fundamental keyword to understand the idea behind adversarial attacks is ***distortion***.
It quantifies the difference between two inputs I and I', where I is pristine and I' perturbed by an attacker respectively, in terms of **similarity**. For ad adversary to be defined as such, it should be able to **change the decision** of a NN model (regardless of the application domain) forcing it to **produce wrong results**. This is the reason why these attacks are usually referred to as **inducing imperceptible modifications** to the input.

Distance metric and pure classification problem

The first application domain to be addressed in the field of adversarial attacks for CV has been **pure classification** [46]. At system level, there are two ways imperceptibility can take form:

1. By addressing the whole set of image pixels, i.e. constraining the perturbation intensity to be bound by a certain amount in value (**spatially unconstrained**, **intensity constrained**). Typically, there is the interest in finding the **minimum disturbance** that produces a decision change in the task to fulfill by the model;

2. By addressing a subset of the image pixels, without constraining the perturbation intensity (**spatially constrained**, **intensity unconstrained**).

Problem number 1 has been the first addressed historically after [46] ([47], [48], [49]).
It typically consists of selecting a way to model **similarity** as perceived by human eye in the form of ℓ_n norm between the **original** and **adversary inputs**, x and x' respectively:

$$\|x - x'\|_n = \left(\sum_{i=1}^{m} |x_i - x_i^{'}|^n \right)^{\frac{1}{n}} \tag{3.1}$$

Among them, three are traditionally employed in the field [48]:

1. ℓ_0 **norm**, which indicates the number of non-zero elements. In the present task, it is the number of pixels that differs from the original image, i.e. the i for which $x_i \neq x_i^{'}$ and corresponds to the perturbation itself;

2. ℓ_2 **norm**, the first employed in [46], is the usual **Euclidean distance** or squared sum of squares; it can be kept small when all the pixels undertake small changes;

3. ℓ_∞ **norm** is defined as:

$$\|x - x'\|_\infty = \max(|x_1 - x_1^{'}|, |x_2 - x_2^{'}|, \ldots |x_m - x_m^{'}|)$$

   This means that the whole set of pixels can be **perturbed up to a certain value** which defines the maximum distortion.

In general, given the perturbation δ and distance metric $\mathcal{D}$(x, δ), the optimization problem for **targeted and un-targeted misclassification** reads:

$$\text{minimize } \mathcal{D}(x, \delta) \quad \text{so that} \begin{cases} C(x+\delta) = t, \quad x + \delta \in [0,1]^n & \text{targeted misclassification} \\ C(x+\delta) \neq C(x), \quad x + \delta \in [0,1]^n & \text{un-targeted misclassification} \end{cases} \tag{3.2}$$

where C(...) represents the set of **classifier predictions** and t the misclassification target. As it can be seen, the key difference is that when dealing with targeted misclassification the adversary addresses a specific class during the deceiving process. As an example, both of them have been employed for crafting attacks against face recognition [2] in [43].

In the framework of pure classification, the most effective methods used to build up adversarial examples for both targeted and un-targeted cases are:

1. **Fast Gradient Sign Method (FGSM)**: introduced in [50] by *Goodfellow et al.*, it employs ℓ_∞ norm on the perturbation δ so that the update rule for input x reads:

$$x' = \text{clip}_{[0,1]}(x - \delta \cdot \text{sign}\nabla\mathcal{L}_x) \quad \text{so that} \quad \|\delta\|_\infty \text{ is minimized} \tag{3.3}$$

   where $\mathcal{L}$ is the loss function that depends on the class prediction score and can take the form of `Softmax` loss;

2. **Projective Gradient Descend (PGD)** [49]: also called Basic Iterative Method or Iterative Gradient Sign Method [51], it is an **iterative version of FGSM algorithm** that either manages to achieve high convergence speed and guarantees a minimum-crafted distortion (imperceptibility) as well. Moreover, `PGD` happens to be a universal first-order attack for ℓ_∞ norm based attacks;

3. **Carlini-Wagner `CW` [47] method:** it solves the optimization problem in eq. 3.2 by choosing C(x) so that:

$$f(x) = \max_{i \neq t}(C(x)_i) - C(x)_t \tag{3.4}$$

[2]Face recognition is a synonym for classification, where spatial detection is not considered.

where clipping [3] is accounted internally as detailed in [47]. An attempt with `CW` method has been done with `YOLOv3` by adapting towards the framework of object detection and further reported in sec. 4.2.1.

Note that clipping either the input image x and the perturbation δ in the range of representable pixels is crucial in the process. It takes the name of **'box constraint'** [48].

In general, it is **unlikely** that an attacker can provide accessibility to the whole pixel set of an input image, which makes ℓ_1, ℓ_2 and ℓ_∞ based attacks quite non-realistic.
Conversely, even though ℓ_0 distortions still fail to model a **real-world based attack** in the **sparse** form ($\ell_{0,sparse}$) [52], they succeed in the **structured** form ($\ell_{0,struct}$), where spatial constraints are introduced to keep **adversarial pixels adjacent** to each other, defining a **patch-like structure**.
Spatially-constrained attacks correspond to the second approach towards imperceptibility previously outlined. Henceforth, the terms *structured* ℓ_0 and *patch* associated to attack/defense will be considered as synonyms and used interchangeably.

3.3 Structured ℓ_0 (patch) attacks

ℓ_n norm attacks, n $\in \{0_{sparse}, 0_{struct}, 1, 2, \infty\}$, either in the form of **targeted** or **un-targeted** adversaries, are said to produce **misclassification**. The domain of pure classification is the most suitable for this kind of attacks. Note that this does not exclude misclassification task in the context of object detection, which can be addressed as well even though not in the present work ([44], [53]).

In addition, $\ell_{0,struct}$ attacks well fit **object detection domain** due to the spatial constraint feature. This unlocks an additional type of model deceiving procedure that aims at **confusing a foreground object with background** and is named **misdetection** (starting with the work by *Thys et al.* [1]).

Fig. 3.2 shows an high level picture of the proposed adversarial attack hierarchy to summarize the concepts.

[3] Clipping operation allows to bound the range of pixel intensities within the allowed interval, namely 0-255 for `RGB` input data.

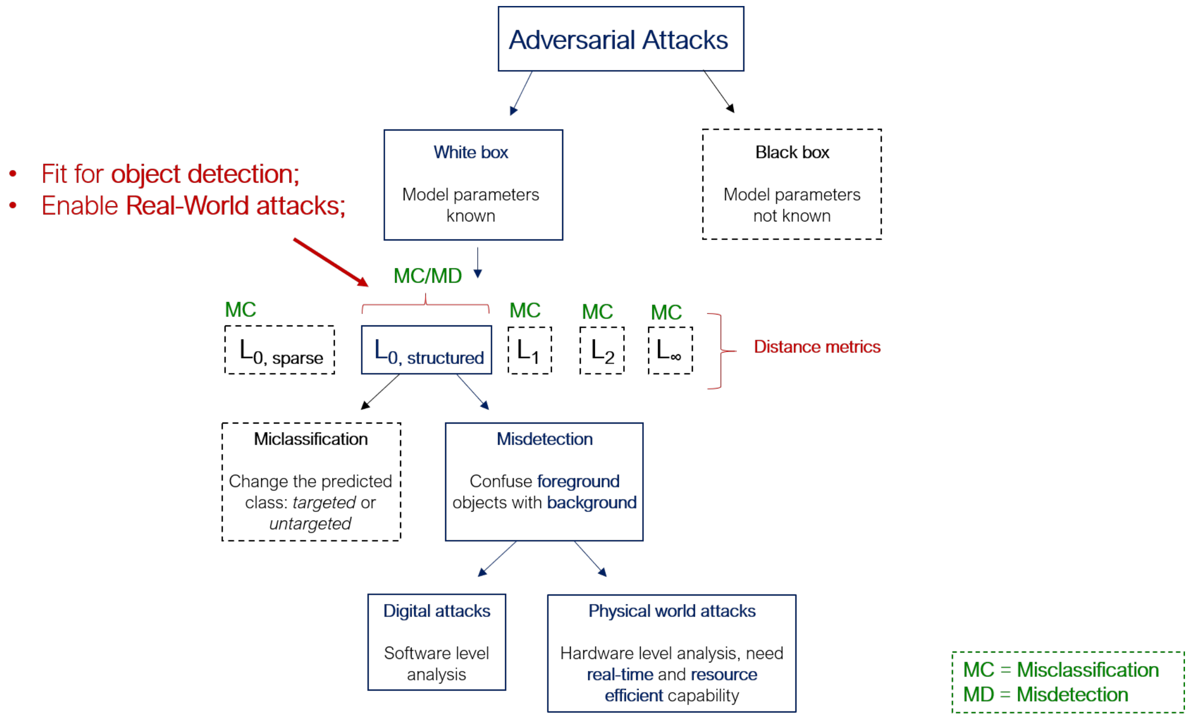

Adversarial Attacks
White box
Model parameters known
Black box
Model parameters not known
• Fit for object detection;
• Enable Real-World attacks;
MC/MD
MC
MC
MC
MC
L0, sparse
L0, structured
L1
L2
L∞
Distance metrics
Miclassification
Change the predicted class: targeted or untargeted
Misdetection
Confuse foreground objects with background
Digital attacks
Software level analysis
Physical world attacks
Hardware level analysis, need real-time and resource efficient capability
MC = Misclassification
MD = Misdetection

Figure 3.2. Adversarial Attacks high-level typologies

In the following I will summarize some results obtained in previous works concerning both misclassification and misdetection against structured ℓ_0 norm-based attacks.

Related works targeting misclassification against pure classifiers and detectors, $\ell_{0,struct}$ norm

The first work targeting adversarial patches has been proposed by *Brown et al.* [44] in the context of misclassification for a pure classification problem, resulting in a 'universal' patch.
Athalye et al. [53] has been the first to introduce either the concept of Expectation over Transformation (EoT) for adversary robustness and to extend adversarial attacks **from the digital world to the real, physical world**, crafting a 3D perturbed object.

Last, *Chen et al.* [45] and *Eykholt et al.* [54] employ a similar technique to actually deceive stop signs targeting misclassification on object detectors.

Related works targeting misdetection against object detectors, $\ell_{0,struct}$ norm

The problem of addressing more challenging tasks such as adversarial attacks on classes that present **infra-category variety** (e.g. *Person*) has been addressed first in *Thys et al.* [1], and then in *Wu et al.* [2], *Xu et al.* [55] as well for the `YOLOv2`, `YOLOv3` and `R-CNN` families.
Note that there exists a conceptual difference in terms of attack complexity **between two-stage and single-stage object detectors** for misdetection [2]. `R-CNN` family, say its last and most complete member `Faster R-CNN` (chapter 2), employs RPN to produce an ensemble of proposal that are already filtered once and ready to feed a `R-CNN` classifier.
Conversely, single-shot detectors propose an ensemble of bounding box candidates that are not

filtered yet and can **overlap with redundancy over the same object**. It is an harder task to suppress all of them and generate misdetection.

More recently, *Lee et al.* [56] addressed a 'generic suppressor' patch against `YOLOv3` that does not need to overlap with a specific target class, yet inducing false negatives over all the objects in the scene, extending the similar work by *Liu et al.* [57] from digital to real-world scenarios.

Loss function for misdetection

Inducing a model to generate a **negative wrong decision** (false negatives for misdetection) is rather an harder task than forcing it to produce a **positive wrong decision** (false positives for misclassification).
The subsequent loss functions employed during adversary training do target the optimization problem in eq. 3.2. In this context, C(x) is still the function that maps the input image to a number encoding the prediction likelihood of belonging to a certain class and its spatial coordinates.

Conversely with respect to other distance metrics-based norms, $\ell_{0,struct}$ constraint is implied in the patch pre-processing procedure (affine [4] transformation and application). Therefore, the cost function is assessed by **considering a score-based approach**: minimize network prediction capability by directly addressing its scoring output.
Two possibilities arise according to eq. 2.3:

- **Minimize prediction `objectness` score (`obj`)**, which is the parameter that allows a single-stage object detector to distinguish between foreground and background by a network at training time through `IoU` computation;
- **Minimize the final class confidence score (`obj-cls`)** of the class under attack (adversary's target);

Two loss functions to be minimized at patch training time have been employed in the present document, following [1] and [2] respectively:

$$\mathcal{L} = \max_i(\mathrm{f}(I,P)_i) \tag{3.5}$$

and

$$\mathcal{L} = \sum_{i=1}^{n} \max_i(\mathrm{f}(I,P)_i - t_{filt},\, 0)^2, \qquad t_{filt} \in [0{,}1] \tag{3.6}$$

where *f(...)* indicates the usual mapping to generate probability scores and its ith is the **bounding box proposal** associated to a certain anchor.
Eq. 3.5 aims at minimizing the loss by targeting the **maximum value of confidence score predicted by the model** $(f(I,P)_i$ over the whole set of proposals. Eq. 3.6 follows a similar approach by setting up **a threshold t$_{filt}$** and filtering accordingly the candidates set, since low-valued predictions do not yield significant contribution when penalizing the loss.
Filtering has been applied after network prediction normalization through `Softmax`/`Sigmoid` with threshold $\mathrm{t}_{filt} = 0.35$.
As previously mentioned, f(...) can assume either `obj` or `obj-cls` forms.

[4]Affine transformation: a geometrical ensemble of operations that allows to modify an object in space by preserving line parallelism. This implies that the object is not subjected to deformations, which is the case of another family, called perspective transformations.

3.4 Summary

This chapter theoretically introduced the field of **adversarial attacks** against pure classification and object detection. It described adversary typologies based on the **distance metrics** framework: find a perturbation that is able to change the model's decision while bounded by specific **norm constraint**: ℓ_0, ℓ_1, ℓ_2 and ℓ_∞.
In particular, ℓ_0 perturbations in the **structured form** have been referred to describe **real-world** based adversarial **patch** attacks, which is the main typology addressed by this document.

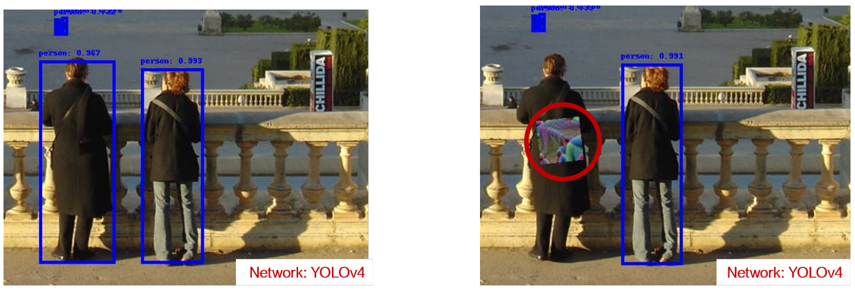
Chapter 4

Crafting Adversarial Attacks

Chapter 3 gives a theoretical insight about the framework of Adversarial ML from the attacker perspective. In particular, it outlines its main typologies and recent developments.
With reference to fig. 3.2, this document focuses on attacks that are:

- **White-box**;
- **Digital**;
- structured ℓ_0, or **patch attack**;
- targeting **misdetection** on object and face detectors;

As a general reference intuition, fig. 4.1 reports and an example of patch attack success against a SOTA and benchmark object detector, `YOLOv4`. Note that the success is encoded in the target misdetection, as clearly visible.



(a) Non-attacked input, after detection

(b) Attacked input, after detection

Figure 4.1. Adversarial patch attack example against `YOLOv4` object detector. On the right picture the **person** on the left is attacked. The blue rectangles represent the object detection made by `YOLOv4`, while the red circle represents the patch attack that induces `YOLOv4` to make mistakes.

This chapter aims at reporting the **simulations** that have been performed with this kind of attacks typology, from the general framework setup for training and assessment (sec. 4.1) to the actual experimental results (sec. 4.2.1, 4.2.2 and 4.3).

4.1 Adversarial flow: setup

The underlying key concept in the crafting process of an adversary sample, independently of its typology, is **gradient-based learning** as for a traditional NN.
The entity to be updated applying **backpropagation is provided by the ensemble of input pixels intensities that make up the adversary**, being it **(a)** distributed to the whole input feed pixel set, **(b)** sparse or **(c)** spatially constrained as discussed with more detail in chapter 3.

The training stage is addressed first - sec. 4.1.1 - while the attack evaluation metrics that have been adopted are discussed in sec. 4.1.2.

4.1.1 Training stage

An adversarial attack with the aforementioned properties can be crafted by following the **setup flow recipe** [1] [58] reported below and schematized in fig. 4.3.

In order to give a visual idea of the adversary training process, fig. 4.2 reports some frames of the patch evolution obtained by targeting `YOLOv4` model.

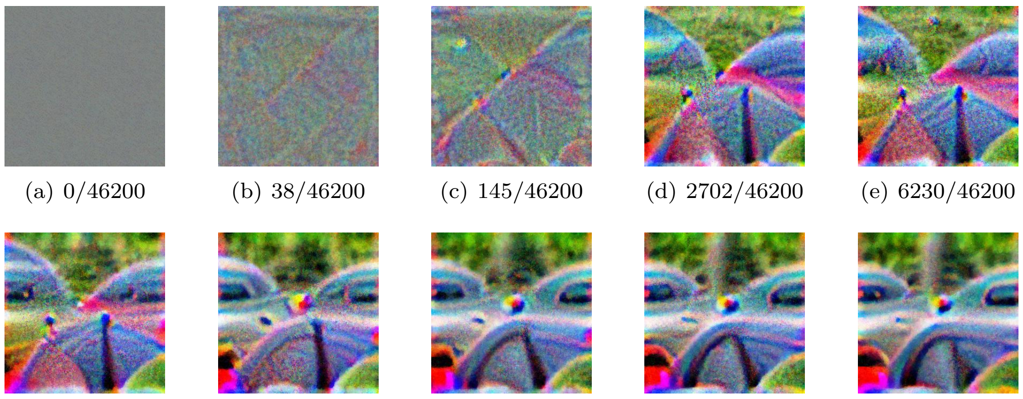

(a) 0/46200 (b) 38/46200 (c) 145/46200 (d) 2702/46200 (e) 6230/46200
(f) 11534/46200 (g) 19958/46200 (h) 28788/46200 (i) 36212/46200 (j) 46199/46200

Figure 4.2. Patch training evolution at different optimization steps

Adversarial patch - training flow

1. **Train images selection** The set I_{train} of input feeds I to train the adversary. According to the target, it could focus on a specific object class;

2. **Un-learned initial patch selection** It constitutes the ensemble of **starting pixel intensities** before gathering adversarial knowledge. As already mentioned, ℓ_0 assaults are not constrained in intensity [52], therefore the initial values choice becomes a degree of freedom.
   In this document, all **patches P are `RGB` tensors initialized with uniform gray intensity** [a] and have rectangular/square size with dimension $(3 \times H_P \times W_P)$;

3. **NN model *f* selection** Addressing a white-box attack **unconditionally binds the adversarial sample performances** to the network's weights it has been trained with.
   As specified in sec. 4.1.2, model's weights are frozen during training, i.e. the latter is set in **inference/testing mode**.

4. **Adversarial perturbation preprocessing**
   This step comprises the ensemble of procedures that should be undertaken to fulfill two tasks:

   (a) **Preprocessing for patch location:** patch size may equal or be in the vicinity of input's. Therefore, **proper scaling and translation** are required. If the attack does not target a specific object class (e.g. [59], [56]), scaling factor and translation can be *a priori* defined to be steady over the set of testing inputs, i.e. the patch is **always scaled and placed in a unique position**.
   Conversely, it should **adapt to the aspect ratio and location** of the object to be applied on. This document addresses the latter case following [1].

   (b) **Preprocessing for patch robustness:** set of strategies developed to enhance patch effectiveness towards physical-world based attacks by including brightness, noise and contrast changes at training and inference time. It takes the name of **Expectation over transformations** (`EoT`) [53].

   (c) **Patch Application:** the transformed adversary P is effectively applied to the input image I by replacing pixels in the proper position. **This action encodes the most natural meaning of structured $\mathbf{L_0}$ attack.**
   Given an application function $\mathcal{A}$ and a patch pre-processing function $\mathcal{T}$, the overall output tensor reads:

$$\mathrm{I}_P = \mathcal{A}(I, \mathcal{T}(P))$$

5. **Cost function**
   Typically, it comprises three contributions [43] [1] [2]:

   (a) **Adversarial contribution:** The cost function can take one of the forms outlined in chapter 3. This contribution deals directly with the **decision boundary switch** of the network between model's correct and polluted predictions;

   (b) **Smoothness-wise contribution:** when dealing with structural ℓ_0 attacks that are unconstrained in the pixel space, two further contributions keep control of their intensity [43]: Total Variation (TV) loss and Non Printability Score (NPS). TV loss increases patch smoothness to prevent sharp color variations. NPS matches the constraints imposed by **color gamut** [b] alteration for real-world scenarios;

6. **Forward step** I_P is fed to the network which computes its set of predictions $\mathcal{S}$ in the forward pass:

$$\mathcal{S} = f(I_P)$$

with $\mathcal{S}$ encoding regressed bounding boxes, `objectness` confidence score and conditional class probabilities **for each fixed prior** when f is a single-stage object detector as detailed in sec. 2.4.2;

7. **Backpropagation step** Loss gradient computation;

8. **Optimization step** Adversarial patch pixel intensities are updated according to the chosen **descend method**.
   Their update is coherent with the overall target: provided a patch $P \in [0, 1]^{(3 \times H_P \times W_P)}$ find the value of P that **minimizes the cost function**, i.e. either

   - the ℓ_n-norm distance between I_P and I, typically for n $\in \{1, 2, \infty\}$ norm-based attacks;
   - the class confidence scores predicted by the network during the forward pass, the one adopted in this context for structured ℓ_0 attacks;

   s.t. the network judging decision changes when the adversary pollutes the input, providing a wrong result (**misclassification**) or missing it (**misdetection**).

   Here, `Adam` [c] is selected as optimizer for all the experiments.

   Key observation #1: it is **crucial** that a **gradient path does exist** all the way back from the network's output to the set of input perturbed pixels P.
   In practice, this means that the model should deal with input tensors and not arrays, being the **gradient information suppressed** in the latter case. In software terms, ML frameworks such as `TensorFlow` or `PyTorch` must be employed, disregarding the more common NumPy library, at all levels of the process [d]

   Key observation #2: In order to keep the set of patch pixels in the range of representable colors [0,1] (or [0,255] without normalization), **the ensemble of pixels should be clamped** in that range after any update:

$$P_{i+1} = \mathrm{clamp}(P_i, 0, 1)$$

9. Repeat steps 1-8 until the convergence of the loss is reached, typically for $\mathrm{N}_{epochs} = 1000$. The chosen dataset is `INRIA Pedestrain` [e], that targets and annotates the class *Person* with roughly 600 and 300 training and testing images respectively, following the example of [1].

