Adversarial Patches Attack Transparency in Industrial Applications

ACHTATOU Sean
Email: sean.achtatou.001@student.uni.lu

A paper presented for the Thesis of Master in Information and Computer Science

Master in Information and Computer Science
University of Luxembourg
Luxembourg
Wednesday 4th January, 2023

Contents

***Abstract*—With the progress in autonomous vehicles technology, the growth in the researches around attacks and defenses towards these vehicles have been observed. Self-driving cars being based on Artificial Intelligence artifacts, these autonomous vehicles are prone to have their operations altered using multitudes of attacks to wrongfully miss-operate them in a specific desired state. In our research, we focused on adversarial patches attacks, as they are being more prone to be used in industrial applications and simple to be deployed. Indeed, among existing adversarial patches attacks researches, many are lacking the idea of pure applications of the attack in the real environment, mainly focusing on the theoretical part of the attack (i.e. White-Box attack). In the Black-Box attacks, researches with adversarial patches have been as well presented, however they do not constantly focus on the target object to be attacked, and based on the content of the attack efficiency. In our research, we successfully combine the idea of focusing on the target object to be attacked, in order to be applicable in a real environment, and the limitations of the visibility of the patch based on the color domain of this same target object, that is based on visibility towards the human factor.**

1. INTRODUCTION

Nowadays, the industry of autonomous driving cars is in perpetual expansion, and progressively self-driving vehicles are being produced, improved and used on live roads participating in the traffic[[1]. In order to acquire this efficiency of operation, self-driving cars are based on the integration of Artificial Intelligence models, granting the capability for a vehicle to be aware of its state and traffic at any moment, by analyzing the environment with sensors (*visual and waves*). Among these sensors, we discover the usage of a specific camera, responsible for the visual related tasks, such as the recognition of cars, pedestrians, and obviously the traffic signs on the roads.

Acknowledged, we are aware that the objective of any Artificial Intelligence model θ is to operate on the classification tasks it has been defined for, minimizing the loss L of a set n of data x and their corresponding labels y, and therefore is highly prone to be exploited and alternated.

$$min \sum_{i=0}^{n} L(x_i, y_i; \theta)$$

Indeed, numbers of attacks on Artificial Intelligence models have already been produced, either based on the Black-Box and/or White-Box attacks (*terms explained further in the research*. We may enumerate few of the well-known attacks : *L-BGFS*, *FGSM*(gradient attack), *BIM* (iterative gradient attack), *JSMA*, *Carlini-Wagner* (CW).

These attacks are precisely attacking the whole sample of an Artificial Intelligence model. Nevertheless, there exists attacks based on only attacking a restricted region of the sample,

- Adversarial patches
- GAN-Based

Among the entire set of possible attacks, the objective is relatively similar, we want to maximize a loss L with the use of an adversarial perturbation δ on a sample x on model θ, in order to produce an alternated desired output y.

$$max\ L(x+\delta, y; \theta),\ ||\delta|| < \epsilon$$
$$or$$
$$min\ ||\delta||_p,\ L(x+\delta, y; \theta) > \epsilon$$

$$\Rightarrow max\ L(x+\delta, y; \theta) + c * ||\delta||_p$$

$$max_\delta\ l(h_\theta(x+\delta), y),\ with \delta = \{\delta \mid ||\delta||_\infty \leq \epsilon\}$$

For this research, we have focused on the usage of the adversarial patches, as several projects have already worked on attacks based on it[[14]]. Indeed, adversarial patches attacks have been applied by multitudes of researches papers, highlighting variability in their usability, implementing new and/or improved types of attacks algorithms, either based on the Black-box and/or White-box attacks.
Nevertheless, we can observe that these researches focused principally on the efficiency of the adversarial patches, and fewer on the reproducibility and usability of the attack proposed in a real environment. Indeed, we may notice that some of these researches do not apply specific restriction to the adversarial patches, firstly related to the position (*the patch is not focused on a specific position - few exceptions where the patch is in a corner of the image and/or is the frame*) and secondly the adversarial patches content (*the factor about the percentage of visibility of the patch is neither exploited and/or taken into consideration*) where it is highly noticeable by a human factor external to the Artificial Intelligence model. Due to these reasons, most of these attacks lays into the theoretical section, as the lack of non-reproducibility possibility in a real environment is present(*some exceptions can be observed, and will be enumerated further*).

In our research, we have therefore contributed toward this direction by providing an adversarial patch capable to fulfill the limitations of these past researches. Being usable in a real environment, as at the same time limiting the visibility of the adversarial patch from the human factor (*experimenting on the color domain related to the targeted object*), our work focused on attacking the traffic signs classification via the manipulation of the

Artificial Intelligence model classifier, based on the images provided by the camera sensors of an autonomous vehicle. To achieve this objective, we used a multi-objective optimization framework named **Pymoo**, granting the manipulation of evolutionary algorithms that benefited us to produce adversarial patches.

Consequently, we defined the next three researches questions related to the patches capabilities to be used in a real environment.