[a] i.e. tuple `(127, 127, 127)` if $P \in [0, 255]^{(3 \times H_P \times W_P)}$ or equivalently its normalized form `(0.5, 0.5, 0.5)` if $P \in [0, 1]^{(3 \times H_P \times W_P)}$

[b] Color **gamut** is defined as the complete subset of colors described by a **color model** (e.g. `RGB`)

[c] see appendix A.

[d] This also means that some useful tools towards ML frameworks unification - like **the intermediate format ONNX** - are not a deal at the moment in the adversarial attack context, being crafted exclusively to **exploit inference in a forward fashion** by employing arrays.

[e] `http://pascal.inrialpes.fr/data/human/`

Note that a structured ℓ_0 attack requires more effort in terms of preprocessing (step 4) from a practical point of view. In fact, even though the underlying intent is the same as other norm-based attacks, **spatial constraints** coming from structured ℓ_0 norm imply **detaching the ensemble of pixels** to be trained from the actual input feed and **make the pixel-by-pixel substitution** for application in a second step.

The latter happens to be the most valuable feature of structured ℓ_0 attacks, since it makes them **feasible towards physical-world attacks**: the **spatial restraints models attacker's inability to access every pixel of the input scenario**.

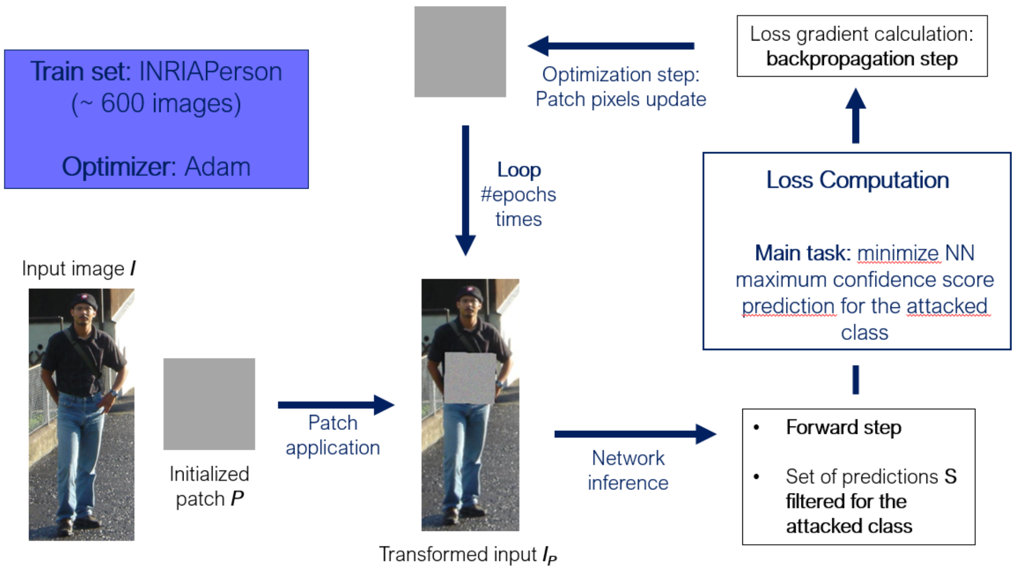

Train set: INRIAPerson (~ 600 images)
Optimizer: Adam
Optimization step: Patch pixels update
Loss gradient calculation: backpropagation step
Loop #epochs times
Loss Computation
Main task: minimize NN maximum confidence score prediction for the attacked class
Input image I
Initialized patch P
Patch application
Transformed input I_P
Network inference
Forward step
Set of predictions S filtered for the attacked class

Figure 4.3. Adversarial patch training flow

Adversarial perturbation preprocessing

Typically, a sequence of functions is applied to the adversarial patch P in order to either locate it on the target and train it for being robust against physical world-based phenomena. Both of these cases are addressed in the following.

Pre-processing for patch location

The present work aims at deceiving CNN predictions by targeting a specific class of objects, e.g. the `Person` class, at digital level. It is worth noting that some works, such as [56], aim at crafting a universal patch that is **position-agnostic** either for misdetection and misclassification cases. In the present case, patch application should be performed accordingly on the pure and clean original image.
The reference code for the set of geometric transformations in the framework of object detection has been taken from [1] and mostly re-used. Likewise, **challenges introduced** for patch pre-processing in the context of face detection have been dealt with by starting from the same code as underlying structure.

Spatial Transformer Network (STN) are employed by computing a `(2 × 3)` **affine transformation** matrix θ [1] [60]. The latter is the result of the composition (i.e. matrix multiplication) of several rigid transformations: **translation**, **scaling**, **rotation**.
The first two are described below and target patch geometrical application at the digital level, the third is detailed further down and aims at increasing patch robustness and pose-invariance.

1. **Translation** To translate an object by a tuple (δ_x, δ_y). In 2-D space:

$$\begin{pmatrix} x_{loc,P} \\ y_{loc,P} \\ w \end{pmatrix} = \begin{pmatrix} 1 & 0 & \delta_x \\ 0 & 1 & \delta_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ y_0 \\ w \end{pmatrix} = \mathbf{T} \cdot \begin{pmatrix} x_0 \\ y_0 \\ w \end{pmatrix}$$

   where w is a blind parameter introduced to allow matrix multiplication with homogeneous coordinates[2], (δ_x, δ_y) **are the box center coordinates** and $(\mathtt{x}_0, \mathtt{y}_0)$ the coordinates of the patch initial position, i.e. image top-left corner. For the *Person* class deceiving, the perturbation is applied at the center of the corresponding bounding box.

   Face detector case When dealing with Multi Task Convolutional Neural Network (MTCNN) for face detection, the adversarial perturbation has been chosen to take a **rectangular shape** when targeting **face's mouth** and the more challenging **shape of glasses** with respect to **human eyes**, as detailed in sec. 4.3. The spatial reference to perform translation is selected to be the middle point between mouth landmarks and eyes, where the facial landmarks are provided as network's outputs and shown in fig. 4.4. In this way, solid anchors exist to place the patch reliably.

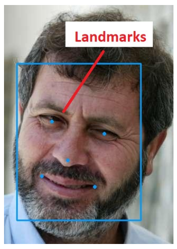


Figure 4.4. MTCNN **face** and **landmarks** detection (eyes, mouth extremities and nose)

2. **Scaling** Provided two scaling factors along box width and height, s_w and s_h respectively, the scaling matrix multiplies patch width ($\mathtt{W}_P$) and height ($\mathtt{H}_P$) by:

$$\begin{pmatrix} H_P' \\ W_P' \\ w \end{pmatrix} = \begin{pmatrix} s_w & 0 & 0 \\ 0 & s_h & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} H_P \\ W_P \\ w \end{pmatrix} = \mathbf{S} \cdot \begin{pmatrix} H_P \\ W_P \\ w \end{pmatrix}$$

   where s_w and s_h typically take the form:

[1]An affine transformation keeps parallelism of dimensions but not angles and distances.

[2]`https://cs.wellesley.edu/~cs307/readings/04-affine-math.pdf`

$$s = \frac{\text{target\_size}}{\text{original\_size}}$$

In the experiments provided for object detection (`YOLO` and `SSD`), the patch is a square with dimensions $\mathrm{H}_P = \mathrm{W}_P = 300$, therefore original_size = 300. The target size can be chosen as target_size = $\min(\mathrm{H}_{img}, \mathrm{W}_{img})$ or, following the approach exploited in [1] by empirically smoothing the target box's diagonal by a factor α:

$$\text{target\_size} = \alpha \cdot \sqrt{(W_{img}^2 + H_{img}^2)}$$

where α is an empirical factor and equals 0.2. The last method has been borrowed and its effectiveness properly checked.

When dealing with MTCNN for **face detection**, **no major changes** involve patch scaling for both 'mouth' and 'eyes' targets.

Pre-processing for patch robustness

Athalye et al. [53] first proposed the need to **improve attack's robustness** towards real-world environmental changes and variations such as angle pose, light and rotations. The approach has henceforth been adopted in nearly all the subsequent works on adversarial attacks [1][2][55][56][59][45] and is called **Expectation over Transformations** (`EoT`).
First, an ensemble of transformations applied on the adversarial patch including randomly tuned parameters is introduced **at each iteration of the patch training process**. They are detailed in the following:

1. **Brightness, contrast and noise** These are modeled as in the code provided by [1] with uniform distributions that are applied to patch P and affect its pixels intensities. Indicating the transformations as tensors **B**, **C** and **N** respectively:

$$\mathbf{P'} = \mathbf{P} \cdot \mathbf{C} + \mathbf{B} + \mathbf{N}$$

2. **Rotation** Given a rotation angle ϕ, the matrix is expressed as:

$$\begin{pmatrix} x'_P \\ y'_P \\ w \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ w \end{pmatrix} = \mathbf{R} \cdot \begin{pmatrix} x \\ y \\ w \end{pmatrix}$$

   Rotation is introduced with a varying angle $\phi \in$ [-20°, +20°] with respect to the axes-aligned position to increase model's stability and pose-invariance in view of an extension towards physical world-based attacks. This method has been proved to work fairly well still in [1] [2] [55] [53].

   The problem of face detection poses an additional constraint in terms of geometrical transformation due to the fixed rotation established by the direction joining eyes and mouth extremities as well. For the same reason, the random rotation **R** for `EoT` is removed when addressing this problem.
   Matrix $\mathbf{R}_{face}$ moves the patch towards an angle ψ. With reference to fig. 4.5:

$$\mathbf{R}_{face} = \begin{cases} \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} & \text{if } y_{\text{mark,l}} \geq y_{\text{mark,r}} \\ \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} & \text{otherwise} \end{cases}$$

and

$$\sin\psi = \frac{c_1}{i}, \qquad \cos\psi = \frac{c_2}{i}$$

from simple trigonometric relations. $y_{mark,l}$ and $y_{mark,r}$ indicate the left and right extremities of mouth/eyes respectively.

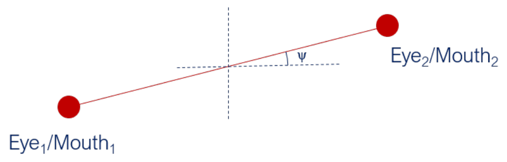


Figure 4.5. Patch rotation for mouth/eyes application, fixed-angle alignment

Provided a patch image input tensor P at training iteration ith, its entries intensities are altered and varied through tensors **B**, **C** and **N**, giving **P'** = **P** · **C** + **B** + **N**. The matrix θ that **properly locates and transforms *P*** according to the target bounding box I_{box} of image I is the composition - i.e. matrix multiplication - of **T**, **S** and **R**:

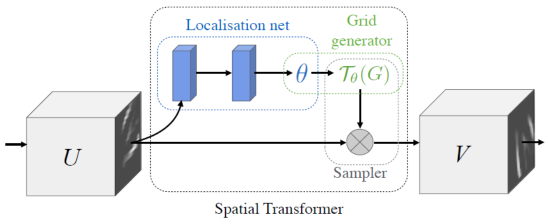
$$\theta_{affine} = \mathbf{T} \cdot \mathbf{S} \cdot \mathbf{R} \cdot \mathbf{P'} \tag{4.1}$$

θ_{affine} has dimensions `(2 × 3)` [3] and encodes the parameters of the function $\mathcal{T}_\theta$ responsible for the actual transformation, which is schematized in fig. 4.6(a).
Once it is computed, the whole process can be handled by two built-in functions introduced in either `PyTorch` or `TensorFlow` ML frameworks according to that in use. Each of them implements STN as described [60].
Given **P'** and θ_{affine}, a **regular grid** $G_{r,P'}$ is calculated from P'. Its transformed version, the **sampling grid** $G_{s,P'}$ is provided by:

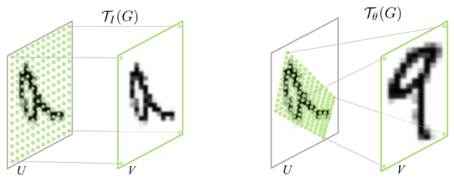
$$G_{s,P'} = \mathcal{T}_{\theta_{affine}}(G_{r,P'}) \tag{4.2}$$

i.e. it is obtained by warping the regular grid with the affine parameters embedded in θ_{affine}. Note that, as mentioned in [60], if the warping happens with $\mathcal{T}_I$, where **I** is the identity matrix and $G_S = G_R$ (the sampling grid equals the regular grid). Fig. 4.6(b) helps clarifying the concept.

[3] θ typically embeds also information about the shear, which is absent in the present work



(a) STN process flow



(b) STN sampling grid behavior

Figure 4.6. Spatial Transformer Network working features [60]

Finally, note that Spatial Transformer Networks are crafted to allow:

- Input tensors of any type, e.g. input images or convoluted feature maps;
- Transformations of any type: in addition to affine transformation, an extension involves **projective transformations** [4] like **Thin Plate Splines** `TPS` [61], which do **not necessarily preserve line parallelism**.
  In the context of adversarial attacks they have been applied by some authors [2][55] to model those patch deformations that would happen with a **physical application on non-smooth surfaces**. In particular, although `TPS` allows deformation modeling at the digital level, [2] shown that in this case patch effectiveness tends to be degraded with respect to the rigid perturbation case, perhaps because of the increased complexity at training time.
- The existence of a **gradient path through the sampling function** is guaranteed, provided the differentiability of θ_{affine}, which holds in the case of affine transformations. As mentioned in the adversarial attack training flow, that is **critical for updating the set of pixels of P** during the optimization step.

After patch transformation, its application on the input image is performed as described in the adversarial attack training flow.

The final set of learned pixels P_{last} at the end of the training loop is therefore a result of the expectation $\mathbb{E}$ over the whole set of transformations t $\sim\mathcal{T}$ **applied to the patch** $\forall$ ***i* steps**. This last feature is known as the already mentioned Expectation Over Transformations.

An example of the overall result for patch spatial localization is shown in fig. 4.7.

[4]e.g. `(3 × 3) matrices`

(a) Trained adversary patch

(b) Original unperturbed image

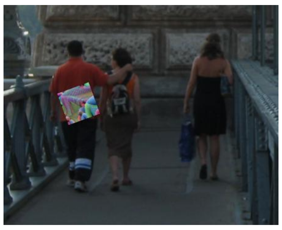

(c) Perturbed image after **affine transformation**

Figure 4.7. Patch transformation for digital application, `INRIA` dataset input image

Smoothness loss contribution

In order to preserve a smooth color variation within the adversarial perturbation pixel set, the distance in intensity between pixels in the same neighborhood should be kept low. This is provided by TV loss [62] [43], defined as:

$$\mathcal{L}_{TV}(P) = \sum_{i,j} \sqrt{(p_{i,j} - p_{i+1,j})^2 + (p_{i,j} - p_{i,j+1})^2} \tag{4.3}$$

where $\mathrm{p}_{i,j}$ is a pixel from the patch image ensemble at position `(i,j)`. As it can be observed, total variation loss increases when the distance gets larger, meaning sharp variation at pixel level.

Non-printability loss contribution

In general, a color is out of gamut, or altered, each time there is a change in the color space: digitizing or printing an image, changing a digital image color space.
In order to craft adversarial patches that could be printed safely, Non-printability score (NPS) loss is introduced as in [43]:

$$\mathcal{L}_{NPS}(P) = \prod_{p \in C_{spc}} |p - \hat{p}| \tag{4.4}$$

where $\hat{p}$ is a pixel from the set of printable colors C_{spc}.
In order to determine C_{spc} the authors from [43] printed a color palette and then digitized it to extract the set of all possible `RGB` triplets. Actually, this number has been reduced to the most significant 30 triplets (to get minimal variance from the full set), which however allowed them to keep optimal results. This reference color set is provided in their released code as well and used accordingly.
From eq. 4.4, `NPS` loss gets larger when the distance between $\hat{p}$ and p increases, which is an indicator of non-printability condition.

Overall, the **optimization problem** described in chapter 3 embedding the **adversary loss** (eqs. 3.5 or 3.6) is modified as follows after having introduced **(1)** the affine function $\mathcal{T}_{\theta_{aff}}$, **(2)** `EoT` over each iteration and **(3)** patch-dependent-only loss contributions $\mathcal{L}_{NPS}$ and $\mathcal{L}_{TV}$:

$$\text{Find } \underset{P}{argmax}(\mathcal{L}(P)) \text{ with } \mathop{\mathbb{E}}_{t \sim \mathcal{T}} [\, \mathcal{L}(f(\mathcal{A}(I,\, \mathcal{T}_{\theta_{aff}}(P))) + \mathcal{L}_{NPS}(P) + \mathcal{L}_{TV}(P),\, P) \,] \tag{4.5}$$

4.1.2 Inference stage and Evaluation metrics

In order to evaluate the network performances on the set of perturbed images two criteria are employed:

- Traditional NN evaluation metrics, i.e. **Precision-Recall** (`PR`) curves and **mean Average Precision** (`mAP`);
- Adversarial patch **Success Rate** (`SR`);

Precision-Recall

This statistics comes from the Information Retrieval research field. Given a binary classification problem distinguishing between two classes of elements labeled as **relevant** and **non-relevant**, the system under analysis is let providing its predictions, which are further divided into retrieved and non-retrieved elements. According to table 4.1, four scenarios arise [63]:

	Relevant	**Non-relevant**
Retrieved	True positives (`tp`)	False positives (`fp`)
Non-retrieved	False negatives (`fn`)	True negatives (`tn`)

Table 4.1. Information retrieval, four case scenario

They encode the system performances with respect to the expected outcome. It is clear from table 4.1 that `|tp|`[5] and `|fp|` indicate the number of system's correct and incorrect answers respectively, while `|fn|` and `|tn|` instead point out incorrect and correct abstentions.
Precision P and recall R are defined as:

$$P = \frac{tp}{tp + fp} = Pr(\text{relevant}|\text{retrieved}) \tag{4.6}$$

$$R = \frac{tp}{tp + fn} = Pr(\text{retrieved}|\text{relevant}) \tag{4.7}$$

Precision states the number of relevant samples over the set of retrieved, quantifying the system ability to propose correct results among those given.
Recall instead measures the model's ability to collect the set of relevant elements, independently on their correctness (model's **sensitivity**).
A useful and visual scheme is provided in fig. 4.8

[5] $|\dots|$ indicates set cardinality.

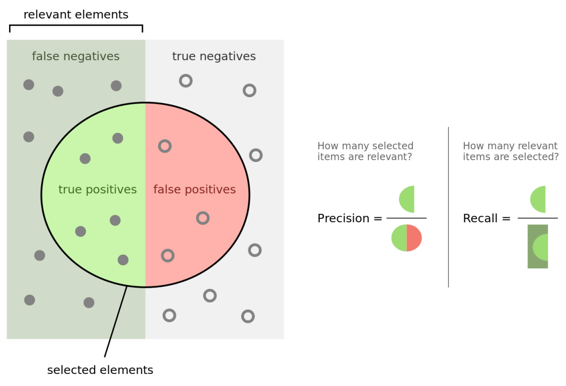


Figure 4.8. Precision-Recall high level scheme (source Wikipedia)

It is common to plot how precision varies **at varying recall**, i.e. progressively tuning the set of retrieved elements and measuring the corresponding system's precision. As the recall steps forward, precision may increase or decrease according to the retrieved element being relevant or not.
When R = 0, no-elements are retrieved (ideal maximum precision), while at R = 1 all the predictions are labeled as positive with a drop on P. Typically, recall is varied by setting up a threshold value T, and R=1 when T=0.

As already mentioned, the present work targets **misdetection, i.e. object hiding** by confusion with the background. This further contributes to **increase the number of false negatives** on the testing dataset, **reducing recall** in eq. 4.7. The latter trend can be viewed as a **misdetection footprint** and the selected parameter to monitor when evaluating the adversary's performances.

Mean Average Precision

In this context, a typical evaluation metric is embedded into **mean Average Precision** (`mAP`). Following the definition provided in [63] within the framework of object detection, and as described in [64], provided:

- a set of classes $q_i \in Q$;
- a set of k relevant bounding boxes $B = \{\mathrm{b}_1, \mathrm{b}_2, \ldots\ \mathrm{b}_{m_i}\}$ coming from the testing dataset;
- Model's precision for class q at box b_{k_i}, k $\in$ [0, m] defined as $\mathrm{P}(\mathrm{R}_{q_i,k})$, where $\mathrm{R}_{q_i,k}$ is the set of retrieved results until box k [6];

Mean average precision reads:

$$mAP = \frac{1}{|Q|}\sum_{i=1}^{|Q|} AP_i = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{m_i}\sum_{k=1}^{m_j} P(R_{q_j,k}) \tag{4.8}$$

[6]this reflects the trend of varying the recall **by progressively adding relevant elements** which can be either retrieved or not, as mentioned further up in the paragraph

where |Q| is the cardinality of Q.
This means first calculating the **average precision AP** for class q by computing the area under the PR curve and then averaging over $q \in Q$.

In the context of adversarial attacks targeting misdetection, `mAP` is a **discrete performance indicator**. In fact, its overall **drop** may be affected as well by the contribution coming from precision drops when a retrieved element is non relevant (false positives), loosing the capability of **probing a mAP decline exquisitely due to recall decreasing**.
This could be addressed in two ways:

- **Dealing with single-class datasets**: it allows to **filter out network's detection** corresponding to classes other than the targeted one, preventing false positives: model's non-relevant - yet retrieved - predictions are not taken into account. This is the case of `INRIA Person` dataset employed for patch training;
- **Introducing a second evaluation metrics, patch Success Rate (`SR`)**: it's a an **element-wise counting of model's false negatives** and true positives in the presence of an adversarial perturbation.

F1 score

Another measure that can be extracted from Precision and Recall is the F-measure. It is written as the weighted harmonic mean between P and R [63]:

$$F = \frac{1}{\alpha \cdot \frac{1}{P} + (1-\alpha) \cdot \frac{1}{R}} \tag{4.9}$$

It equally weights P and R when $\alpha = 0.5$. By setting

$$\beta^2 = \frac{1-\alpha}{\alpha}$$

the condition $\alpha = 0.5$ means $\beta = 1$, which gives the name to the measure itself (`F1`) and reads:

$$F_1 = \frac{2PR}{P+R}$$

Note that by tuning α (or β) the metrics can emphasize Precision or Recall accordingly by unbalancing the weight itself.

Patch Success Rate `SR`

`mAP` measured for the provided experiments deals with applying the adversarial perturbation in a **compound fashion** over the **whole set of boxes** belonging to the target class prior to inference. The approach that has been followed with the second metric is slightly different: the adversary is applied to only **one object at a time**.
Given a testing dataset with N$_{imgs}$, and provided N$_{obj,j}$ boxes in image I$_j$, j ∈ {1, 2, ..., |N$_{imgs}$|}, patch Success Rate is calculated as:

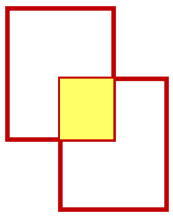
$$SR = \frac{\sum_{j=1}^{|N_{imgs}|} \delta_j}{\sum_{j=1}^{|N_{imgs}|} N_{obj,j}} = \frac{|\mathrm{fn}_{box,tot}|}{\sum_{j=1}^{|N_{imgs}|} N_{obj,j}} \tag{4.10}$$

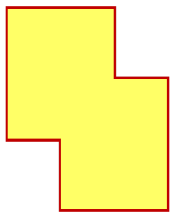
where $\delta_j \in [0,1]$ indicates if the target object in image jth is retrieved ($\delta = 0$) or not ($\delta = 1$). $|\texttt{fn}_{box,tot}|$ indicates the **total number of non-retrieved elements**, i.e. false negatives, on the dataset under analysis. Overall, `SR` actually plays the role of a **granular metrics** that directly counts adversary effectiveness.

It is calculated by considering the intersection over union `IoU` between:

- Predicted bounding boxes in the presence of an adversary, which targets one out of multiple objects in the image under analysis;
- The single clean reference box coordinates of the targeted object in absence of any perturbation, i.e. coming from **clean detection**. The latter is assumed as **ground truth reference** for patch evaluation;

Bounding box intersection Handling bounding box coordinates plays a key role for success rate calculation. Since it will be employed as well in part III, it is worth describing it in the following. Figs. 4.9(a) and 4.9(b) provide two samples overlapping boxes which are used as a reference to show intersection and union respectively.


(a) Overlapping boxes, **intersection**


(b) Overlapping boxes, **union**

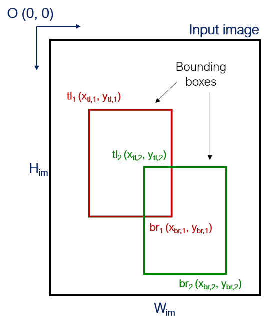

O (0, 0)
Input image
Bounding boxes
tl$_1$ ($x_{tl,1}$, $y_{tl,1}$)
tl$_2$ ($x_{tl,2}$, $y_{tl,2}$)
H_{im}
br$_1$ ($x_{br,1}$, $y_{br,1}$)
br$_2$ ($x_{br,2}$, $y_{br,2}$)
W_{im}
(c) Boxes coordinates **reference system**

Figure 4.9. Overlapping bounding boxes sample and coordinate reference system

Consider a set of N_{box} bounding boxes predicted on a perturbed input with coordinates B_i = $(x_{i,tl}, y_{i,tl}, x_{i,br}, y_{i,br})$, where subscripts *tl* and *br* mean **top-left** and **bottom-right** corners of ith box respectively, $i \in [0, |N_{box}|]$. The reference system has axes aligned with bounding box sides, and its origin lies in the top-left corner as shown in fig. 4.9(c).
The same notation applies for the **single clean ground truth box**, whose coordinates are B_{gt} = $(x_{gt,tl}, y_{gt,tl}, x_{gt,br}, y_{gt,br})$.

Provided input image *j*, `IoU` is computed by checking if there exists intersection between B_i and B_{gt}, $\forall$ i. B_{gt} is cloned $|N_{box}|$ times to allow **bit-wise (concurrent) comparison** and save computation time. This step would be particularly important in part III.
Henceforth, overlap check can be performed by probing the conditions for which:

1. $B_{gt} \cap B_i = \emptyset$ $\forall$ i, i.e. **non-intersection** case, or

2. $B_{gt} \cap B_i \neq \emptyset \ \forall$ i, **intersection** case;

The **first method** happens to be the most straightforward to implement, and it is shown in code 4.1:

Listing 4.1. Boxes negative intersection check

```python
def doOverlap(tl_1, br_1, tl_2, br_2):
    # Check if one rectangle is on the left side of the other
    no_int_left = torch.ge(tl_1[0], br_2[0]) | torch.ge(tl_2[0], br_1[0])

    # Check if one rectangle is above the other
    no_int_top = torch.ge(br_1[1], tl_2[1]) | torch.ge(br_2[1], tl_1[1])

    no_intersec = no_int_left | no_int_top

    return ~no_intersec_cond
```

where `torch.ge(...)`, `torch.le(...)`, `&`, `|` and `~` indicate bit-wise $\geq$, $\leq$, `AND`, `OR` and `NOT` operators, while $\mathtt{tl}_1$, $\mathtt{br}_1$, $\mathtt{tl}_2$, $\mathtt{br}_2$ are top-left and bottom-right `(x,y)` tuples of the first and second set of boxes.
The code simply checks if boxes extremities do not touch on the horizontal or vertical directions respectively. If it does happen, the result is inverted to provide a response about the intersection status.
The **second method** brings equivalent results, but checks for intersection directly. The ensemble of conditions becomes fairly harder to formalize and has **an additional cost of 4 `for` loops** over the vertices of B_i. Although the code will not be provided here, overlapping conditions are described below again with reference to figs. 4.9(a) and 4.9(b). For each vertex tuple $(x_{1,j}, y_{1,j})$ of b_1, j $\in$ [0, 4]:

1. **$\mathtt{cond}_A$:** check whether $x_{l,2} \leq x_1 \leq x_{r,2}$ `AND` $y_{l,2} \leq y_1 \leq y_{r,2}$ and vice-versa. There is an `AND` operator between the two opposite cases;

2. **$\mathtt{cond}_B$:** check whether $x_{l,2} \leq x_1 \leq x_{r,2}$ `OR` $y_{l,2} \leq y_1 \leq y_{r,2}$ and vice-versa. There is an `OR` operator between the two opposite cases;

3. **$\mathtt{cond}_C$:** check whether vertices do not fall inside the counterpart box and vice-versa. There is an `OR` operator between the two opposite cases;

4. **Final intersection condition `intersec_cond`:** it reads

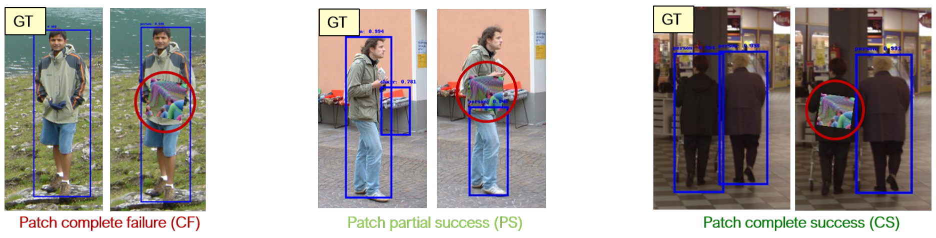
$$\mathtt{intersec\_cond} = \mathrm{cond}_A \ \mathtt{OR} \ (\mathrm{cond}_B \ \mathtt{AND} \ \mathrm{cond}_C)$$

When an intersection is asserted, it means that the **adversarial perturbation fails to provide misdetection**. The latter condition is further investigated according to the actual **grade of damage** caused by the adversary, which is obtained by `IoU` computation and the hard-coding of a threshold $t_{patch\,fail}$:

1. **`IoU` $>$ $\mathbf{t}_{patch\,fail}$**, fig. 4.10(a): adversarial perturbation makes no difference with respect to the un-perturbed case and produce high-valued overlapping. The set of over-threshold detection is named **Patch Complete Failure** (`CF`);

2. **`IoU` $\leq$ $\mathbf{t}_{patch\,fail}$**, fig. 4.10(b): perturbed and un-perturbed detected boxes do overlap, but the perturbation effects are not negligible and let a proposal with different aspect ratio to survive after inference anyway. The set is assigned the name **Patch Partial Success** (`PS`).

The last available case to probe, i.e. `IoU` $= 0$, is probed by checking whether the returned value from code 4.1 is `False` and the corresponding ensemble is assigned the name of **Patch Complete Success** (`CS`, fig. 4.10(c)). It counts the number of objects misdetection for each image. Being the perturbation applied one-object at a time, **it can count up to 1 for each input**.


GT
Patch complete failure (CF)
GT
Patch partial success (PS)
GT
Patch complete success (CS)

(a) Patch Complete Failure `CF`

(b) Patch Partial Success `PS`

(c) Patch Complete Success `CS`

Figure 4.10. Adversarial Patch Success Rate cases

Each inference generates a response $r \in \{$`CF`, `PS`, `CS`$\}$ associated to the targeted object. By including the newly introduced partial success ensemble, eq. 4.11 the overall `SR` is given by:

$$SR = \frac{\sum_{j=1}^{|N_{imgs}|} \delta_j}{\sum_{j=1}^{|N_{imgs}|} N_{obj,j}} = \frac{|CS| + |PS|}{\sum_{j=1}^{|N_{imgs}|} N_{obj,j}}, \qquad \delta \in [0,1] \tag{4.11}$$

while patch **Failure Rate** `FR` reads:

$$FR = \frac{\sum_{j=1}^{|N_{imgs}|} \delta_j}{\sum_{j=1}^{|N_{imgs}|} N_{obj,j}} = \frac{|CF|}{\sum_{j=1}^{|N_{imgs}|} N_{obj,j}}, \qquad \delta = \bar{\delta} \tag{4.12}$$

4.2 Fooling single-stage Object detectors: results

The majority of code concerning the practical implementation of adversarial attacks has been borrowed from [1], that applies it on `YOLOv2` as already mentioned and released it publicly. The reference code structure has been thereafter extended towards the simulation cases of interest in terms of NN typologies.
Two brief notes before proceeding:

Note #1: this section will report the results obtained employing the **loss function** in eq. 3.5, which shows the best results of the two. As a remainder, it minimizes the **maximum network prediction** within the ensemble of proposals.

Note #2: it is interesting to provide evaluations by addressing either **(1) objectness confidence score only** (obj approach) or **(2) final class confidence score** (obj-cls approach) as described further up in this chapter.
Even though from [1] it appears that obj method leads to better results in terms of adversarial capability - because it exclusively targets foreground/background distinction, i.e. bare misdetection - it seems that **the optimal choice strictly depends on the model under analysis**.

As a matter of facts, **mAP and SR assessments are performed with the best-case patch**, being it either obj or obj-cls. Some more specific network-related measurement under different adversary conditions is sometimes provided and explicitly mentioned.

4.2.1 YOLO family

Best-case trained patches

Following the procedure described in sec. 4.1.1, some structured ℓ_0 perturbation is trained with three different networks: YOLOv2, YOLOv3 and YOLOv4. For all the three of them, the best-case scenario happens by minimizing on obj only.
The obtained adversaries, which have dimension (3 $\times$ 300 $\times$ 300), are reported in fig. 4.11.

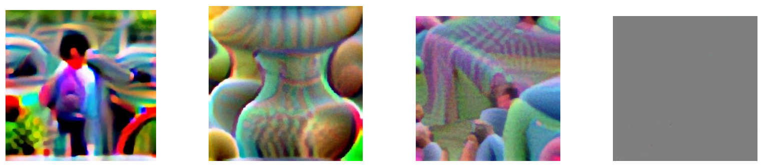

(a) YOLOv2 (b) YOLOv3 (c) YOLOv4 (d) Uniform gray (U_GRAY)

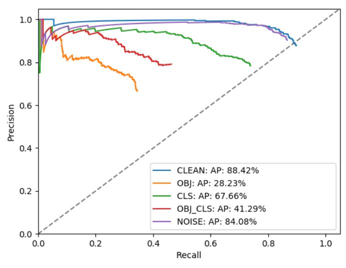
Figure 4.11. YOLO family trained patches, minimization over objectness

The last patch, called U_GRAY, is introduced as a **non-learned, uniform gray perturbation** to strongly assess the capability reached by a trained adversary in terms of adversarial robustness, following the trend .

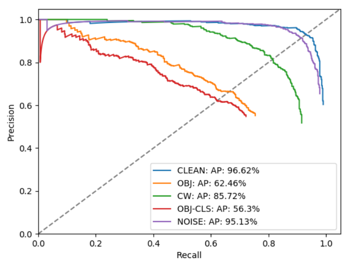
It is important to highlight again that these perturbations are **not hand-crafted**, but **come from a regression problem** that employs model's predictions to build knowledge over a set of pixel variables.

PR curves Recall drop, varying patches

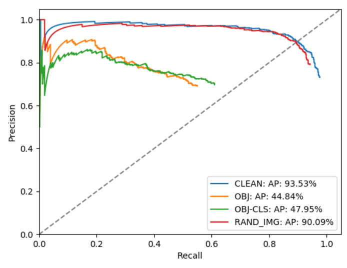
The primary analysis that has been done involves the computation of P-R curves under adversarial attacks with each of the three networks.
Figs. 4.12 displays the results after testing with several YOLO-based patches.



(a) `YOLOv2 P-R`



(b) `YOLOv3 P-R`



(c) `YOLOv4 P-R`

Figure 4.12. `YOLO` family PR curves evaluation under attack with **several patches**, `INRIAPerson` dataset

The curves are intended for a single-network inner comparison. The expected result is the **drop of the curve as an effect of misdetection (attack footprint)** with respect to the un-perturbed scenario (`CLEAN`). Some observations can be risen:

- **Best case loss** As already mentioned, minimizing on `objectness` only degrades network performances the most in terms of misdetection, in average. However, the effect is sharper with `YOLOv2` (mAP 28% - 41%), less definite with `YOLOv4` (`mAP` 41% - 44%) and reversed with `YOLOv3` (mAP 56% - 62%).

  For **`YOLOv2`**, results are **comparable** with those reported in [1] which indicates a mAP of around 25% [7].

  For **`YOLOv3`**, an interesting adversarial attack analysis exists in [2], as already mentioned. They target `MS COCO` dataset, yet the overall approach is equal.
  According to their results, `YOLOv3` (original `Darknet` weights from [24]) witnesses a drop in `mAP` from 74% to 36% [8].
  Even though the testing dataset is different, the methodology is **similar** - one class only is targeted - and therefore the results are formally comparable.

[7]The comparison is made under the same conditions, also because nearly anything has been changed from their reference code

[8]Success Rate is evaluated on physical attacks, i.e. printed patches physically applied on object or **paper dolls** such as printed versions of digitally perturbed images

- **Class only minimization** Minimizing over conditional class probability (`cls`) only does **not provide optimal results** in terms of model deceiving. The measurement has been performed with `YOLOv2` only (green curve) and reports `mAP` light decrease with respect to `CLEAN` (68%);
- **Carlini-Wagner loss** An experiment has been tried by minimizing Carlini-Wagner `CW` loss with `YOLOv3`, fig. 4.12(b). Since eq. 3.4 has been thought with misclassification in mind, the expression has been adapted to the present case. In particular, the adversarial target $\mathrm{C}(\mathrm{x})_t$ is **background itself** that can be expressed as

  $$P_{background} = 1 - \mathtt{obj}$$

  Henceforth, the overall `CW` loss reads:

  $$\mathcal{L}_{CW} = \sum_{i=1}^{N_{prop}} \max(P_i, 0) - P_{background,i} = \sum_{i=1}^{N_{prop}} \max(\mathrm{obj}_i \cdot \mathrm{cls}_i, 0) - P_{background,i} \quad (4.13)$$

  As it can be seen, the result in terms of `mAP` (legend: `CW`) shows that an inner maximization among proposals as in eq. 3.5 is still the best choice in terms of attack effectiveness.
- **Random patch PR curves** PR curves obtained with `U_GRAY` patch (`RAND_IMG` in the legend) nearly overlap with the `CLEAN` un-perturbed measurements, showing that **each model seems able to preserve its accuracy** when the adversary is non-trained and randomly initialized.

Average Precision and Patch `SR` cross-testing evaluation

`mAP` metric evaluated on the `INRIA Person` single-class-capability dataset is measured by considering cross-evaluations. Each network is tested against perturbed images with the whole ensemble of **best-case patches** introduced in fig. 4.11. Measurements results are reported in table 4.2.

mAP (%)	**YOLOv2$_p$**	**YOLOv3$_p$**	**YOLOv4$_p$**	**UGRAY$_p$**	**CLEAN**
YOLOv2	**28.23**	74.07	76.50	**84.08**	**88.42**
YOLOv3	69.19	**56.3**	86.42	**95.13**	**96.62**
YOLOv4	63.12	69.03	**44.84**	**90.09**	**93.53**

Table 4.2. `mAP` cross-evaluation on `INRIA Person` dataset, `YOLO` family

`CLEAN` results are reported in blue for the un-perturbed reference. **Pure white-box detection** is highlighted in red: it indicates a network that faces a perturbation crafted by accessing **its own weights** at training time. Other measurements (in black) are the set of all complementary cross-combinations available.

As for the second evaluation metric, **Patch Success Rate `SR`** is assessed following the steps described in sec. 4.1.2. The most distinctive feature in terms of high level approach is that the

perturbation is **applied one object at a time** for each image and not over the whole set of bounding boxes (Person object) as for `mAP` assessment.
Having this said, fig. 4.13 shows a bar plot of the results for each `YOLO` family network.

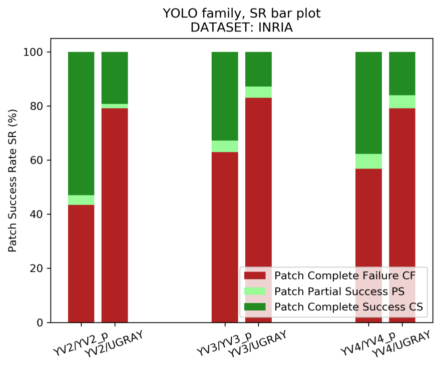


Figure 4.13. `YOLO` family networks, `SR` evaluation

The range of measures has been restricted to the **pure white box** evaluations along with `U_GRAY` evaluation for reference purposes (red and gray colors respectively in table 4.2).

Considerations

Provided the trustworthiness of both kind of metrics employed, though their methodological difference, some considerations may be pointed out:

- **Random patch impact** `U_GRAY` `U_GRAY` patch brings better results within the `mAP` metric domain, even though its effect is still less sharp than its trained counterpart, as expected;

- **Models comparison** `YOLOv2` appears to be the most damaged model of the family under attack, either in terms of `mAP` and `SR` assessments (refer to the pure white box attacks highlighted in red). Searching for reasons behind this behavior are out of the scope of this document, nevertheless some guesses could be made.
  YOLOv2 is a **fully connected NN** that does not bring **multi-scale information**, a feature introduced from `YOLOv3` (sec. 2.4.2). This increases the total number of network proposals after inference, making it more difficult for the adversary to suppress every bounding box surrounding an object.
  The explanation reflects the same difference already discussed between single- and two-stage detectors fooling, with the latter being an easier task due to former proposal filtering by RPN;

- **Black-box capability** Cross-evaluation measurements show that **black-box** attack typology is heavily affected by the network's weights employed for training, causing a drop in the patch effectiveness. However, `YOLOv2` patch seems to have the best impact of the three by looking at table 4.2. The reason may perhaps lie in the less number of total proposals as mentioned above.
  See appendix B for an attempt targeting adversary training on network ensembles to **improve black-box transferability**, as attempted in [2].

4.2.2 SSD family

The whole setup has been applied targeting `SSD` object detectors family. In particular, the adversarial patches have been crafted employing `SSD` trained on `VOC` with `MobileNetV1` and `MobileNetV2` as backbones. The reference repository is taken from GitHub. [9]
In addition, several `SSD` variations have been considered from the `TensorFlow Object Detection API Model Zoo` [10], with the following features:

- Complete pre-training on the `COCO` dataset - model already prepared and ready for inference;
- `MobileNetV1`, `MobileNetV2` and `MobileNetV3` [27] backbones;
- Resource-friendly (quantized versions or with `SSDLite` detection layer [15]) and full precision configurations.

The variety of configurations allows to test **adversary transferability among different backbones, dataset and model precision as well**.
Last, `SSDLite` is taken as a case study to test against several adversary training configurations for either `VOC` and `COCO` based implementations.

Best-case trained patches

Two patches are trained with `MobileNetV1_SSD` and `MobileNetV2_SSDLite` on `VOC` dataset. Uniform gray patch `U_GRAY` is employed as for `YOLO` family as well.

Note that `SSD` architecture encodes `background` as a standalone class. Therefore, given `n` classes for the corresponding dataset (e.g. n = 21 with `VOC`), `objectness` is targeted as

$$P_{obj} \; = 1 - P_{backgr\_cls} \; = 1 - \sum_{i \neq bakgr\_cls} P_i$$

where P$_i$ is the network output. Conditional class probability is therefore given by

$$P_{cls} \; = \frac{P_i}{P_{obj}}$$

Trained patches are reported in fig. 4.14

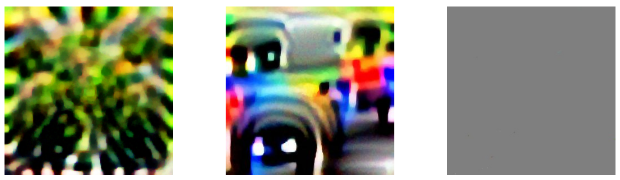

(a) `MBNTv1-SSD` (b) `MBNTv2-SSDLite` (c) `U_GRAY`

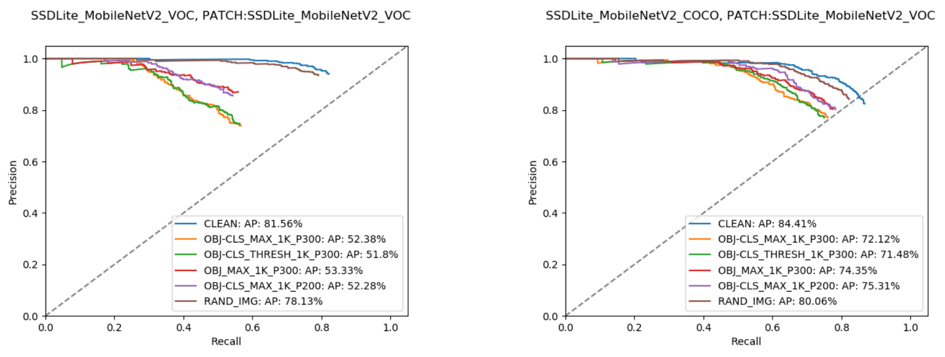
Figure 4.14. `SSD` family trained patches, minimization over `obj-cls`, `INRIAPerson` dataset

[9] https://github.com/qfgaohao/pytorch-ssd

[10] https://modelzoo.co/model/objectdetection

Adversary variants: `SSDLite` PR case study

Given the opportunity to assess adversarial results with respect to dataset changes in the SSD framework, a more detailed PR evaluation has been performed targeting `SSDLite` with `MobileNetV2` backbone from both `VOC` and `COCO` datasets.
In addition to `obj`/`obj-cls` and loss function changes, fig. 4.15 reports the results after varying the patch size from `(3` × `300` × `300)` to `(3` × `200` × `200)`.