R1 - *How to restrict the visibility of the adversarial patches from the human factor?*
We restricted the modification of the adversarial patch based on the channels pixels values of the attacked image. The three channels pixels, the *mean* and *values counting*, allowed to determine a threshold around a base RGB, to set a maximum and minimum for each channel of the adversarial patch. The visibility metric granted the possibility to inquiry for a human factor the noticeability difference between the adversarial patch color produced and the image attacked. If the threshold is not respected, the difference of color between the adversarial patch and the target is relatively significant.

R2 - *Is producing adversarial patches of different forms and size delivering similar attack efficiency?*
We experimented the attack with different forms of adversarial patches: *circle*, *square* and *triangle*. Concurrently, we defined multiples sizes for a patch, based on a percentage of the attacked image size. These experiments allowed the adversarial patch appliance to become more versatile and usable in different situations with traffic signs of different forms. We have firstly observed, indeed that the size of the patch mattered, considered that fewer the number of pixels to change in image, less is the power of attack efficiency. Similarly, with alternative patch form, the amount of pixels participating in the attack becomes variable; based on the square patch, we restricted the patch to obtain a form corresponding *circle* and *triangle*. Nevertheless, few exceptions were observed, usually due to the type of traffic sign we were experimenting with.

R3 - *Can we apply Pymoo to produce adversarial patches fulfilling the restriction in visibility and attack position, but still being efficient?*
We benefited from the use of the framework **Pymoo**, formerly implemented to solve multi-optimization problems based on evolutionary algorithms. For our research, we retrieved and applied the Pymoo genetic algorithm process, and adapted it to produce adversarial patches based on a given image and attacked model. The optimization of each adversarial patch is based on three parameters: the content of the adversarial patch (*the pixels values*), the position (*x,y*) and the angle (*z*) of the patch on the attacked image. We managed to design and produce a rigorous adversarial patches generator, based on a given attacked image and model, restricting the visibility of the patch from a human factor.

1.1. Domains

The domains of this paper lies into two important and large fields of computer science, namely *Artificial Intelligence* and *Security*.

Artificial Intelligence is involved, as throughout this research paper, we apply the notion of Neural Networks in order to develop a model using computer-vision to be capable to detect road signs. We learn how develop an efficient deep learning model, and provide details on its structure and capabilities to detect signs. We will carry on, with an attack towards the model, applying adversarial methods in order to make the model miss-classify an original input, given a constructed adversarial sample.

The **Security** is involved, as the possibility of attacking such model represents a real danger for the community of self-driving vehicles. As computer-vision is an important element on which relies self-driving algorithms to analyze its environment - such as a human does with its eyes. We must be able to defend the model against such attack, enumerated earlier, to happen, and will consequently provide a defense applied to the model against it.

2. STATE OF THE ART

The domain of Computer Science is vast and progressively in expansion. With the progression of the actual technology, the apparition of new solutions for many problems emanate and is improving the worldwide population life quality. Our domain of expertise evolving around Artificial Intelligence and Security, a proper comprehension of this branch of science in the work progression realized up to now, is therefore required. As Artificial Intelligence is progressively taking more space into our everyday life, solving problems regarding the introduction of automated tasks of classification and prediction in sensible area, the concerns of their efficiency and security for the being have been introduced and raised discussion.
Indeed, the domain of Artificial Intelligence have been subject to the opening of multitudes numbers of attacks during these last decades, with the goal of miss-leading the model behavior. In response to these threats, expert have designed defenses mechanisms to be set in order to mitigate nay deny the attack to occur.

2.1. Refreshment in Artificial Intelligence

Nowadays, the word *Artificial Intelligence* is commonly used by the population, and unfortunately rarely understood. Neither in its usability and/or its real operation, mainly due to the lack of knowledge in its domain and complexity to be discerned.
In the last years, definition about the question of what Artificial Intelligence is, have been raised multiple times and diverge from one being to another. From our knowledge, we have managed to define a global and easily understandable definition that can be described as:

"The domain of computer science related to the automation of tasks, involving the implementation of a program , based on the use of complex mathematical algorithms, in order to achieve a defined objective such as classification, prediction or regression."

Artificial Intelligence mechanisms are applied for many purposes, improving the quality of our life, and are composed of subgroups. These last are specified based on the algorithms and/or methods application based on the difference of structure and the tasks to be accomplished. There exists seven different types of Artificial Intelligence that can be enumerated in two parts based on the tasks objective.

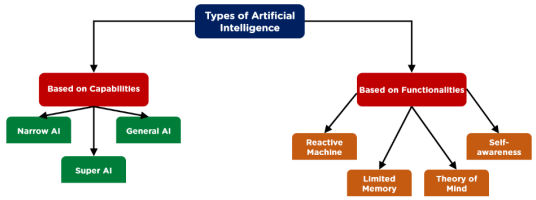


Figure 1. The 7 types of AI (source)

2.1.1. The Seven Pier in Artificial Intelligence.

Artificial Intelligence can be divided into three distinct categories based on the tasks capabilities (TYPE-1):

- *Narrow AI*
- *Strong AI*
- *Super AI*

Narrow AI - This is the Artificial Intelligence systems that we rub shoulders with, and is defined as weak(narrow) AI. As stated earlier, these systems defines the ones capable to perform a narrow task for a single problem such as facial recognition or driving cars, and can not perform outside these limitations, as only trained for this specific tasks outperforming the human only for this particular problem. These systems are limited.