(a) `MBNTv2-SSDLite` from `VOC` (b) `MBNTv2-SSDLite` from `COCO`

Figure 4.15. `MBNTv2-SSDLite` comparison between `VOC` and `COCO` dataset

Besides `CLEAN` and `U_GRAY`, Figs. 4.15(a) and 4.15(b) show several measures:

- `obj-cls` with adversarial loss from eq. 3.5;
- `obj-cls` with adversarial loss from eq. 3.6;
- `obj` only with adversarial loss from eq. 3.5;
- `obj-cls` with adversarial loss from eq. 3.5 and patch size `(1` × `200` × `200)`;

Some considerations:

- **Dataset transferability** It seems that dataset variation is not effective and follows the white-box constraint. The adversary is indeed more robust (`mAP` falls at around 50%) on the very network used for training than on the other trained with `COCO` (`mAP` at around 70 %);
- Patch features variations with **(1)** different types of losses, **(2)** minimization strategies (`obj`/`obj-cls`) and **(3)** patch size does not seem to change the adversarial effect by a **substantial amount**.

Average Precision and Patch `SR` cross-testing evaluation

Table 4.3 reports the results on `mAP` calculation after cross-evaluation with the best-case trained patches while fig. 4.16 shows **Patch `SR`** for pure white-box and uniform gray cases as well.

mAP (%)	**M1SSD**$_{VOC,p}$	**M2SSDL**$_{VOC,p}$	**UGRAY**$_p$	**CLEAN**
M1SSD$_{VOC}$	**46.5**	61.19	**75.97**	**80.46**
M1SSD$_{COCO}$	63.78	68.65	**79.2**	**84.41**
M1SSD_Q$_{COCO}$	69.94	76.98	**81.61**	**85.73**
M2SSDL$_{VOC}$	64.81	**52.38**	**78.13**	**81.56**
M2SSDL$_{COCO}$	74.42	72.12	**80.06**	**84.92**
M2SSD$_{COCO}$	69.45	68.76	**79.53**	**83.48**
M2SSD_Q$_{COCO}$	73.41	71.08	**79.63**	**83.94**
M3SSD_l$_{COCO}$	76.81	78.79	**82.4**	**87.31**
M3SSD_s$_{COCO}$	63.63	68.0	**74.1**	**82.14**

Table 4.3. mAP cross-evaluation on INRIA Person dataset, SSD family

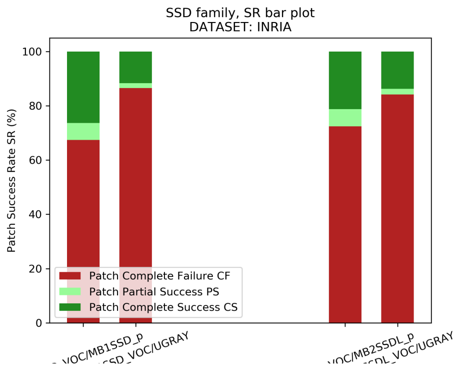


Figure 4.16. SSD family networks, SR evaluation

The overall setup and notations are the same employed with the YOLO family.

Considerations

Random patch (U_GRAY) impact and black-box capability lead to comparable observations with those provided when addressing YOLO family, while models comparison deserves a little bit of attention more.

Models comparison Table 4.3 summarizes the complete set of possible cross-combinations after inference. The most notable points to point out read:

- Pure white-box attacks lead to the highest mAP and SR drops. From fig. 4.16 however, it emerges that patch Success Rate is less sharp than YOLO, especially YOLOv2 network (SR lies in the range 20% - 30% for MobileNetV1 and MobileNetv2 backbones respectively);

- **Quantization** TensorFlow `ModelZoo` provides some quantized model following the procedure known as 'quantization-aware training' (mentioned in sec. 7.1). This allows to test the impact of the adversary against different kinds of **numerical precision**. The result is a small change on the fixed point model with respect to its floating counterpart (rows 3 and 7 in table 4.3).
  What is less expected is the direction of the change: `mAP` increases when low precision models are addressed for both the attempted measures;

- **Model architecture** Backbone typologies as well as the compact `SSD` model proposed in [26], which addresses resource efficient networks, keep the results against adversaries mostly unchanged. This is reasonable when considering that an efficient model size should imply steady accuracy performances by definition, as it is the case for **depthwise separable convolution strategy** applied within `MobileNet` and `SSDLite` frameworks (sec. 7.1);

- **MobileNetV3** Even though the two provided versions of `MobileNetV3` (small and large as in [27]) have similar clean `mAP` on the `INRIA` dataset (82% and 87%), their behavior under attack differs by a more substantial amount (nearly 20%).

4.3 Fooling face detectors: Multi-Task CNN

Face detection research field leads to introducing several modifications with respect to multiple-class based object detection. Two main differences are reported:

1. The task reduces to a **binary classification problem**: face/no-face. Even though bounding box regression and `objectness` score are still evaluated to allow object localization, the NN model is not trained on 'classes', but should be able to **distinguish between the background and foreground**, with the latter corresponding to a single class-like, i.e. faces;

2. Model's architecture in the present case is quite different than single-stage detectors described in chapter 2.

Given the aforementioned novelties, adversary training in the attack framework is modified and faced accordingly, either for what concerns geometrical affine transformations (sec. 4.1.1) and adversarial pixel training stages.

`MTCNN` is a SOTA model in the field of face detection and performs the task following the general setup already introduced in chapter 2 - bounding box and confidence score regression, in addition to score-based filtering and Non-maximum suppression (NMS) as post-processing processes.
Nevertheless, as the name suggests, three networks are employed to detach the tasks [65]:

- **Proposal Net (`P-Net`)**: the input image is resized to different scales in order to build an image-pyramid in a `SSD/YOLOv3`-like fashion. Bounding boxes are proposed and candidates are further filtered through `NMS` and calibrated using bounding box predictions;

- **Refine Net (`R-Net`)**: `P-Net` output are again filtered to suppress false positives. Redundant boxes are deleted again via `NMS`;

- **Output Net (`O-Net`)**: it is responsible for the final refinement and **landmarks predictions** (eyes, mouth extremities and nose for a total of 5 positions);

Trained patches

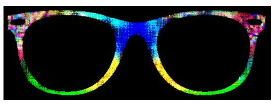
Two kind of patches are attempted in the framework of face detection:

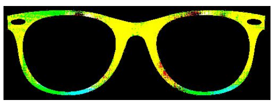
1. **Mouth patch:** A rectangle-shape adversary ($3 \times 250 \times 450$) that is located on the mouth by using **extremities landmarks** as geometrical support. Scaling, translation and fixed-rotation for alignment purposes are employed as described in sec. 4.1.1 by complementing the reference code released by [1] with **new functionalities**;

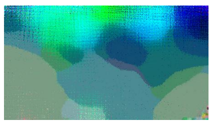
2. **Glasses patch:** A sticker representing a pair of glasses is employed targeting face's eyes taking inspiration from [43] that used it for face recognition (classification). The application step does not differ from the mouth-based case, where location landmarks are replaced with eyes instead of mouth extremities.

   Nonetheless, this second case implies a **novel challenge** in terms of perturbation training, since only a **subset** of the patch pixels should be updated by the `ADAM` optimizer.
   It is not possible to selectively compute loss gradients on a subset of pixels, therefore the path of **suppressing gradients** after computation through the support of an **external mask** of zeros is followed, allowing to keep regions that should be **inactive** in terms of adversarial capability.

The set of patches that have been trained is shown in fig. 4.17, while fig. 4.18 reports an example of actual application after affine transformation.


(a) Glasses, target P-Net


(b) Glasses, target all nets


(c) Mouth, target P-Net

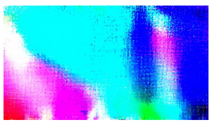
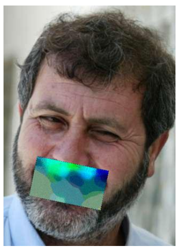

(d) Mouth, target all nets

Figure 4.17. `MTCNN` trained patches, `FDDB` dataset


(a) Mouth patch

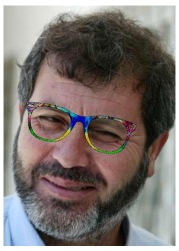

(b) Glasses patch

Figure 4.18. Patches affine transformation and application over faces, `FDDB` dataset

Four adversaries are crafted, given the multi-stage structure of the targeted model, providing several possibilities when assessing the cost function. Its general structure however remains either as in eqs. 3.5 or 3.6, i.e. it targets confidence score predictions by maximizing or summing the squares over a filtered subset:

1. **`P-net` output target:** in this framework, the whole set of proposals before that filtering and refinement processes are undertaken is attacked at training stage.
   Moreover, since prediction candidates are **extracted from each scale** of the image pyramid, a previous operation is required over scales.
   For each scale, the maximum approach over the corresponding number of proposals is performed. These maximum values are then passed to a subsequent maximum operation at scale-level.
   The final adversary cost function reads (eq. 4.14):

$$\mathcal{L}_{Pnet} = \max_{scale\,j,\,Pnet}\Big(\max_{box\,i,\,P-net}\big(\mathrm{f}^{scale\,j}_{box\,i,\,Pnet}(I,P)\big)\Big) \tag{4.14}$$

2. **Whole model output target:** in this context, the final loss has contributions from either `R-` and `O-` nets, which are assigned the same weight. The aim is to extend the attack - at adversary training stage - to an increased number of proposals, at the cost of introducing some redundancy. Results show that patches crafted in this manner behave **slightly better**.
   The final loss takes the form (eq. 4.15):

$$\begin{aligned}\mathcal{L}_{all\_nets} = \max_{net\,k}\Big(& \\ &\max_{scale\,j,\,k=Pnet}\Big(\max_{box\,i,\,Pnet}\big(\mathrm{f}^{scale\,j}_{box\,i,\,Pnet}(I,P)\big)\Big), \\ &\max_{box\,i,\,k=Rnet}\big(\mathrm{f}^{k=Rnet}_{box\,i,\,Rnet}(I,P)\big), \\ &\max_{box\,i,\,k=Onet}\big(\mathrm{f}^{k=Onet}_{box\,i,\,Onet}(I,P)\big) \\ &\Big)\end{aligned} \tag{4.15}$$

As a last note, all the patches have been trained on **`FDDB`** [11] **dataset**, which contains about 2800 images of multiple and single faces on different poses and sizes. The dataset has been **filtered to preserve images with no more that 5 faces** and split in a train and test dataset for adversary training and assessment (2000 and 768 images respectively).

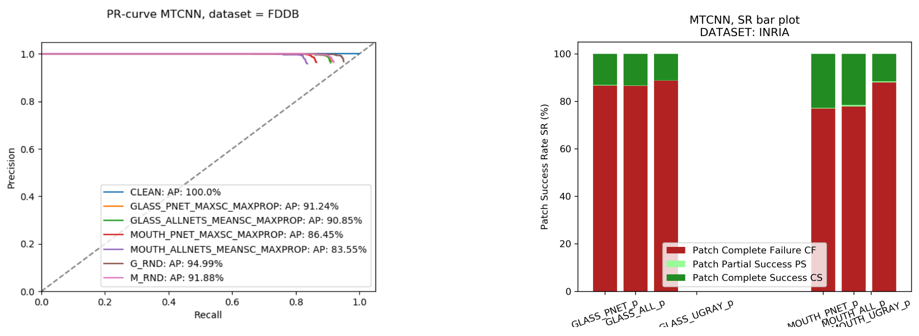
Average precision and Patch `SR` evaluation

As a usual approach by now, `mAP` and `SR` results are shown in table 4.4 and fig. 4.19(b) respectively. In addition, `PR` curve is reported as well in fig 4.19(a).

mAP (%)	**GLASS_PNET$_p$**	**GLASS_ALL$_p$**	**UGRAY$_p$**	**CLEAN**
MTCNN	91.24	90.85	**94.99**	**100**
	MOUTH_PNET$_p$	**MOUTH_ALL$_p$**	**UGRAY$_p$**	**CLEAN**
MTCNN	86.45	83.35	**91.88**	**100**

[11] `http://vis-www.cs.umass.edu/fddb/`

Table 4.4. `mAP` cross-evaluation on `FDDB` dataset, `MTCNN`



(a) `PR` curve

(b) Success Rate

Figure 4.19. `MTCNN` `PR` and `SR` evaluation, `FDDB` dataset

Considerations

Note that `CLEAN` accuracy is indicated as a ground-truth reference, therefore providing 100% `mAP`. This is due to the lack of bounding box annotations in the `FDDB` dataset, which provides elliptical ground truths.
For what concerns the perturbation effect, it is clear from both metrics that the adversary does not provide as sharp impact as with object detectors. The trained patches have nearly the same effect as `U_GRAY`, which is un-learned.
Apart from the overall trend, loss minimization over the three-stages network provides slightly better behavior than targeting P-Net only, though at a **non-sufficient level of attack robustness**. This result is perhaps due to the **proposal redundancy** introduced by targeting all the three networks, since an increased number of proposals should lead to worse results according to what has been previously said (less boxes to attack means less complexity at training time).
Only one work does exist in the literature targeting adversarial attacks on `MTCNN` for face detection other than recognition (e.g. *Sharif et al.* [43]), i.e. the work from *Pautov et al.* [58]. It shows that **addressing P-Net only** allows to degrade the model's `mAP` consistently. Nonetheless, it targets physical attacks with a considerable small dataset for either patch training and testing (around 20 image), **preventing a complete generalization** of the adversary capability.

4.4 Summary

This chapter reported the **experimental results** obtained by implementing adversarial patch attacks ($\ell_{0,struct}$ norm) against single-stage (`YOLO` and `SSD`) and face (`MTCNN`) detectors targeting misdetection. It introduced the general **adversary training flow** as well as attack's evaluation metrics in terms of **Mean Average Precision `mAP`** and **Patch Success and Failure Rates `SR`/`FR`**.

Results led to the conclusion that a **trained adversary**, within the **white-box domain** and against an *a priori* selected class, do **cause harm to a specific model** by a non-negligible amount (more than 35% drop in accuracy and an equivalent Patch Success Rate ratio), at least when compared with the effects of a **non-trained adversary** (`U_GRAY` patch).

Nevertheless, it has to be noted that experiments performed against face detectors provided poor results in terms of adversary effectiveness, as already detailed above.

Part III

Adversarial Defense

Chapter 5

Theory and Principles

The following chapter briefly introduces the novel topic of Adversarial Defense for ML systems, outlining how the need of its development has emerged (sec. 5.1) as well as its most recent advances and proposals (sec. 5.2).
Note that here the attention is again focused on techniques introduced and applied in the field of Computer Vision. In particular, while state of the art works have handled the topic of **pure classification**, the attempt of this document (detailed in chapter 6) is to introduce/extend the defense principles towards **single-stage object detectors** targeting **misdetection** against **real-world based attacks**, starting with a **non certified (heuristic)** defense typology.

5.1 Motivation

Some efforts have been provided in the last few years to **increase NN model's robustness** against adversarial perturbations, which is also an **intuitive reason for their development and study**.
In order to coherently provide a structure to the topic, an on-going and living document [42] actually stands as major guideline to follow in terms of theoretical and practical approach in order to **produce reliable results**. Overall, the most critical concepts that emerge are:

1. Providing a **justification** for the defense, i.e. why is it is important to **assure high model robustness**;

2. Defining the **threat model** to defend against, and clearly outline:

   (a) its **goal** in terms of prediction corruption;
   (b) its main features and **adversarial capabilities**;
   (c) the **level of knowledge** it possesses towards the attacked system;

While the second has been already described in chapter 3 , the first will be quickly summarized below.

Defense justification
In part II the existence of an adversary able to mystify the correct behavior of a system has been faced, by either theoretically describing adversarial results in the domain of image classification and directly performing experiments on object detector's deception by targeting **misdetection**. **Security drop** is one of the crucial consequences of model's poor performances. It implies the

existence of an adversary which is able to mystify the correct behavior of a system.
The latter case is much more **affine to real-world based scenarios** such as surveillance camera or autonomous driving corruption for **face and people detection/counting**.

Other two reasons can state the importance of assessing model's strength and may lead to deepen the field of adversarial defenses: the **worst case** robustness evaluation and **human-machine gap** [42].
The **first** does not imply actual concerns in terms of security, yet focuses on **testing** the actual system capability to fulfill its task. This helps providing an upper bound on its **performances stability**.
The **second** regards the actual gap existing between humans and machines in terms of reliability. Fully 'clean' results towards a certain task have been proved to keep tight the human/machine disparity. Yet, adversarial attacks exactly witness the profound **lack in term of decision-making process** [42] between humans and machines: **small or incognito perturbations can corrupt** a machine even without changing the average decision for a human.

Adversarial knowledge

In addition to what have been said in chapter 3, few words are spent regarding the degree of knowledge the adversary owns with respect to the targeted system. As many times highlighted, the adversary knowledge typically falls in the **white-box domain**, which means **full accessibility** to the model features. Moreover, it is worth mentioning that this knowledge holds in the field of adversarial defense as well: **there is no secret between attacker and defender** and the defense can be fully accessed by the adversary.
The latter is known as **Kerckhoff's principle**, an assumption of cryptography that states a defense effectiveness should not depend on the defender keeping some data secret to the adversary: the '*underlying algorithm*' should be known **but the encryption keys** [42].
This means holding some portion of data hidden to the attacker, following the double principle of **non-extractability and replaceability** of the secret, which should be either hard to draw out and easy to replace by the defender, if extracted.

It is anticipated here that the selected kind of adversarial defense **implies randomness as the encryption key** obscure to the adversary. There, the defense takes the form of structured ablation as a function of ablating **starting position**, whose optimal distribution should be known in advance for efficient defense.

5.2 SOTA walk-through: Adversarial Defenses typologies

Several adversarial defenses have been proposed against the **metric-based attacks** described in chapter 3 in the context of **pure classification**. The object detection domain implies the additional constraint of spatial localization and constitutes one of the **main bottlenecks** that is addressed in this document.

In general, adversarial defenses can be divided in two categories [66]:

- **Heuristic** (or empirical) defenses are experimentally tested, but not theoretically proved. This means that there is not a computed margin of reliability about their effectiveness that represents the **degree of distortion the defense can withstand**;

- **Certified** defenses are theoretically proved in the sense that they provide the degree of

distortion at which the defense it is assured to work.
A clear definition in these terms is given in [67] [1], which introduces:

1. **Certified robustness:** computed on a single input, it is the **maximum perturbation** ρ for which the model gives a correct answer under any adversary that meets the requirements imposed by the n-based distance metric: $|x - x'|_n \leq \rho$;
2. **Certified accuracy:** computed on the testing dataset, it is the fraction of input samples with **certified robustness** ρ **as a lower bound**;
3. **Median certified robustness:** computed on the testing dataset, it represents the median (higher) value of certified robustness across the set. Given input I_i and certified robustness ρ_i, it reads $\max(\rho_i)\ \forall\ i \in [0, N_{dataset}]$;

Defenses **with certification** are **preferred** to experimental ones due to their theoretically asserted validity within a computed perturbation range.
Up to now, certified defenses have been applied in the field of pure classification for CV though their use is more challenging against **real-world** based (or structured $\ell_{0,struct}$) attacks on real-time object detectors, which introduce non-predictable, hard to model elements (e.g. perturbation scale, spatial position, rotation, folds and so on) and have not been considered in this document.

With this in mind, a brief **digression on pure classification-based defense techniques** is provided in the following, with reference to the scheme in fig. 5.1.

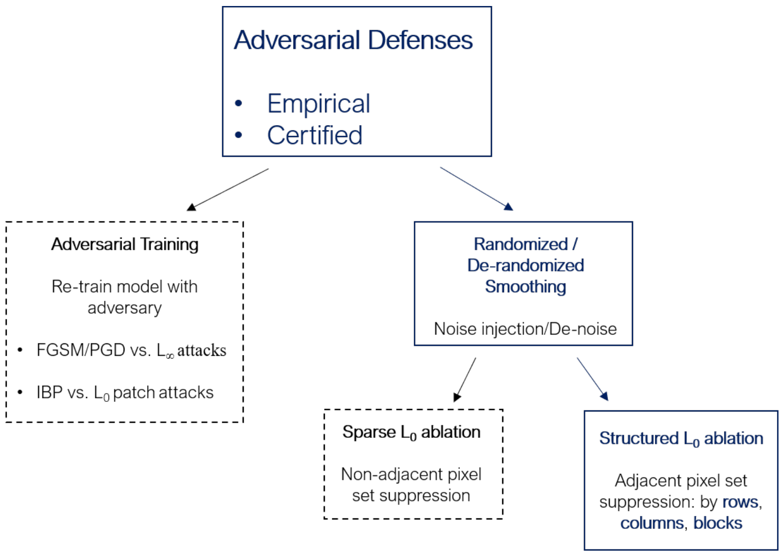

Adversarial Defenses
• Empirical
• Certified
Adversarial Training
Re-train model with adversary
• FGSM/PGD vs. L_∞ attacks
• IBP vs. L_0 patch attacks
Randomized / De-randomized Smoothing
Noise injection/De-noise
Sparse L_0 ablation
Non-adjacent pixel set suppression
Structured L_0 ablation
Adjacent pixel set suppression: by rows, columns, blocks

Figure 5.1. Adversarial defense typologies, high-level classification

Note Some defenses have been thought that try to detect the perturbation before letting the inference stage to reach out the output layer. In particular, two defenses of this kind have been proposed: **Digital Watermarking** (`DW`) and **Local Gradient Smoothing** (`LGS`).
The first deals with (classification) loss gradients while the second with image gradients: both

[1]The definition is independent from the targeted metrics.

concern unusual sharp variations in the region of the adversary. These defenses have been broken by *Studer et al.* [68], in particular `LGS` is confuted by inserting `TV` loss during perturbation training (eq. 4.3) as exploited in this work as well.

5.2.1 Adversarial training

This section will briefly review the most important works on adversarial training by highlighting the **underlying conceptual methodology**.
The intuition is to introduce adversarial samples in the NN model training process in order to reduce their overall impact on its performance. Formally, the optimization problem becomes a **min-max game** or saddle point problem of the form [49] [66]:

$$\min_{\mathcal{W}} \Gamma_{\mathcal{W}} \qquad \Gamma_{\mathcal{W}} = \max_{\rho \in \mathcal{S}} \mathcal{L}(\mathcal{W}, I_\rho) \tag{5.1}$$

where $\mathcal{W}$ represents the network's weights, ρ the perturbation itself and I_ρ the perturbed input image.
The **inner maximization** concerns crafting an attack to pinpoint the most effective adversarial sample while the **outer minimization** is performed through the usual gradient descend-based regression.
For what concerns some state of the art about adversarial training, *Goodfellow et al.* first made a proposal against Fast Gradient Sign Method (FGSM) in [50] by combining eq. 5.1 with the cost function in eq. 3.3 for ℓ_∞ attacks. Despite being effective against `FGSM`, the defense fails with stronger attacks such as iterative methods.

Madry et al. [49] proposed a defense against ℓ_∞ Projected Gradient Descend (PGD) attacks which show to be effective towards others ℓ_∞ adversaries, such as `FGSM` and Carlini-Wagner (CW). Being `PGD` computationally costly, it has the drawback of increasing the training time.

Studer et al. [68] employs adversarial training (or, better, certificate training) on **Interval Bound Propagation** (`IBP`) method to reproduce SOTA results against ℓ_∞ attacks and extend the method towards adversarial patch ($\ell_{0,struct}$) attacks in the field of pure classification.
`IBP` propagates the interval of allowed values for a certain output layer z^k after convolution from $\mathrm{k} = 0$ (first layer) to $\mathrm{k} = \mathrm{K}_{last}$. For ℓ_∞ attacks, the upper and lower bounds are defined by $\bar{z}^{(0)}$ and $\underline{z}^{(0)}$, while for $\ell_{0,struct}$ attacks they are **unaltered for the majority of image pixels** but the patch region, where they vary in the range [0,1].
As it can be noted, all of the mentioned techniques focus on network re-training without targeting image pre-processing. Moreover, all but the last are still empirical defenses.

5.2.2 Randomized smoothing

Another family of defenses that is gaining interest in the field in this field is called **randomized smoothing**. Up to now, some researches have been done against ℓ_2 and $\ell_{0,sparse}$ attacks. Again, this section will briefly review the most important works on this topic by highlighting the **underlying conceptual methodology**.

Randomized smoothing is a method for *'constructing a new - "smoothed" - classifier from an arbitrary base classifier'* [69]. The smoothed model receives as input a **noisy version** of the original one following a certain distribution (e.g. Gaussian or Laplacian).
A set of techniques specifically targeting image pre-processing already exists and provides padding, cropping and applying positive randomization (i.e. noise addition) or negative randomization (denoising, i.e. perturbation removal). They exploit the network capability to **withstand random**

features - also witnessed in part II - as a key element to extend its robustness against adversarial samples. The latter is the fundamental difference with respect to randomized smoothing defenses. Last, they are typically heuristic and not addressed in this document.

Lecuyer et al. [70] provided the first adversarial defense against classifiers targeting ℓ_2 attacks. *Cohen et al.* [69] improved the work by finding tighter bounds for certification purposes. The general method that is followed concerns letting the classifier to perform inference on a **sampled set of inputs perturbed by Gaussian noise** $\mathcal{N}(\mu, \sigma)$ (where μ is the mean and σ is the standard deviation). It is clear that two kinds of perturbation are in use:

- **Random noise perturbation:** the magnitude is encoded in the distribution chosen, being either Gaussian or Laplacian. It can be thought of a non-trained adversary, which does not cause harsh to the model's decision;
- **Adversarial perturbation:** it is the properly trained adversary that makes the decision to change according to a particular distance metric.

The general working principle flow is reported below, independently of the targeted attack metric:

Randomized smoothing process flow

1. **Training dataset**
   Experiments have been provided either on relatively small datasets, like MNIST and CIFAR10, or on SOTA ImageNet when dealing with classification purposes analysed up to now by the community;
2. **Noise introduction**
   Each input image is perturbed following a specific noise distribution. In [70], [69] the number of perturbed images, that are obtained from one single original input, is **sampled statistically** since the complete (deterministic) set of smoothed inputs produced from the original one is huge [a];
3. **Inference stage**
   Each noisy input is fed to the model at inference. This for sure **slows down** the overall **detection time** for each input, being all the 'smoothed' versions processed by the Neural Network.
   In the framework of **real-time defenses**, it is the key bottleneck to deal with.
4. **Majority vote decision**
   Defense effectiveness is performed through **majority vote**. In particular, the number of correct class decisions is counted and compared with the remaining (or the second highest class as in [69]) for both inputs x and x' (adversary).
   From the point of view of the defense alone, the compound statistics should **favor the correct class** when the adversary is injected.
   In terms of certification, if the noise perturbation between input x and x' is larger than the adversarial distortion between x and x', then x and x' will most likely produce the same smoothed sample set, providing robust classification.

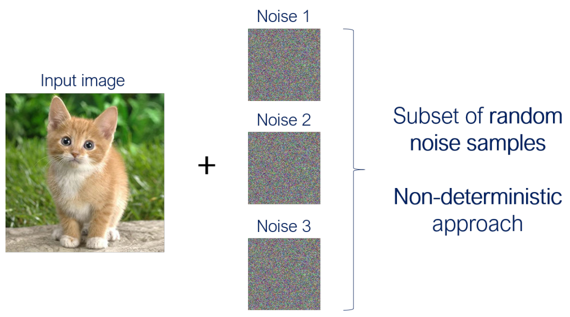
An interesting and peculiar proposal comes from *Zhang et al.* [71] that first inject Gaussian noise to x and x' and then perform k-means clustering to reduce the difference between the random distributions on x and x' (called Kullback-Leibler distance) without need for re-training.

[a]This may lead to define an approximated certification accuracy.

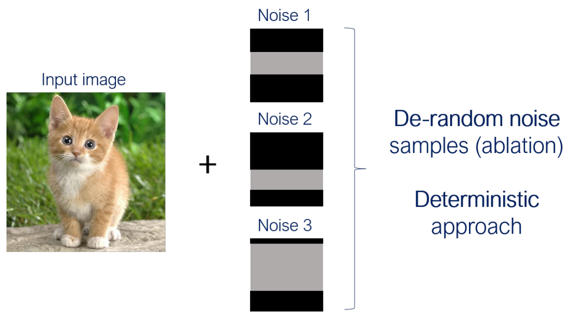
De-randomized smoothing

The aforementioned techniques do not target structured $\ell_{0,struct}$ attacks, the only capable of **modeling real-world scenarios**. Besides, they add **positive noise** to each input's pixel.
The complementary process, called **de-randomized smoothing** and introduced first by *Levine et al.* [67], **suppresses pixel values by reducing them to a unique, common ablated state**. It is indeed referred to as **ablation**. It is capable of transferring towards sparse and structured (real-world compatible) ℓ_0 attacks, as detailed in chapter 6.

Fig. 5.2 reports the major difference between positive and negative random noise injection (the latter case)



(a) Positive randomized smoothing



(b) Negative (de-) randomized smoothing (ablation)

Figure 5.2. Randomized and de-randomized smoothing approaches comparison

One last note is the recent approach proposed by [72], named **self-occlusion** and targeting pure classification as well. It works by minority vote: an occluding window slides at each position of the input image defining a new sample to feed the network with. Model's correct classification is then bounded to a reduced number of cases, i.e. those where the adversary is covered (occluded) by the sliding window.
Self-occlusion is quite **similar to de-randomized smoothing**, even though it does not aim at lowering the probability for the network to sample a patch, but acts in the opposite way.

5.3 Summary

This chapter introduced the field of adversarial defenses, highlighting the **motivations** behind its need - among the others - in terms of **security** for new edge-based applications such as surveillance camera and autonomous driving systems.
Thereafter, SOTA defenses against ML models are described. Among them, the most noteworthy is randomized smoothing, a technique that implies training a **smoothed classifiers/detectors** in order to reduce the adversary's effect.
It is underlined how randomized smoothing research has fallen in the domain of pure classification only, and the technique is chosen and implemented in the next chapter to fulfill the task of defense against object detectors under attack.

Chapter 6

Crafting and implementing an Adversarial Defense

Chapter 5 provides an overview on the topic of SOTA **adversarial defenses** and the corresponding main features. In addition, it has been highlighted that the most targeted domain of study has been **pure classification** either with **empirical**, **certified** or **conservative** defenses.
This chapter is organized as follows:

- Sec. 6.1 provides a deeper insight into de-randomized smoothing defenses and justifies their choice to address the problem of **object detection for real-time applications**;
- Sec. 6.2 gives the overall experimental results obtained by implementing and characterizing the selected defense with SOTA `YOLOv3` detector;

6.1 De-randomized smoothing defense

Challenges dealing with the introduction of an adversarial defense in the field of object detection have been already mentioned and deal with the **additional bounding box and `objectness` confidence regression problems** the network has to face in order to **properly locate and classify multiple objects** in a given input.
The open source, living document on adversarial defenses [42], clearly makes the point of properly defining the **attack threat model** features to defend against. For the present work, these are defined at the beginning of chapter 5: structured ℓ_0 (or patch), digital, white-box attacks.
In this context, the already mentioned work by *Levine et al.* recently extended their previous study on ablation defenses against sparse ℓ_0 attacks [67] **towards patch-based threat models** in the field of pure classification [3].

The latter has been taken as a reference framework in order to **craft an adversarial defense for OD**, including the released code [1] for the image ablation part.

6.1.1 Pure classification problem

The approach followed in [3] aims at ablating **structured portions** of the input image I by taking into account the equally structured nature of the adversary. They find this method to be

[1] `https://github.com/alevine0/patchSmoothing`

more effective than pure sparse ablation against patches, as shown below.
Assume:

- Input's dimensions as $(3 \times H_{img} \times W_{img})$, with $N_{I,pixels}$ set of input pixels and $|N_{I,pixels}| = H_{img} \cdot W_{img}$;
- a square perturbation of size $(3 \times H_P \times W_P)$, with $N_{P,pixels}$ set of patch pixels and $|N_{P,pixels}| = H_P \cdot W_P$.

Considering a set of $N_{retained}$ **sparsely** retained pixels at inference stage, the probability $Pr(A)$ **that one of the $N_{P,pixels}$ pixel is not preserved** (i.e., it is ablated) at inference is given by [67]:

$$Pr(A) = \frac{\binom{N_{I,pixels} - N_{P,pixels}}{k}}{\binom{N_{I,pixels}}{k}} \approx k\frac{N_{P,pixel}}{N_{I,pixel}} \tag{6.1}$$

that is the number of combinations needed for grouping all but the adversary's pixels in k partitions over the combinations of grouping all the input pixels in k partitions. As a consequence, the probability $Pr(\bar{A})$ that **one of the $N_{P,pixels}$ is retained reads $Pr(\bar{A})$ = 1 - Pr(A)**.
Applying a sparse defense against a structured attack leads to poor results, hence the proposal of structured ablation against patches in [3]. The authors selected three main kind of organized ablation:

1. **Rows ablation:** the preserved set of pixels is an horizontal band sampled along the image height;
2. **Column ablation:** the preserved set of pixels is a vertical band sampled along the image width;
3. **Block ablation:** a combination of the previous two keeping a block square windows from the original input;

Similar calculations as in 6.1 provide a **decreased theoretical probability** for the preserved window to sample the adversary with respect to the sparse case.

This kind of 'organized' ablation is shown to be able to provide not only clean, but also high certified accuracy as well on several SOTA classification datasets such as `MNIST`, `CIFAR10` and `ImageNet`.

6.1.2 Object detection problem

Two main differences exist between the work provided in [3] and the actual framework under analysis:

- **Threat model** Even if dealing with adversarial patch attacks, the setup introduced by *Levine et al.* considers **small input images and equally small adversarial perturbations, whose dimension is known *a priori*** and axes-aligned with the input's sides. This allows to **unseal deterministic computations** about defense certifiability by exploiting the geometric structure of the perturbed input feed.

These assumptions **do not fit in the domain of real-world oriented attacks** theoretically described and practically evaluated in chapters 3 and 4 respectively: the adversary is scaled and located according to the target object, while EoT are applied introducing **non-predictable** rotation, brightness, contrast and noise adjustments.
This makes **more difficult to assess a certification criterium**, but does not prevent an extension of the defense principles to check its effectiveness even though not proven with respect to a specific adversary size margin;

- **ML task** Object detection adds a major constraint to the problem of pure classification, as discussed in chapter 2: **spatial location of multiple-objects**, which introduces `objectness` confidence score and bounding box regressions in the overall cost function calculation.

Overall, this means assessing the adversarial defense by ℓ_0 smoothing in two directions:

1. Keep network performances in term of detection **high enough when the defense by ablation is active and the adversarial perturbation is absent**, on a double-sided front: spatial coordinates and final confidence score computations, the latter expressed as in eq. 2.3;
2. Assure that the de-randomized smoothing effectively allows to decrease patch Success Rate `SR` when the **model is under attack**;

The path that has been followed **introduces network re-training on ablated images**, a strategy already followed in [67] [3] as well other randomized smoothing approaches like [69][70] addressing ℓ_2-norms.

6.1.3 Ablation defense: notation

Given an input image I and named A and R the sets of ablated and retained pixels respectively, the subsequent procedure allows to **perform structured ablation on *I***, providing I_A at output:

Adversarial defense - Image ablation

1. **Ablation state encoding**
   Following the approach proposed in [67], **image channels are doubled** passing from 3 to 6 for `RGB`, or from 1 to 2 for `grayscale`. With reference to the first case, this **uniquely encodes the ablated state as the tuple `(0, 0, 0, 0, 0, 0)`**, thus differentiating it from any other pixel intensity combinations belonging to the retained fraction of the image.
   Provided a pixel p_i, $\mathrm{i} \in \mathrm{N}_{I,pixel}$ and its `R`, `G` and `B` intensities, then for each channel $\mathrm{C} \in \{$`R`, `G`, `B`$\}$ of p_i it holds:

$$C' = \begin{cases} 0 & \text{if } p_i \in A \\ 1 - C & \text{otherwise} \end{cases} \tag{6.2}$$

   Note The effectiveness of doubling the number of channels has not been tested directly. Therefore, their introduction may not be strictly necessary to the purpose, and it should be further investigated.

2. **Retention zone (`RZ`)**

It states the number of k pixels to retain. When addressing different types of ablation, it reads:

$$RZ = \begin{cases} \text{ceil}((\beta_{RF} \cdot H_{img})) & \text{if } \textbf{rows} \text{ ablation} \\ \text{ceil}((\beta_{RF} \cdot W_{img})) & \text{if } \textbf{columns} \text{ ablation} \end{cases} \tag{6.3}$$

where β_{RF} is named **retention factor** and

$$\beta_{RF} \in [0,1]$$

It indicates the fraction of pixels to retain along the corresponding image dimension according to ablation type. If $\beta_{RF} = 1$ **the original, non-ablated image is recovered**.
When block ablation is performed, the retained block sides are calculated from both eq. 6.3 for horizontal and vertical dimensions.
Fig. 6.1(a) shows the aforementioned notation applied to an input image with ablation by rows.

3. **Starting ablation position**
As already mentioned, randomized smoothing in the form of ℓ_2 norm defense [70][69] has the disadvantage of **not being deterministic**: the exact probability of the smooth classifier (or detector in the present case) should be calculated on every randomized input I. Since dealing with Gaussian noise distributions, *Cohen et al.* in [69] provides a **sampling method for the noise based on Monte-Carlo technique**. A similar approach has been provided in the framework of sparse ℓ_0 defenses by *Levine et al.* in [67].

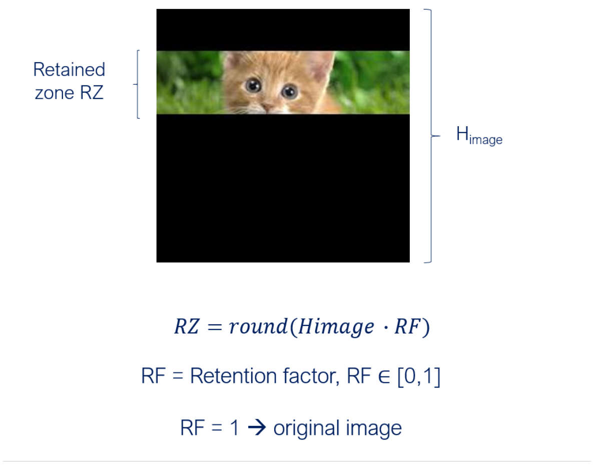
   Conversely, structured ℓ_0 defense further reduces the number of (de)randomized examples to feed the model, allowing a **complete study over the whole set of possible ablation states**. This fashion has been followed in the present work for evaluating the defense performances either with and without the perturbation by **sliding the starting position** $\mathbf{s}_{pos}$ according to the ablation type chosen, for each input I [a].

   However, the task of **real-time applications is not suitable for a deterministic application of the defense**, because it multiplies the number of inferences for each input image I, as described in chapter 7.

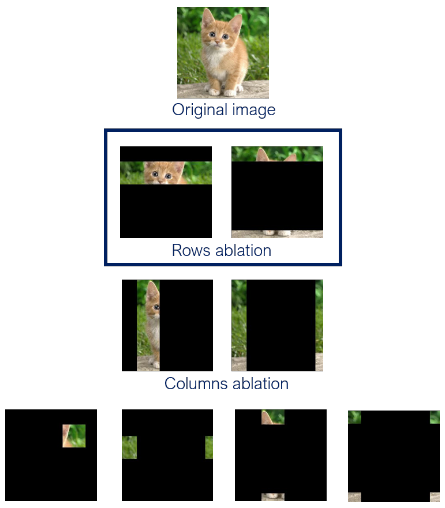
4. **Ablation boundaries**
If the sum of s_{pos} with the retained zone RZ exceeds image dimensions (either W_{img} or H_{img}) the retained zone is **wrapped around the image** itself. This allows to **take into account each ablation starting position** $\mathbf{s}_{pos}$ **as equally probable**.

   Fig. 6.1(b) shows that this solution induces the number of configurations for each typology to increase.

[a] The origin coincides with the top-left corner of the image as in fig. 4.9(c)



(a) Ablation notation



(b) Ablation typologies

Figure 6.1. De-randomized smoothing defense features

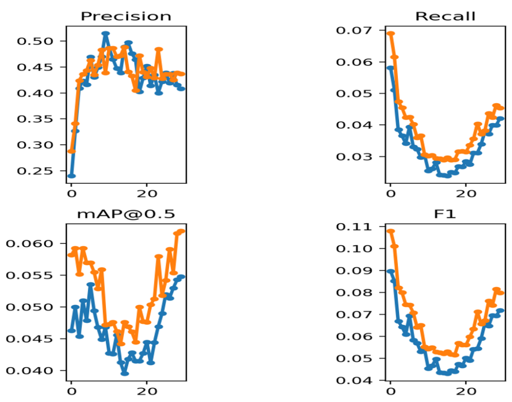
6.1.4 Network re-training

As assessed in [3], pure classifiers are re-trained on ablated images in order to strengthen and extend their knowledge towards partially visible inputs. The same approach is **borrowed for object detectors.**
From this point of view, it may appear that there is **no much difference between this kind of re-training and a data augmentation procedure** (chapter 1), because several techniques implies a similar **'hiding' approach** such as *Random Erase*, *Cutout*, *Hide and Seek*, *Grid Mask*, *MixUp*, *CutMix* [73].
Nevertheless, two key variations do exist:

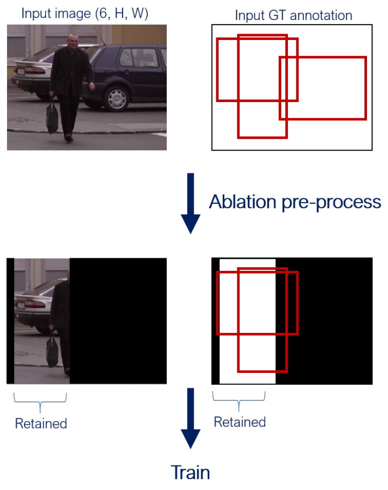
1. **Ablation is performed at inference time as well**, which marks a separation line with respect to standard data augmentation techniques;

2. In the framework of object detectors, it emerges the **need of ablating the ground truth annotations** that encode reference spatial coordinates to be used as teaching material at training time.
   The first experiment that has been done with image ablation re-training has indeed witnessed a large **decrease in model's recall** on the evaluation dataset (fig. 6.2(a)). This is not surprising because the actual multiple-objects framework may lead to the complete ablation of some image characters and of their spatial locations (encoded into the annotations) as well.

As a matter of fact, the following frame highlights the main steps towards performing ablation re-training and is sketched in fig. 6.2(b) for either images and annotated labels.



(a) **Recall drop** without annotations ablation



(b) Ablation training process flow

Figure 6.2. Recall drop after training and overall training flow sketch

Adversarial defense - Ablation re-training

1. **Image ablation**
   The first step consists in performing image ablation, as described further up in the text. The process can be dealt with in batches of inputs;

2. **Ground truth ablation**
   As mentioned, ground truth annihilation is needed to preserve good recall (and therefore `mAP`) performances and properly train the network. This step practically means to **calibrate the available teaching material** according to **what is left** effectively after the suppression process.

   This procedure is performed by **checking if an intersection** between the set of annotations and the fixed retention zone `RZ` exists. In order to do that, the same code in 4.1 is applied again in a **bitwise fashion**. The latter condition is **critical** because it allows to **handle the cutting out in a concurrent way**, making the impact of the whole ablation pre-processing module **negligible in terms of training time**[a].

   Fig. 6.2 gives a visual representation of both the previous procedures with column ablation.

3. **Starting ablation position**
   Training phase deals with batches of images. Therefore, borrowing from [3], ablation is performed with a **single position** for each batch of inputs to **save training computation time**. This means:

   (a) Applying the same kind of (de)noise - i.e. equal `RZ` - to several inputs in parallel;

   (b) The starting position is **handled and clocked by a common seed** that generates reproducible random numbers in order to fairly compare different ablation typologies under the same conditions - e.g. retention factors. This suggests that training time is treated via **probabilistic rather than deterministic ablation**;

[a]Some experiments dealing with non-optimized codes have even led to double the average training time.

6.2 Case study: YOLOv3

The network selected for characterizing the ablation is SOTA object detector `YOLOv3`. The baseline training and validation setup is borrowed from an existing repository[2] which exploits `PyTorch` ML framework. In particular, the analysis has been fragmented into two parts:

1. Subsection 6.2.1 treats network **re-training** on ablated images;
2. Subsection 6.2.2 covers the defense **evaluation** at digital level after re-training;

6.2.1 Re-training results

Re-training options are listed below:

- **Input image size:** `(3` $\times$ `416` $\times$ `416)`;
- **Activation:** `ReLu`;
- **Optimizer:** SGD with Nesterov momentum (appendix A);
- **Training Dataset:** `Microsoft COCO`, about 80000 images with 80 labeled classes;
- **Evaluation Dataset:** `Microsoft COCO`, about 40000 images;
- **Input batch size:** It has been set at 8 to fit one single GPU memory capability - no multi-processing is exploited;
- `GPU`: NVIDIA GeForce GTX 1080 Ti, without enabling multiple GPUs training;
- **Training epochs:** network's re-training is performed starting from a **pre-trained detector**.
  This means that the model has already been taught to localize and classify objects on the same dataset (for 300 epochs), thus re-training with ablation can be thought of as a **fine-tuning** [3].
  In this framework, it has been performed for each ablation typology (A_{set} = rows, columns, blocks) by:

[2]https://github.com/ultralytics/yolov3

[3]a term that typically indicates the **refinement and sharpening of a given model** on a peculiar and specific task, provided it has been already trained on a more **generalized** target. `Microsoft COCO` and its 80 classes provide a **generalized dataset**, because it does not supply infra-class variety, like e.g. `Google OpenImages` dataset

1. Exploiting **binary search** [4] to characterize the optimal retention factor `RF`. The following set $\mathrm{RF}_{set} = \{\ 0.25, 0.375, 0.5, 0.75, 1\}$ has been fully investigated $\forall$ `RF` $\in \mathrm{RF}_{set}$. The last case (`RF` $= 1$) corresponds to a further fine tuning of the network on the non-ablated dataset, and it is referred to as **baseline**;

2. Selecting first a training time of **30 epochs** for each analyzed case (ablation types and `RF`s). In order to have a more robust case study, a deeper insight has been provided by specifically training on **rows ablation for 100 epochs** with RF $\in \{0.375, 0.5, 0.75, 1\}$. This latter **training has a duration of about 8 days for each `RF` under the conditions previously mentioned**. Finally, it is the one selected to show the results of this section and characterize the defense.

Training results on rows ablation for 100 epochs are summarized in fig. 6.3.

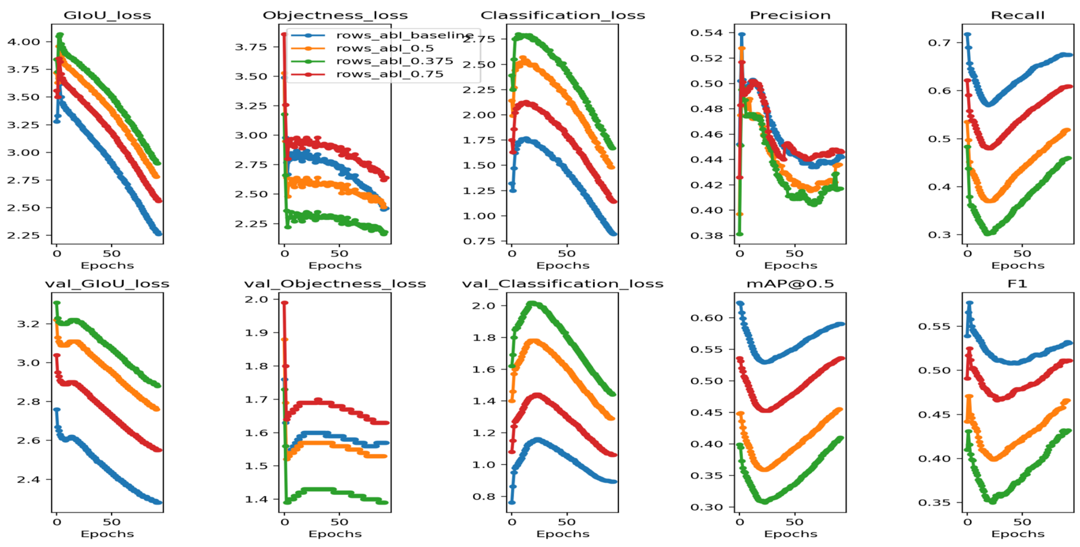


Figure 6.3. Training and validation tracking under **rows ablation** - `YOLOv3` on `COCO` dataset

It shows **training** and **validation** results in the first and second rows respectively. For each of them, it displays the three major loss contributions (sec. 2.4.3) as well as Precision, Recall, `mAP` and `F1` scores (sec. 4.1.2) evolution as a function of training time for different retention factors `RF`.

It is interesting to briefly **compare** the obtained results at varying `RF`, they can provide some hints on each ablation state by keeping in mind that **the fraction of preserved image increases** when moving from `RF` $= 0$ to `RF` $= 1$.

[4] also named **dichotomous method**, it consists of searching for a target by **exploiting half-by-half smart handling of selected intervals** in the range of interest.
In the present case, it is to be intended as retaining fractions of the input following the sequence: $1/2$, $1/4$, $1/8$ and so on, before proceeding with the analysis.

Loss contributions

The Bounding boxes regression in the form of generalized `IoU` and classification loss (sec. 2.4.3) show some common trends. Here are few comments:

1. The curves seem to increase first and then descend rapidly, where the effective learning takes place. This kind of behavior is perhaps associated to the pre-trained nature of the initialized weights, which have to 'get used' to **clash against and build learning from** an harder teaching material set than the one they have been trained on. **Harder does not mean simply different**, e.g. another dataset with infra-class specificity, but really a **more restrictive environment**.

2. The slope of the decrease seems to be quite similar for different ablations, even though the initial loss values are quite detached and tend to diminish as `RF` is tending to 1. Usually, there is no real interest in the absolute value of the loss, as long as it manages to converge and reach a stable state. Yet, different starting values for the penalization may indicate an **inner difficulty for the common set of pre-trained weights to gain knowledge when RF → 0**.

For what concerns `objectness` score regression, the trend seems to be reversed with respect to the other two losses. This may be due to the fact that, by reducing the retention size, the number of objects decreases because of ablation, thus providing **less contributions** to the squared sum in [22], which is the same employed in the following versions including `YOLOv3`.

Evaluation metrics

Graphs on the right in fig. 6.3 show Precision, Recall, mAP and F1 score for different ablation states. Here, the comparison with respect to the baseline network, that is trained with `RF` = 1 (without ablation) is useful to **look at performance evolution as a function of training time**.

Precision seems to not be affected by pixel suppression. It starts decreasing, possibly for the same reasons outlined previously, and ends by stabilizing for each investigated typology. The underlying message is that, provided a pre-trained network as a starting point, ablation does not affect model correctness, i.e. the **number of false positives remains steady**.

Recall represents the process bottleneck. **`mAP`** and **`F1`** depends on both P and R, and being P nearly steady with training moving forward, it is of **key importance** network performances will be bounded to recall drops.
Ground truth ablation introduction has helped building a formally correct smoothing process during training. In addition, an **underlying trend** does exist among the ablation states: **as `RF` → 1, recall tends to increase**. It seems reasonable since it would get more difficult for the network to distinguish between foreground and background as the **number of retained pixels is reduced**.
As an example of how much reducing the size of the retained window can affect network performances, experiments done with **block ablation training at `RF` = 0.25** have shown to cause loss divergence, defining a limit for that kind of ablation.

In addition to the previous two observations, it has to be noted that loss contributions from fig.

6.3 have not converged yet, even at 100 epochs. Note that the technique of early-stopping [5] is not employed to automatically end the training, preserving the original setup provided in the repository.
This means that better results **may perhaps be collected by training the model for a larger number of epochs**.

6.2.2 Post-training analysis

As previously mentioned, networks re-training has been performed with several ablation states through **binary search**. For what concerns **post-training** assessment, it is organized as reported in table 6.1.

Defense evaluation	`Patch OFF`	`Patch ON`
`Ablation ON`	Re-trained network	Re-trained network
`Ablation OFF`	Original network	Original network

Table 6.1. Defense post-training evaluation, cases under analysis

The entry encoded by the tuple `(Ablation OFF, Patch OFF)` corresponds to the **un-perturbed, un-defended case** and is labeled as **ground truth reference** for the evaluation, a way of proceeding strongly suggested in [42].
The **second row** indicates the same set of analysis that have been provided in part II when dealing with adversarial attacks, i.e. the comparison between perturbed and un-perturbed cases without the defense activated (`Ablation = OFF`).
The **first row** represents the **novelty** introduced with the defense. Here, two kinds of evaluations are necessary:

1. (`Ablation ON, Patch OFF`): it embeds an **inner comparison** among re-trained networks with ablation, in order to verify the **most suitable trade-off** in terms of **un-perturbed accuracy**;

2. (`Ablation ON/OFF, Patch ON`): the second column of table 6.1 concerns the **defense impact evaluation** in the perturbed domain, i.e. the system's behavior when the network is under attack and the defense mechanism is either active or not.

Testing dataset
Since the application field is real-time detection against real-world attacks, the testing dataset has been chosen as an **ensemble of frames** (about 100) coming from a **short hand-made video** which represents a single person moving towards the camera. This choice has the purpose of simulating what a NN model effectively sees at each instant of time in video streams applications. Even though several objects are present in the background of each frame, in the framework of defense evaluation there exists an interest **restricted to the single class** targeted under attack [6]. Every other class, following the approach described in part II, is **filtered out accordingly**.

[5]A technique employed to further **stop the training** of a NN model when some **trigger is reached**. Triggering conditions may interest either some specific loss values or, typically, performance measures in terms of NN accuracy.

[6]this feature serves as well as a **simplification** in order to handle post-processing evaluation easily.

These holds true not only for the perturbed case, but for the clean evaluation too, as detailed below.

Evaluation metrics
Two kinds of evaluations are implemented:

1. The first draws `PR` **curves** along with `mAP` on the provided dataset. However, such a compound metric does not help to display how near is the network after re-training to the ground truth reference: it **lacks *granular* information** on **spatial localization capability** as well as **class confidence score prediction strength** with respect to the ground truth;
2. **Granular, box-by-box evaluation** to address the two features mentioned in the previous bullet point, which is introduced in the results assessment;

Ablation position distribution
As described in chapter 5, the term **randomized smoothing** indicates injecting noise (or denoise when dealing with ablation) to the input before feeding it to the model.
Each random state applied to the input represents a variation with respect to the original state that should be evaluated through inference.
In the case of structured ablation, after having selected and fixed the retention zone size `RZ`, **each of this states** is obtained by choosing:

1. The ablation **starting position distribution**, that can be e.g. a uniform distribution if there is no preference among positions and each of them is assumed equally probable;
2. The **number of ablation states** tested, which is an additional **constraint** to be taken into account when dealing with real-time applications. It depends on the **acquisition frame rate** of the camera sensor, i.e. the time available between two captures.

All the experiments that have been performed **investigate the whole set of positions by assuming a uniform distribution over their ensemble**.

Fig. 6.4 shows a sample position distribution within the **time constraint** imposed by a real-time application. The latter influences the number of usable positions, which are chosen where the distribution provides optimal results.

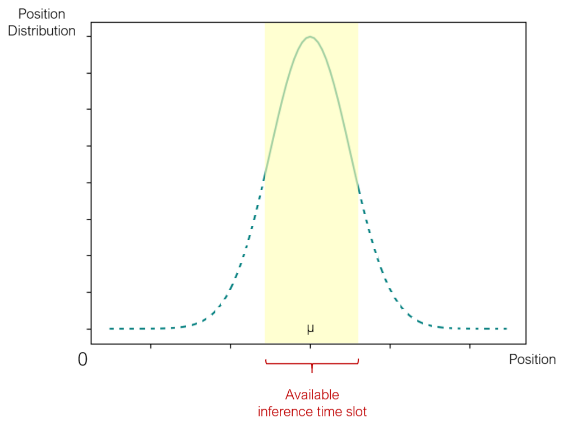


Figure 6.4. Position distribution sample, highlighting timing constraints for real-world scenarios

Notation Last, the original network will be indicated as `YOLOv3_orig`, while the re-trained ensemble as `YOLOv3_abl`$_X$`RF`$_Y$, with X $\in$ rows, columns, blocks and `Y` $\in$ [0,1].

First analysis: (`Ablation ON, Patch OFF`)

In this section, results are provided by evaluating on the whole testing dataset **and addressing re-trained networks on rows ablation for 100 epochs**.

Given an input image dataset D_{imgs} with $\mathrm{H}_{img,YOLOv3} = 416$ as NN input feed size, inference is run $\mathrm{N}_{pos} = \mathrm{H}_{img,YOLOv3}$ times at several `RF` values.
First, the re-trained network capability to mimic the ground truth reference model performances in terms of **information retrieval** is evaluated following the definitions provided in sec. 4.1.2. With reference to table 4.1, they are borrowed and **adapted** as follows:

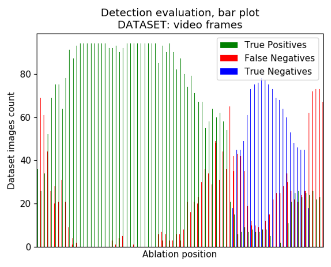
- **True positives:** the targeted object is correctly retrieved and classified;
- **True negatives:** background is correctly assigned non-object state;
- **False negatives:** the targeted object is not correctly retrieved;
- **False positives:** the targeted object is assigned a wrong class label. In the present single-class fashion, this last situation is not expected to assume values different from 0;

Within the ensemble of true positives `tp`, two entities are thereafter measured and tracked:

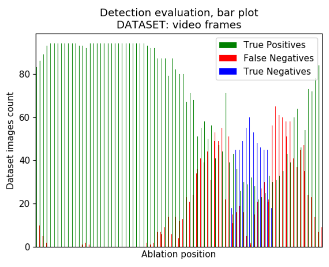
1. **Intersection over Union** `IoU` of the targeted class bounding box with respect to those predicted by the ground truth reference model.
   It answers the question: '*How good is the re-trained network (with ablation type X) at spatially detecting and locating an object when the same ablation type X is applied at inference time?*"
2. If the detection happens, independently of its spatial regression quality (a good or a bad detection), what it is measured is the **final class confidence score degradation** w.r.t. the ground truth reference model.
   It answers the question: '*How good is the re-trained network (with ablation type X) at assessing the final confidence score of a prediction when the same ablation X is applied at inference time?*'

The entire set of experimental results on the whole dataset D_{imgs}, for each ablation position and in the forms detailed above, reports **(1) predictions retrieval** assessment in fig. 6.5 and **(2) spatial localization and confidence score strength** assessments in figs. 6.6, 6.7 respectively.

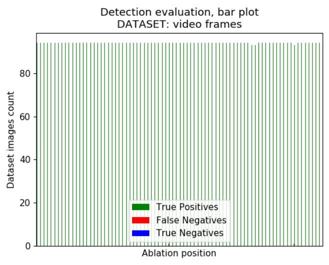
Predictions retrieval assessment



(a) `RF` = 0.375



(b) `RF` = 0.5



(c) **`RF` = 0.75**

Figure 6.5. Predictions retrieval, several ablation types, `RF` = {0.375, 0.5, 0.75}

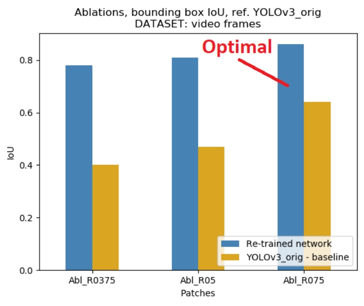
Figs. 6.5(a), 6.5(b) and 6.5(c) report a bar plot indicating evaluation statistics after information retrieval with three retention factors (`RF` = 0.375, 0.5, 0.75). Green, red and blue colors indicate **true positives `tp`**, **false negatives `fn`** and **true negatives `tn`** respectively [7] as a function of the ablation starting position, which slides along the image covering all the possible locations.
Since this is the un-perturbed case, the number of **(1)** true negatives and **(2)** false negatives should tend to 0.
The **first** is a drawback caused by the process of ablation applied within multi-objects scenarios, that can completely hide **small objects** for all locations.
The **second** is a defect coming from training with ablation. It may indeed happen that some bounding boxes are retained after ablation, but only a small portion of the object is effectively visible to the network, which **struggle to fulfill the learning process**, a lack that is reflected at training time.

In this context, the **optimal scenario** is `RF` = 0.75 (highlighted in bold in fig. 6.5(c)), with a negligible number of `tn` and `fn`. This result seems reasonable because with $^3/_4$ of the original image kept after ablation the network has **enough space to look at the majority of the input image**, reducing unwanted side cases.

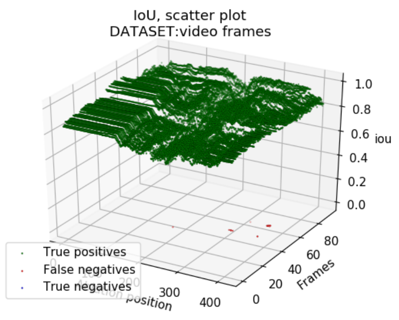
Spatial localization assessment

This paragraph shows the results obtained by evaluating spatial the NN model localization capabilities after the defense has been applied and switched on (`Ablation ON`).

[7]with reference to the target object, in a single-class detection framework



(a) Bar plot, re-trained `YOLOv3_abl`$_R$, `RF` = {0.375, 0.5, 0.75, 1}

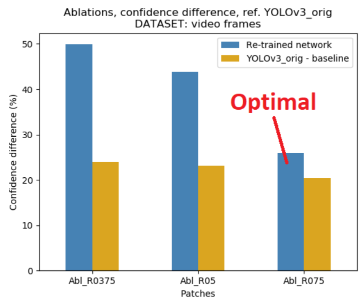


(b) 3D scatter plot, re-trained `YOLOv3_abl`$_R$, **optimal trade-off** `RF` = 0.75

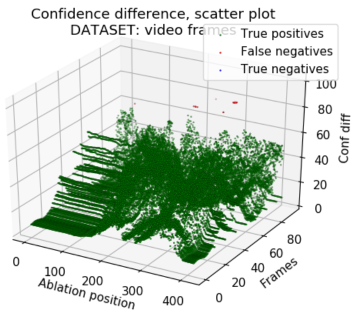
Figure 6.6. `IoU` evaluation

$$\mathbb{E}(IoU_{pos,D_{imgs}}) = \frac{1}{N_{TP}} \sum_{i=1}^{N_{pos}} \sum_{j=1}^{N_{TP}} IoU_{i,j} \cdot \Pr(\text{Position}) = \frac{1}{N_{TP}} \sum_{i=1}^{N_{pos}} \sum_{j=1}^{N_{TP}} IoU_{i,j} \cdot \frac{1}{N_{pos}} \tag{6.4}$$

Confidence score strength assessment



(a) Bar plot, re-trained `YOLOv3_abl`$_R$, `RF` = {0.375, 0.5, 0.75, 1}



(b) 3D scatter plot, retrained `YOLOv3_abl`$_R$ **optimal trade-off** `RF` = 0.75

Figure 6.7. `Confidence` score degradation

$$\mathbb{E}(S^{diff}_{pos,D_{imgs}}) = \frac{1}{N_{TP}} \sum_{i=1}^{N_{pos}} \sum_{j=1}^{N_{TP}} S^{diff}_{i,j} \cdot \Pr(\text{Position}) = \frac{1}{N_{TP}} \sum_{i=1}^{N_{pos}} \sum_{j=1}^{N_{TP}} S^{diff}_{i,j} \cdot \frac{1}{N_{pos}} \tag{6.5}$$

Considerations Bar plots from figs. 6.6 and 6.7 are structured as follows:

- Bars that indicate re-trained networks with several kinds of ablation (reported on the horizontal axis) are drawn in steel blue color;
- Fig. 6.6(a) shows network's capability to preserve optimal spatial location in terms of bounding box regression. Therefore, `IoU` is computed with respect to the ground truth case `(Ablation = OFF, Patch = OFF)` on the target single-class object;

- In a similar fashion, 6.7(a) reports how much, in case of true positive detection, the final class confidence score differs with respect to the ground truth case.
  Note that the last two trends **affect model's accuracy when the defense is activated** and looks after the ideal reference scenario of de-activated defense;
- Eqs 6.4 and 6.5 mathematically show how the values reported in figs. 6.7(a) and 6.6(a) are computed.
  The measured values of `IoU` and confidence difference are averaged by summing on each element in the true positive set, providing a mean value on the dataset.
  Thereafter, the final value depends on the position distribution, which can take the form of a **uniform density** $^1/\mathrm{N}_{pos}$ if there is no preferred position to access for ablation;
- In addition to the behavior of the re-trained networks, the **baseline's** (`RF` $= 1$) under the **same inference conditions** is appended as an extra comparison term (in gold color). The latter appears to be quite interesting because it **displays benefits and limitations** of the re-training procedure. Note indeed that:
  1. `IoU` drops when the baseline network is tested on ablated images, meaning that the non-trained network is unable to perform efficient bounding box regression.
     Conversely, re-trained networks show optimal results (more than 80%) w.r.t. the ground truth clean case: **ideally, the value should tend to 1**;
  2. `Confidence` difference shows that the baseline network is able to preserve tight gap (less than 20%) w.r.t the ground truth clean case: **ideally, the value should tend to 0**.
     The re-trained models perform **worse** in this context (peak of around 50% at `RF` $=$ 0.375), though reasonable results happen when the spatial context accessible by the neural network at inference time becomes large enough, i.e. at growing `RF`;

  The analysis allows to understand that the **optimal retention factor** among those tested is again `RF` $= 0.75$.
- Figs. 6.6(b) and 6.7(b) simply show a 3D scatter plot of the best performing trade-off case for both `IoU` and confidence difference, i.e. `RF` $= 0.75$, before computing frames average and position distribution.

Second analysis: (`Ablation = ON/OFF`, `Patch = ON`)

The second task involves **testing the actual defense effectiveness**.
In order to do that, the **perturbation is activated** and the following **measurements are performed with the best trade-off** `RF` $= 0.75$:

1. Assess patch Success Rate `SR` (sec. 4.1.2) of the original, **un-defended**, ground truth reference network `YOLOv3_orig`. Model's deceiving method should follow the directions depicted in part II;
2. Activate the defense and perform the same evaluation on the best re-trained model according to the previous analysis. Quantify the impact on patch `SR`.

Perturbation against un-defended network

The operative method is the same employed in part II. Being the defense de-activated, one single inference is performed over the whole dataset, from which it is possible to compute `Success Rate SR` metrics. The latter is shown in fig. 6.8.

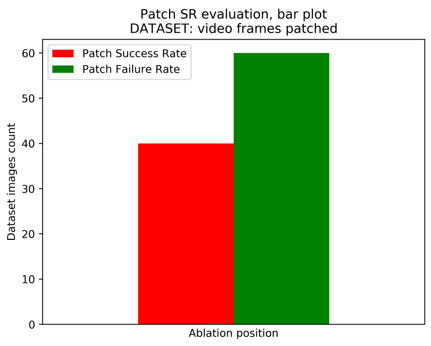


Figure 6.8. Patch Success Rate `SR`, Ablation = OFF, Patch = ON

This graph shows that the adversary **causes network deceiving with a `SR` = 40%**. This means that nearly half of the frames are misdetected according to the definitions introduced in chapter 4.
The next measurement introduces ablation defense and checks the same evaluation under attack.

Perturbation against defended network

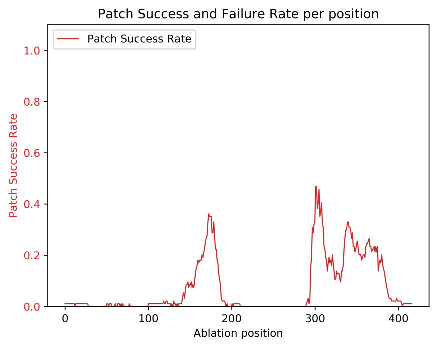
When the **defense mechanism is activated**, inference is performed for each image in the dataset and repeated for each position according to the chosen distribution.
The expected patch Success Rate is therefore obtained from:

$$\mathbb{E}(SR) = \sum_{i=1}^{N_{pos}} \Pr(\text{Patch Success} \mid \text{Position}) \cdot \Pr(\text{Position}) = \sum_{i=1}^{N_{pos}} SR_i \cdot \Pr(\text{Position}) \tag{6.6}$$

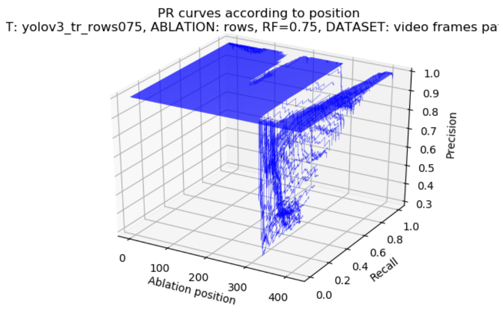
As previously mentioned, if there exists no preference for a particular subset of the available positions, a **uniform distribution** can be used:

$$\Pr(\text{Position}) = \frac{1}{N_{pos}} \tag{6.7}$$

In the latter case, the expected success rate takes the form of an **average over the number of positions**. Fig. 6.9(a) shows `SR` as a function of the ablation state (i.e. ablation position).



(a) Patch Success Rate SR vs. Ablation position



(b) mAP SR, Ablation = ON, Patch = ON

Figure 6.9. Adversarial attack assessment under defense, `Ablation = ON`, `Patch = ON`

Table 6.2 shows SR, expected SR and their complements (**patch Failure Rate** FR) for both the aforementioned cases:

Defense evaluation	Patch SR (%)	Patch FR (%)
Defense ON	**6**	94
Defense OFF	40	60

Table 6.2. Post-training evaluation under attack, DEFENSE ON/OFF

Applying the defense and performing the expected computation over single-position SR leads to a **decrease of the overall Patch SR from about 40% to 6%.**
The effect is associated to a **subset of optimal positions** where the adversary is not sampled by the network at inference time. It is clear that the results highly depend on the position distribution that is chosen, which is related to the dataset. A direct consequence is that model **performances under defense can be fine-tuned by selecting a proper position distribution**.

For the sake of completeness, fig. 6.9(b) shows adversary evaluation from the point of view of mAP assessment, reporting the same position-dependent behavior. Note that **there exists correspondence between the two evaluations** in terms of better and worse ablation locations.

Finally, it is important to highlight that **real-time capability is constrained** by the defense model that has been selected, which implies **multiple inferences** over the same input sample. These features will be addressed within the framework of real-time efficient Edge applications in the next chapter.

6.3 Summary

This chapter outlined the main features to be addressed in order to setup **structured ablation defense** against $\ell_{0,struct}$ adversarial attacks. Object detection introduces an additional constraint related to objects **location in space**, which is absent in the pure classification domain. As a consequence, neural network **re-training on ablated images and ground truth annotations** is performed before testing the defense at inference time.

The implementation of this defense is performed against structured ℓ_0 attacks targeting SOTA `YOLOv3` detector.
At the cost of constraining the results in terms of inference time, the defense leads to **lowering Patch Success Rate from 40% to 6%**, thus increasing networks robustness, on the tested dataset (frames of a short, hand-made video).

Chapter 7

Defense module Hardware implementation

This chapter introduces the field of **Edge AI** and highlights its growing importance inside the research community over several application fields (sec. 7.1).
Thereafter, it outlines the potential implementation of the defense described in chapter 6 as a pre-processing module at hardware level in sec. 7.2.

As a general scheme with respect to the present work, section 2.4 gave an insight to the most recent trends in the domain of object detection, giving in parallel an historical glance at its evolution. The most remarkable **take-away** reads:

Object detection is moving towards increasing model's inference speed - Frame per seconds (FPS) -, without damaging its accuracy [12].

Increasing inference speed means enhancing the number of inputs that can be processed by the NN model every second (or **frames per seconds FPS**), i.e. the model's working frequency. Its inverse indicates the average inference time, or **latency**, of the machine and it is expressed in seconds.
For a NN model to be feasible with real-time applications such that video streams, it should be able to work at a frequency $>$ **30 FPS** [1], which moves the operational choice towards single-stage detectors. As from chapter 2, this kind of detectors allows to reach real-time detection capability by facing either the classification and detection problems as a unique regression problem.
As reported in [12] there exists multiple aspects that can be controlled and optimized to fulfill the task either by acting at training stage or at inference's (*bags of freebies* and *specials*), as mentioned in chapter 2.

From the perspective of this document, which mostly deals with Adversarial ML in real-world based scenarios, it emerges the **critical need to handle real-time capable** machine learning models.
Conversely, the property of being **resource efficient** in terms of **model** size and compactness does have less impact on Adversarial ML framework itself. It is much more related to its actual **potential implementation** as a pre-processing module in a cloud-detached context. In fact:

[1] human eye's **limit**, even though it can be extended for more trained and focused eyes - e.g. airplane pilots

- **Adversarial attacks**. adversarial sample's crafting process heavily depends on the targeted model in a white-box attack (chapter 4).
  The model's choice is up to the attacker: either a floating point as well as a low-precision fixed point network could be used to craft the attack, which would still be **more effective on the exact network architecture it was trained with**, as further reported.
  Chapter 4 shows one example of attack evaluation against quantized and non-quantized model pairs with respect to an adversarial sample crafted with the same model architecture at floating point precision (sec. 2.4.2).

- **Adversarial defense**. It should be **model-agnostic** by definition. Therefore, network's model choice reflects the application itself, whether it is a real-time edge (such as smart vision on FPGA) or a computation unconstrained scenario.
  In addition, the defense should be **platform-agnostic** as well, i.e independent on the targeted board.

7.1 Machine Learning at the Edge

AI is witnessing a huge impact in the research community. In particular, applications such as computer vision, surveillance camera and autonomous driving are promoting the need of a **shift from centralized (or *cloud*) to de-centralized** (or *edge*) data acquisition and processing: this fashion takes the name of **edge AI**.
Edge AI brings several challenges over server-based systems, such as [74] [75]:

1. **Latency:** data should be acquired quickly for *in loco* analysis. Communicating with a server for data handling increases data transmission time by a non-negligible amount;

2. **Privacy and Security:** users privacy and data information security may benefit from the shift to the edge. In fact, sending sensitive information to centralized servers induces data manipulation which is hidden from the user knowledge. This may be reduced in the framework of data collection proximity to the sensor itself;

3. **Scalability:** it concerns the number of devices connected to the cloud. It can be a bottleneck if a large amount of devices is connected, as it may happen with the smartphones network which is growing larger and larger.

In order to develop Edge AI systems two frontiers should be addressed:

- **Deep Learning design** from the software side;

- **Hardware deployment** techniques from the implementation viewpoint on suitable **edge devices**. An edge device, also called leaf device, is a part of an AI system that contributes to its inner working. They can be divided in **(1)** devices with processing power, such as smartphones and boards and **(2)** devices without processing power, mainly responsible for data collection such as smart camera sensors and health monitoring systems. Edge devices can be CPU-, GPU-, FPGA- or ASIC- based;

In addition, others methodologies exist such as SW/HW co-design and Design Space Exploration for HW optimization [74].

In the framework of the present work, the targeted **edge device** is Lattice Semiconductor `ECP5` FPGA. It is not the most optimal device in terms of power consumption provided by the company, being overcome by iCE40 UltraPlus family [2].

For what regards Deep Learning software design, the contingent application of adversarial ML is quite **model-agnostic** as already mentioned for both attack and defense. This means that the choice of non-resource friendly models such as those from the `YOLO` family - large and requiring high computing capability - does not affect the results from a functional viewpoint.
Some useful software design methodologies are shown hereafter.

Trading off model size, accuracy and inference time

Deep and large NN models often benefit from a high performance accuracy. In addition, in the domain of CV and object detection single-stage models manage to keep **real-time** capability (high inference speed) as well.
Nonetheless, their **size** - which is affected by the number of convolution layers and indicating the dimension of the trained weights - **prevents them to be efficient or even implementable** for resource-constrained applications at the edge.
In order to make deep and large NN models feasible for this type of applications there exist some *size control techniques*, whose purpose is to generally optimized the models in terms of hardware application.
More specifically, NN size control techniques can be divided in two categories: : **model design** and **model compression** [74]:

- **Model design** involves the use of automated tools such as **Neural Architecture Search** (`NAS`) to find the optimal and hardware-aware solutions. It is the case of some SOTA backbones such as `MobileNetV2` and `MobileNetV3` that have also been tested in this document as well;
- **Model Compression** aims at reducing model size, which leads to low power and low latency (improved inference speed) performances.

Among model compression methods, **quantization**, **pruning**, **knowledge distillation** [75], can be mentioned in this context.
Quantization reduces the numeric precision of weights and activation functions (the network's parameters) from **floating point** (32 or 64 bit) to **fixed point** (8 bit) notations by multiplying for scale factor and subsequently rounding the result. This allows to save memory by **reducing data-width** at the cost of some accuracy drop.
Several quantization typologies exist [3]:

1. **Quantization-aware training:** floating values are rounded in order to **mimic fixed point values** during training. Nevertheless, the computation is performed with full precision values, hence the term "aware": the NN is made aware of the fact that the final model will be in fixed point precision;
2. **Dynamic quantization:** it involves weights and activation scale factor **tuning** during inference, i.e. at run-time;

[2] `http://www.latticesemi.com/Products/FPGAandCPLD/iCE40UltraPlus`

[3] `https://pytorch.org/docs/stable/quantization.html`

3. **Static post-training quantization:** scale factors for both weights and activation is **statically** determined by a **calibration step**: inference is run over a subset of the testing dataset (calibration set) in order to determine optimal values. The latter are used as fixed and steady for each following inference;

Pruning concerns suppressing some near-to-zero values, thus lightening the model size;
Knowledge distillation concerns training a smaller model based on the output of a larger, more powerful model which shares the same structure, thus defining an approximated version.
As shown in chapter 6, either quantization and hardware-aware automated design examples (`MobileNet`) under adversarial attacks are analysed when dealing with `SSD` variants. Overall, the adversary results **low precision-agnostic**.

7.2 Case study: Object Detection on FPGA

The reference platform and tools for this section come from Lattice Semiconductor and are briefly outlined in sec. 7.2.1 and in sec. 7.2.2.
Besides, sec. 7.2.3 describes the implementation of the adversarial defense detailed in chapter 6 as a standalone `RTL` module targeting the pre-processing step of an object detection flow. The module is **integrated** into an existing `RTL` design that exploits the pre- and post- processing steps of the detector itself.

7.2.1 Lattice Semiconductor's toolkit

Among the ensemble of tools provided by Lattice, a primary classification detaches between ML frameworks and `FPGA` design:

1. **`NN Compiler (Machine Learning Software 3.0)`** generates the firmware (or binary) file from the NN model. It natively supports `Caffe`, pure `Tensorflow` or `TensorFlow` with `Keras` backend as ML frameworks.
   It converts the architecture to a `Caffe` model by applying **static post-training quantization** (sec. 7.1) after proper calibration with a sample ensemble of images.

2. **`Lattice Diamond 3.11`** performs all the steps characterizing Semi-custom Design, from architectural level `RTL` code down to `Place&Route` (`P&R`). As a synthesis tool, it uses **Lattice Synthesis Engine** (`LSE`) that exploits `Synplify Pro` of the `Synopsys` family.
   The output file after `P&R` is a bitstream to program the `FPGA`.

In the framework of this project, `Lattice Diamond` has been used when dealing with pre-processing ablation module, while `NN Compiler` is involved less because the adversarial defense should be ideally model-agnostic.

7.2.2 Lattice Semiconductor's EVDK

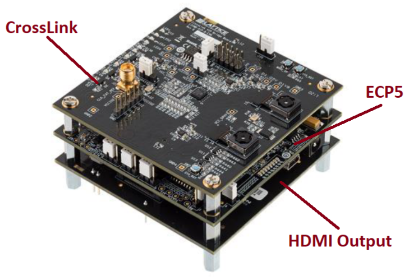
Lattice Embedded Vision Development Kit comprises three main platforms:

1. **CrossLink Input Bridge Board**, that handles the **frame preparation and clocking** from the input video stream. In particular, it takes raw data from both `CSI-2` camera sensors (`Sony IMX214`) and merges them through parallel logic in order to feed CNN Accelerator;

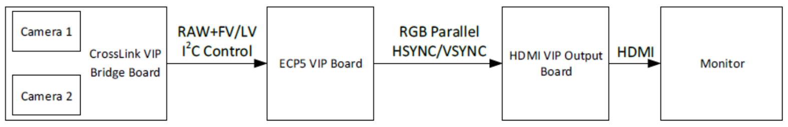
2. **ECP5 Processor Board**, that **embeds the FPGA**. It contains **all the pre-processing** needed to prepare the input before feeding it to the CNN Accelerator, which implements the inference stage and produces output predictions;

3. **HDMI Output Bridge Board**, that receives the input image and NN predictions in terms of bounding boxes and class confidence scores and displays it to a 1080p (Full HD) monitor;

The full board, which stacks the three aforementioned components, is shown in fig. 7.1(a) along with a black box system diagram (fig. 7.1(b))



(a) EVDK three-level board



(b) EVDK high-level block diagram

Figure 7.1. Lattice Vision Kit with `ECP5` `FPGA` and `HDMI` output capability [76]

A typical design flow for object detection needs two key elements, a **binary** (firmware) file embedding the NN model weights **frozen at inference**[4] and a **bitstream** file representing the `RTL` level design, where the CNN accelerator is instantiated as an `IP` [5] block.
The firmware file is loaded into an outer `SD` card and written in the external `DRAM` on `ECP5` board, while the input image is pre-processed as mentioned previously and stored either in external `DRAM` or CNN internal memory according to the training size.

When both the elements are available, the CNN accelerator has all what it needs to perform inference (both input and NN weights) after FPGA programming. The computed predictions are handled according to post-processing design.

For what concerns **resource availability**, it depends on the specific `ECP5` device selected. Table 7.1 shows the features of the board employed here, which belongs to the `ECP5-LFE5UM` family:

[4]This means that some layers such as `DROPOUT` and `BATCHNORM` used for regularization are removed with respect to the training model. This is performed either by setting the model in inference mode (**`model.eval()`** with `PyTorch` or **`tensorflow.keras.backend.learning_phase`** `= 0` with `TensorFlow/Keras`)

[5]Intellectual property

ECP5-LFE5UM85	**Availability**
Voltage	1.1 V
LUT	83640
Registers	83640
EBR	208

Table 7.1. Resource usage, ECP5-LFE5UM85 family

The number *85* in the name of the considered family board refers to the availability in terms of `LUTs` and registers, that is around 85000.

7.2.3 `RTL` pre-processing module

It is first useful to understand how the input image coming from `CrossLink` board is handled by `ECP5` intermediate board all the way up before feeding the NN accelerator for inference at hardware level.

Typically, the sequence of pre-processing steps for input preparation deals with pixel intensity adjustment and `(R, G, B)` channels extraction. It is shown in the following fig. 7.2:

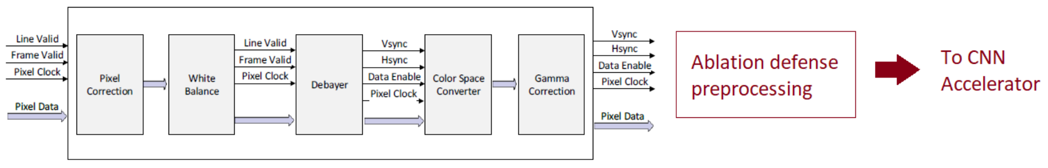


Figure 7.2. Input Image **pre-processing** steps, `ECP5` board [76]

These steps are briefly summarized in table 7.2 [76]:

Pre-process step	**Task**
Pixel correction	Repair damaged pixels from the camera sensors
White balance	Widen the range of each color via gain and offset controls
Debayer	Extract `(R, G, B)` data from raw input
Color Space Converter	Match real-world *gamut* via gain and offset controls
Gamma Correction	Match 8-bit depth of common displays with sensors larger depth

Table 7.2. Lattice ECP5-LFE5UM pre-processing steps

The defense pre-processing module is placed at the end of this chain, and should:

1. **Encode ablation** by doubling each pixel channel size;
2. **Suppress the pixel at the correct position**, according to the defense settings chosen by the user;
3. Have a **minimum impact** on the existing design with de-activated defense either in terms of **resource usage and delay**. In this context, a **combinational** module is sought after.

Ablation starting position
When performing defense simulations at software level from a static point of view [6] every position is checked and sampled with equal probability. This can not happen **without increasing the risk of preventing real-time capacity** due to the additional constraint of camera sensor acquisition frame-rate.

Therefore, some optimal position distribution that accurately takes into account the **trade off between performances and time** should be crafted:

1. The time constraint can be assessed by deciding the number of positions to be tested *a priori*. According to acquisition frame rate and accelerator inference speed, this number is defined as:

$$N_{ablations} = \frac{t_{acquisition}}{t_{inference} + t_{processing}} \tag{7.1}$$

   where $\mathrm{t}_{acquisition}$, $\mathrm{t}_{inference}$ are camera acquisition time and accelerator inference time respectively, while $\mathrm{t}_{processing}$ embeds contributions from either pre- and post-processing times. Supposing $\mathrm{t}_{processing} = o(\mathrm{t}_{inference})$ (i.e. $\mathrm{t}_{processing}$ negligible w.r.t. $\mathrm{t}_{inference}$), and given $\mathrm{FPS}_{inference}= {}^{1}\!/_{t_{inference}}$ and $\mathrm{FPS}_{camera} = {}^{1}\!/_{t_{camera}}$, eq. 7.1 becomes:

$$N_{ablations} \simeq \frac{\mathrm{FPS}_{inference}}{\mathrm{FPS}_{camera}} \tag{7.2}$$

2. The position distribution should be assessed as well. For the present implementation with one inference per frame in mind (therefore one ablation as well), the starting position is selected by a **pseudo-random number generator** that takes the form of a **Linear Feedback Shift Register** `LFSR` to evaluate the final resource usage.
   If some other distributions is chosen (within the time limitations, which shape the distribution by introducing boundaries), they should be implemented in hardware as well, following the same fashion.

Alternative for ablation starting position
One last note on this topic. Since **each frame bears high similarity with respect to its nearest neighbors**, the application of the defense following real-time restrictions can be thought as **distributing a single ablation state** over the whole set of positions N_{pos} **on the first** $\mathbf{N}_{pos}$ **frames** instead of applying it N_{pos} times on a single frame.
In this context, `LFSR` could be employed as well as described further down in the text.

[6] *static* means without considering the constraint introduced by the **camera sensor acquisition frame rate**, i.e. the time required for the camera to sample two adjacent frames.

User free parameters
The user is allowed to choose the **ablation typology** as well as the **retained zone size** for the suppression to be applied. The module **encodes actions** for the whole set of ablation cases shown in fig. 6.1(b).
In addition, if `LFSR` solution is selected, its length is chosen so that it can generate all the numbers in the range [0, H_{img}] in the rows ablation case. In general **the whole set of testing positions should be accessible** for the `LFSR`.

Ablation module
The process of ablation that is started immediately before feeding the accelerator - as shown in fig. 7.3 - can be handled in two ways:

1. Employing a **line buffer** to store re-scaled pixels along one line of the input;
2. Employing a **frame buffer** to store re-scaled pixels of the whole image;

The **second** method is not feasible in terms of resource usage since it exceeds the number of available Embedded Block RAM (EBR) due to **large memory depth requirement**, which equals $H_{img,inf} \cdot W_{img,inf}$. If $H_{img,inf} = W_{img,inf} = 224$, frame buffer depth exceeds 50000 rows.

In the **first** fashion, each accumulated pixel is doubled in terms of channel size (from 3 to 6) and then suppressed according to its position by employing a **supporting mask**.
When the ablation is activated, a line buffer in the form of a **true single port RAM** (`TSP-RAM`) is modified by **doubling its width** size accordingly. Input pixels are written into CNN accelerator internal memory and ready for inference.
The ablation module implements two counters that add up until $H_{img,inf}$ and $W_{img,ing}$ are reached, before restarting when the next frame is available.

Note that the process of pixel suppression is performed in a **fully combinational way** because it does not rely on data storage. This helps **not degrading performance speed** when the defense is activated.

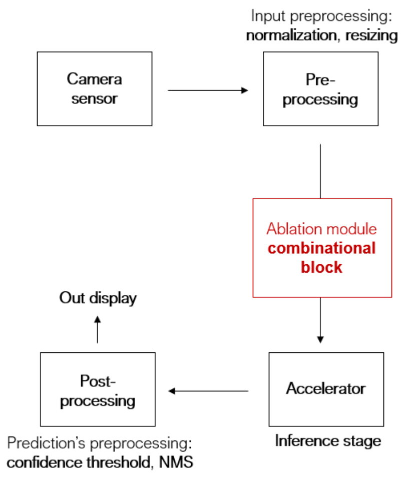


Figure 7.3. Object detection design, high-level diagram

LFSR

It is implemented as a standard **Fibonacci typology** and schematized in fig. 7.4. An LFSR bears several key properties:

- It generates pseudo-random numbers which are predictable by hand-calculation;
- It is characterized by flip-flops and XOR gates that are located to change the subsequent state. XOR gates in this configuration are called **taps**;
- According to the modulo-2 algebra (or Galois field GF), each state can be calculated from a specific polynomial whose **grade** n corresponds to the number of Flip-flops employed while the **binary coefficients** (either 0 or 1) to the taps.
  The LFSR can generate **at maximum** 2^n - 1 pseudo-random numbers if it is **primitive**, otherwise it will generate less.
  It is therefore important to choose the most suitable n according to the number of positions to reach.
- LFSR needs an initialization state. **0 and 1 are forbidden initial states** when employing XOR and XNOR gates respectively, since the state will be frozen;
- In the optimal case of the largest set of distinct vectors generated (2^n - 1), the LFSR will loop again by proposing the **same sequence in the same order** as the loop just expired.

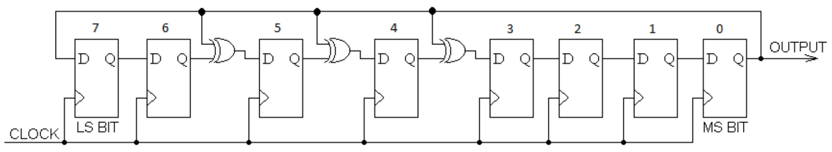


Figure 7.4. 8-bit, maximum length Fibonacci LFSR. Source: https://www.oocities.org/siliconvalley/screen/2257/vhdl/lfsr/lfsr.html

For the present case where $H_{img,inf} = 224$, the LFSR is designed with n = 8 stages, whose primitive polynomial reads.

$$x^8 + x^6 + x^5 + x^4 + 1 = 0 \tag{7.3}$$

As a matter of facts, coefficients at monomials x^6, x^5 and x^4 generates taps, i.e. XOR gates, at positions 6, 5 and 4 respectively.

The frequency at which the LFSR changes its state depends on the next frame captured and its handled by a counter that updates when the last image pixel of the current frame is reached.

Verification and post-synthesis results

The standalone module before integration into the RTL design is **functionally verified** via a **test vectors based testbench**. An image is digitally ablated and each pixel (R,G,B) tuple is saved accordingly along with the expected result after pixel ablation.
This ensemble of binary test vectors is used to feed the module and compare its results with the **expected ground truth** for the whole set of image pixels, providing a strong verification method.

After integration in the existing `RTL` design, Lattice Diamond tool is employed to perform `Synthesis`, `Mapping` and `Place&Route` processes under the same resource constraint conditions of the un-defended design. They are summarized in table 7.3:

Defense @HW	**Original design**	**Design with ablation**	**Available**
`Number of EBRs`	186	192	208
`Number of Registers`	28157	28220	84735
`Number of LUTs`	49072	49327	83640

Table 7.3. Resource usage, un-defended and defended design

Post-synthesis results show a negligible increase when the ablated module is activated. `EBR` number is affected by the **incremented width memory size associated to the line buffer**, while registers and `LUT` come from the `LFSR` stage.

7.3 Summary

This chapter outlined the constrains introduced when exporting the defense crafted in chapter 6 towards real-world adversarial attacks for **edge applications** in the form of a pre-processing module.
In general, the module is **affordable from the resource usage viewpoint**, and does not seem to have heavy influence on the overall process speed.
The **other major point of interest** of the ablation procedure seems to be the **introduction of timing constraints** associated to frame acquisition, punctuated by the camera sensor capability. The two alternatives mentioned in the text treat the problem by either **applying the defense once on each frame**, which is quite a too much simplified solution and rather un-realistic, or **exploiting $\geq$ 1 ablations per inference** by calibrating acquisition and detection time to follow de-randomized smoothing foundations more tightly.

Conclusions

This work had the purpose of analyzing the two complementary processes of Adversarial Attacks and Defenses in the framework of Object Detection for Computer Vision, either from theoretical and practical point of view, addressing **real-world based** attacks.

Hence, **Adversarial patch attacks** targeting **misdetection** have been addressed against three network families: `YOLO`, `SSD` and `MTCNN`, with an in-depth evaluation through Precision-Recall curve **`PR`**, mean Average Precision **`mAP`** and Patch Success Rate **`SR`**.
In general, the perturbation shows to be **quite effective** against **white-box** attacks (full-model knowledge by the attacker), reproducing already assessed results in the literature or extending them to novel architectures. **Networks internal structure** seemed to play a crucial role in order to quantify the entity of the damage caused by the perturbation.

Adversarial defense against the aforementioned kind of attacks has been chosen among the **randomized smoothing** category.
In particular, de-randomized smoothing by **structured ablation** is extended towards the field of object detection with the whole set of consequences this research field introduces with respect to pure classification.
Having real-world and **real-time** applications in mind, the process has been extended towards an existing object detection design targeting a **low-power `FPGA`** in the framework of *edge AI* applications.
The overall results show that, at software level, the defense is **able to reliably strengthen network performances under the attacks** described in the first part of the work.

As a **future outlook**, some trade-offs still need to be introduced when assessing the same approach at **hardware level**. First of all, this means selecting a resource-friendly NN model that would fit into the FPGA (or other platforms such as Microcontrollers) in terms of resource usage. This is not strictly necessary in the framework of the adversarial defense itself, rather for what concerns low-power deployments for edge devices. The sought after model can belong to the `MobileNet` family along with `SSDLite`, being its size dramatically lower than the network tested in this document (`YOLOv3`). Having this fixed, the major issue related to the Adversarial ML domain is the **real-time capability**, which is bounded by the video-stream flow steadily going from the camera sensor towards the edge device.
Eventually, future outlooks on a network deployed on an edge device in the context of the developed Adversarial Defense for the Object Detection research field in Computer Vision would involve:

- **performing tests at varying position distribution**, in order to increase the defense robustness and effectiveness by targeting a selected subset of positions. This feature is heavily dataset-dependent as previously mentioned;

- **selecting an optimal number of ablations between two consecutive captured frames**. This feature is bounded by the maximum number of inferences that could be performed between two discrete frames, that depends either on the camera frame rate and the NN model inference speed.

in order to completely match real-world based constraints.

Appendix A

Optimization methods

Optimization methods dictates the `speed` and the `quality` of Loss function convergence towards its minimum. This appendix aims at briefly describing two benchmark optimization methods that have rapidly risen above the others: `ADAM` and **`SGD` with momentum**.

SGD with classical and Nesterov momentum

Gradient descend optimization method follows the expression in eq. 1.4, which encodes the **entity of the step** to be taken, whose direction is dictated by the gradient itself.
Gradient descend performs operations over the whole input set, thus assuring loss decrease after each step. This can however slow down the process, mostly if the learning rate assumes small values.
Stochastic or Mini-Batch gradient descend allows to perform the update step computation over an input batch rather than the whole dataset. As a drawback, loss decrease after each step is no more guaranteed because a fraction only of the data is considered.

Momentum is a technique employed to take into account **previous gradients** at each step (by a certain amount), rather than the contingent and actual one. The general update rule reads:

$$\begin{aligned} v^{t+1} &= \mu \cdot v^t - \alpha \cdot g \\ \theta^{t+1} &= \theta^t + v^{t+1} \end{aligned}$$

where v brings information of the past gradients (speed and direction of the update), ν controls the extent at which past history of updates should be considered (previous gradients fraction) and α, g are the learning rate and actual gradient loss respectively.
Nesterov acceleration works by the same update rule, yet it calculates gradients in a different way - fig. A.1(b) provides a schematic representation.

Classical momentum

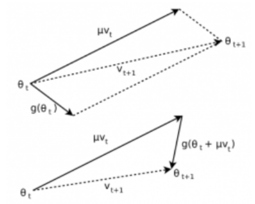
$$g = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \mathcal{L}(x^{(i)}, y^{(i)}, \theta_t)$$

Nesterov momentum

$$\tilde{\theta} = \theta_t + \alpha v_t$$

$$g_{NAG} = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \mathcal{L}(x^{(i)}, y^{(i)}, \tilde{\theta})$$

(a) Gradients computation



(b) Update rule scheme

Figure A.1. Update rule, classical vs. Nesterov momentum [77]

Fig. A.1(a) with classical momentum shows that the loss is computed with respect to the actual value of θ only, while Nesterov introduces the previous gradient information through the variable $\tilde{\theta}$, avoiding oscillations and guaranteeing control over unwanted high loss (for small steps, the two methods coincides).

SGD with Nesterov momentum is used for YOLOv3 training following the corresponding reference repository. In addition, it applies manual learning rate tuning by cosine decay as mentioned in chapter 6.

ADAM

ADAM [10] is an optimization algorithm that has been introduced as an extension of SGD [1]
Its major contribution is that it **automatically tunes the learning rate** instead of leaving it steady across the training stage as in traditional SGD methods.
Among the most noteworthy properties and advantages:

- It adds an exponentially decaying parameter that **tunes the stored past computed gradients** in analogy to what RMSProp does for the sum of squares of the gradients;
- It needs **little memory** requirements and it is **computationally efficient**;
- It fits well for **large data** and parameters problems;
- It is well suited for **noisy problems** to which apply the running mean (stochastic) approach;

ADAM is the optimization method employed for training the patch-like perturbation in the framework of adversarial attacks.

[1]In particular RMSProp algorithm, which is not addressed in this document and therefore not discussed.

Appendix B

Ensemble training

Ensemble training has been attempted in order to probe the **transferability** of adversarial attacks with several networks. Tables 4.2 and 4.3 show that **cross-evaluation** outside the domain of pure white-box attacks lead to **poor results** in terms of average precision drop.
The approach that has been followed by *Wu et al.* in [2] refers to **ensemble training**, i.e. adversary train by employing several networks instead of one only. Eq. 4.5 changes as follows:

$$\text{Find } \underset{P}{argmax}(\mathcal{L}) \text{ s.t. } \mathcal{F}_j(\underset{t\sim\mathcal{T}}{\mathbb{E}}[\mathcal{L}^{(j)}(f(\mathcal{A}(I,\mathcal{T}_{\theta_{aff}}(P))),P)]) \tag{B.1}$$

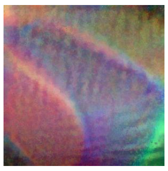
$\mathcal{F}$ can either indicate the worst-case maximum strategy or the average strategy [55] while j stands for jth detector. Henceforth:

$$\begin{aligned} \mathcal{F}_j &= \max_{\text{detector j}}(\dots) && \text{maximum strategy} \\ \mathcal{F}_j &= \frac{1}{N_{det}}\sum_{\text{detector j}}(\dots) && \text{average strategy} \end{aligned} \tag{B.2}$$

It is expected that after ensemble training `mAP` drop could transfer to each of the network employed at training time, as an extension of single white-box train.

Trained patches

Fig. B.1 shows two adversarial patches that have been trained following the procedure in sec. 4.1 and employing multiple models concurrently (`YOLOv2`, `YOLOv3` and `YOLOv4` respectively in this case).
The only difference in the overall process with respect to single-network training is indeed the additional operation that has to be performed on the compound set of prediction scores coming from each network, as in eq. B.2.


(a) Maximum strategy

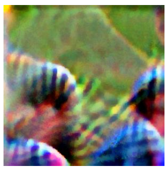
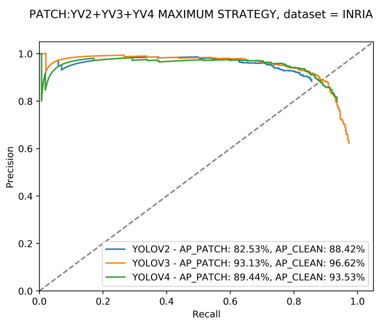

(b) Average strategy

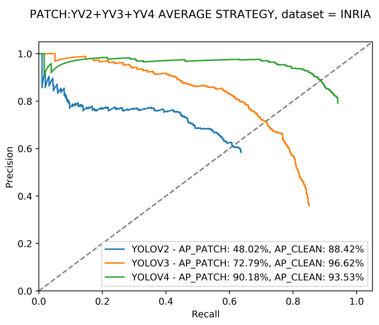
Figure B.1. Ensemble training patches, `YOLO` family, different Loss minimization strategies

Inference results

Fig. B.2 shows PR curves to assess ensemble training effects with either maximum (fig. B.2(a)) and average (fig. B.2(b)) strategies when training the adversary with three networks: `YOLOv2`, `YOLOv3` and `YOLOv4` on the `INRIA` dataset.
As related works, a similar attempt on the `COCO` dataset has been performed in [2] with an equivalent methodology, even though **summation over detectors** has been chosen as the ensemble operation.



(a) Maximum strategy



(b) Average strategy

Figure B.2. Ensemble training, `YOLO` family, PR curve evaluation on `INRIA` dataset

The average procedure strategy seems to be the most effective in terms of adversarial capability. Nevertheless, it shows that the same perturbation, which should encode information over the three network's weights, performs differently when tested against each single model, as if its inner architecture still plays a **key role** on adversarial performances at inference time.

Bibliography

[1] Simen Thys, Wiebe Van Ranst, and Toon Goedemé. “Fooling automated surveillance cameras: adversarial patches to attack person detection”. In: *CoRR* abs/1904.08653 (2019). arXiv: 1904.08653. URL: http://arxiv.org/abs/1904.08653.

[2] Zuxuan Wu et al. “Making an Invisibility Cloak: Real World Adversarial Attacks on Object Detectors”. In: *ArXiv* abs/1910.14667 (2019).

[3] A. Levine and S. Feizi. “(De)Randomized Smoothing for Certifiable Defense against Patch Attacks”. In: *ArXiv* abs/2002.10733 (2020).

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016. ISBN: 0262035618.

[5] A. Rosebrock. *Deep Learning for Computer Vision with Python: Starter Bundle*. PyImageSearch, 2017. URL: https://books.google.ch/books?id=9Ul-tgEACAAJ.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[7] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning Representations by Back-Propagating Errors”. In: *Neurocomputing: Foundations of Research*. Cambridge, MA, USA: MIT Press, 1988, 696–699. ISBN: 0262010976.

[8] *Understanding Backpropagation Algorithm*. 2019. URL: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd.

[9] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. second. New York, NY, USA: Springer, 2006.

[10] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).

[11] C. Zhang et al. “Understanding deep learning requires rethinking generalization”. In: *ArXiv* abs/1611.03530 (2017).

[12] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: *ArXiv* abs/2004.10934 (2020).

[13] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[14] *A Comprehensive Introduction to Different Types of Convolutions in Deep Learning*. 2019. URL: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215.

[15] Andrew G. Howard et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[16] *THE MNIST DATABASE of handwritten digits.* URL: http://yann.lecun.com/exdb/mnist/.

[17] *The CIFAR-10 dataset.* URL: https://www.cs.toronto.edu/~kriz/cifar.html.

[18] J. Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition.* 2009, pp. 248–255.

[19] R. Girshick et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition.* 2014, pp. 580–587.

[20] C. Szegedy et al. “Going deeper with convolutions”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2015, pp. 1–9.

[21] K. He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2016, pp. 770–778.

[22] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[23] Joseph Redmon and Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[24] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[25] Forrest N. Iandola et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size”. In: *CoRR* abs/1602.07360 (2016). arXiv: 1602.07360. URL: http://arxiv.org/abs/1602.07360.

[26] Mark Sandler et al. “Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation”. In: *CoRR* abs/1801.04381 (2018). arXiv: 1801.04381. URL: http://arxiv.org/abs/1801.04381.

[27] Andrew Howard et al. “Searching for MobileNetV3”. In: *CoRR* abs/1905.02244 (2019). arXiv: 1905.02244. URL: http://arxiv.org/abs/1905.02244.

[28] Tsung-Yi Lin et al. “Feature Pyramid Networks for Object Detection”. In: *CoRR* abs/1612.03144 (2016). arXiv: 1612.03144. URL: http://arxiv.org/abs/1612.03144.

[29] Pierre Sermanet et al. “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks”. In: *International Conference on Learning Representations (ICLR) (Banff)* (Dec. 2013).

[30] A. Rosebrock. *Deep Learning for Computer Vision with Python: ImageNet Bundle.* Deep learning for computer vision with Python. PyImageSearch, 2017. ISBN: 9781722487867. URL: https://books.google.ch/books?id=B3sBvgEACAAJ.

[31] R. Girshick. “Fast R-CNN”. In: *2015 IEEE International Conference on Computer Vision (ICCV).* 2015, pp. 1440–1448.

[32] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *Advances in Neural Information Processing Systems 28.* Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 91–99. URL: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf.

[33] Jasper Uijlings et al. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104 (Sept. 2013), pp. 154–171. DOI: 10.1007/s11263-013-0620-5.

[34] M. Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *International Journal of Computer Vision* 111.1 (Jan. 2015), pp. 98–136.

[35] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *ArXiv* abs/1405.0312 (2014).

[36] Chien-Yao Wang et al. "CSPNet: A New Backbone that can Enhance Learning Capability of CNN". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2020), pp. 1571–1580.

[37] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[38] Bichen Wu et al. "SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving". In: *CoRR* abs/1612.01051 (2016). arXiv: 1612.01051. URL: http://arxiv.org/abs/1612.01051.

[39] Zhaohui Zheng et al. "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression". In: *AAAI*. 2020.

[40] *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*. 2018. URL: https://gombru.github.io/2018/05/23/cross_entropy_loss/.

[41] Alina Kuznetsova et al. "The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale". In: *CoRR* abs/1811.00982 (2018). arXiv: 1811.00982. URL: http://arxiv.org/abs/1811.00982.

[42] Nicholas et al. "On Evaluating Adversarial Robustness". In: *ArXiv* abs/1902.06705 (2019).

[43] Mahmood Sharif et al. "Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition". In: Oct. 2016, pp. 1528–1540. DOI: 10.1145/2976749.2978392.

[44] Tom B. Brown et al. "Adversarial Patch". In: *ArXiv* abs/1712.09665 (2017).

[45] Shang-Tse Chen et al. "Robust Physical Adversarial Attack on Faster R-CNN Object Detector". In: *ECML/PKDD*. 2018.

[46] Christian Szegedy et al. "Intriguing properties of neural networks". In: *CoRR* abs/1312.6199 (2014).

[47] Nicholas Carlini et al. "Ground-Truth Adversarial Examples". In: *ArXiv* abs/1709.10207 (2017).

[48] Nicholas Carlini and D. Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *2017 IEEE Symposium on Security and Privacy (SP)* (2017), pp. 39–57.

[49] A. Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *ArXiv* abs/1706.06083 (2018).

[50] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *CoRR* abs/1412.6572 (2015).

[51] A. Kurakin, Ian J. Goodfellow, and S. Bengio. "Adversarial examples in the physical world". In: *ArXiv* abs/1607.02533 (2017).

[52] Francesco Croce and Matthias Hein. "Sparse and Imperceivable Adversarial Attacks". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 4723–4731.

[53] Anish Athalye et al. “Synthesizing Robust Adversarial Examples”. In: *ArXiv* abs/1707.07397 (2018).

[54] Kevin Eykholt et al. “Physical Adversarial Examples for Object Detectors”. In: *ArXiv* abs/1807.07769 (2018).

[55] Kaidi Xu et al. “Adversarial T-shirt! Evading Person Detectors in A Physical World.” In: *arXiv: Computer Vision and Pattern Recognition* (2019).

[56] Mark Lee and J. Zico Kolter. “On Physical Adversarial Patches for Object Detection”. In: *ArXiv* abs/1906.11897 (2019).

[57] Xin Liu et al. “DPATCH: An Adversarial Patch Attack on Object Detectors”. In: *arXiv: Computer Vision and Pattern Recognition* (2019).

[58] Edgar Kaziakhmedov et al. “Real-world Attack on MTCNN Face Detection System”. In: *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)* (2019), pp. 0422–0427.

[59] A. Braunegg et al. “APRICOT: A Dataset of Physical Adversarial Attacks on Object Detection”. In: *ArXiv* abs/1912.08166 (2019).

[60] Max Jaderberg et al. “Spatial Transformer Networks”. In: *ArXiv* abs/1506.02025 (2015).

[61] F. L. Bookstein. “Principal Warps: Thin-Plate Splines and the Decomposition of Deformations”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 11.6 (June 1989), 567–585. ISSN: 0162-8828. DOI: 10.1109/34.24792. URL: https://doi.org/10.1109/34.24792.

[62] Aravindh Mahendran and A. Vedaldi. “Understanding deep image representations by inverting them”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 5188–5196.

[63] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008. ISBN: 978-0-521-86571-5. URL: http://nlp.stanford.edu/IR-book/information-retrieval-book.html.

[64] *Breaking Down Mean Average Precision (mAP)*. 2019. URL: https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52#1a59.

[65] Kaipeng Zhang et al. “Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks”. In: *CoRR* abs/1604.02878 (2016). arXiv: 1604.02878. URL: http://arxiv.org/abs/1604.02878.

[66] Kui Ren et al. “Adversarial Attacks and Defenses in Deep Learning”. In: *Engineering* 6.3 (2020), pp. 346 –360. ISSN: 2095-8099. DOI: https://doi.org/10.1016/j.eng.2019.12.012. URL: http://www.sciencedirect.com/science/article/pii/S209580991930503X.

[67] A. Levine and S. Feizi. “Robustness Certificates for Sparse Adversarial Attacks by Randomized Ablation”. In: *AAAI*. 2020.

[68] Ping-Yeh Chiang et al. “Certified Defenses for Adversarial Patches”. In: *ArXiv* abs/2003.06693 (2020).

[69] Jeremy M. Cohen, Elan Rosenfeld, and J. Zico Kolter. “Certified Adversarial Robustness via Randomized Smoothing”. In: *ICML*. 2019.

[70] Mathias Lécuyer et al. “Certified Robustness to Adversarial Examples with Differential Privacy”. In: *2019 IEEE Symposium on Security and Privacy (SP)* (2019), pp. 656–672.

[71] Yuchen Zhang and Percy Liang. “Defending against Whitebox Adversarial Attacks via Randomized Discretization”. In: *AISTATS*. 2019.

[72] Michael McCoyd et al. “Minority Reports Defense: Defending Against Adversarial Patches”. In: *ArXiv* abs/2004.13799 (2020).

[73] *Data Augmentation in YOLOv4*. URL: https://towardsdatascience.com/data-augmentation-in-yolov4-c16bd22b2617.

[74] Cong Hao et al. "New Design Methodologies and Future Trend for Edge AI". 2020.

[75] J. Chen and X. Ran. "Deep Learning With Edge Computing: A Review". In: *Proceedings of the IEEE* 107.8 (2019), pp. 1655–1674.

[76] *Lattice Embedded Vision Development Kit.*

[77] *Stochastic Gradient Descent with momentum*. 2017. URL: https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d.