Strong AI - Nonetheless, the ideal objective would be to achieve the second type of intelligence, which is a strong(general) AI. These systems are less limited, and would be capable to achieve multiples different tasks, and performing, thinking and acting like a human would do. Nowadays, there does not exist such system under general AI yet, nevertheless worldwide researchers are now heavily focused on their development with the hope to create one.

Super AI As final stage, the last type of intelligence would be the development of super AI. In this type of systems, their ability would be the capability to outperforms the humans in any tasks, solving problems, communicating, and learning better than any of the population. Even if this category of AI is still defined as a concept, the development of such intelligence is on the way.

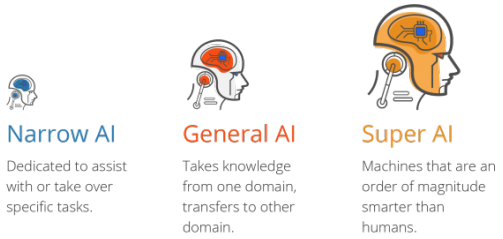


Figure 2. Narrow AI vs Strong AI vs Super AI (source)

Artificial Intelligence can be divided into three distinct categories based on their tasks functionality. (TYPE-2):

- *Reactive Machines*
- *Limited Memory*
- *Theory of Mind*
- *Self-Awareness*

Reactive Machines - In reactive machines, which is the most basic types of AI, the machines are based on action-reaction and are very limited. They do not store any memory of the past actions and/or experiences, and simply react to the given input on the present time without the gained experiences from the past. Therefore, they do not have the ability to learn and/or to improve. An example of such machine is the well known Deep Blue chess-beater by IBM.

Limited-Memory - In addition to reactive machines, the limited-memory machines does store data for a limited time and can be used for further actions. It does mean that the machine is capable to learn from the past actions and information obtained, and take decision based on it. An example of such machine would be the Image Recognition with Neural Networks.

Theory of Mind - Far from the objective of the two lasts machines defined earlier, in the Theory of Mind the AI would be capable to understand, discern and interact with the humans emotions in order to become sociable. This type of AI would help the humans for the interaction between AI-HUMAN, as nowadays communication with an AI is quite limited.

Self-Awareness - The last type of machine correlate with the super AI type defined earlier and would be the final stage of the Artificial Intelligence objective which is the self-awareness of the machine. The machine would be super intelligent, gain a conscious and be smarter than the humans, allowing the possibility to solve amount of problems and issues unsolvable by the humans so far.

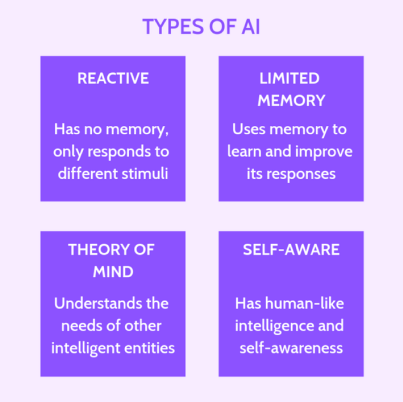


Figure 3. The 4 types of AI functionalities (source)

NOTE There does not exist any AI systems belonging to Theory of Mind and Self-Awareness. So far, the human approach is based on designing systems that think and act like humans, therefore mimicking the behavior. But an ideal approach would be systems that think and act rationally by themselves.

2.1.2. Machine Learning and Deep Learning.

We have defined the different categories of Artificial Intelligence based on their tasks capabilities and functionalities. However, the domain of Artificial Intelligence is itself composed of sub-fields, defining the type of approaches and/or algorithm which is used to find a solution to a specific problem.
Indeed, we can enumerate the two following sub-fields:

- *Machine Learning*
- *Deep Learning*

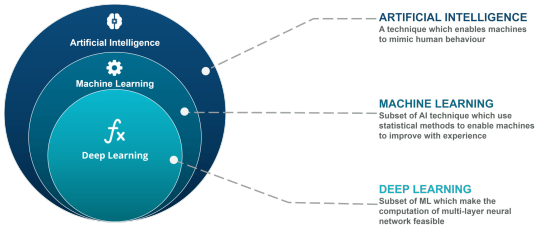


Figure 4. Machine Learning vs Deep Learning in Artificial Intelligence (source)

These two sub-fields do not differ only due to the type of algorithms applied, but as well based on the objective that the Artificial Intelligence must achieve.

Machine Learning - In Artificial Intelligence, a Machine Learning system defines the learning of a machine model to improve itself by using structured data and defining the features. Becoming more accurate and capable to predict outcome using the trained model without being explicitly programmed to do so. Machine Learning is itself composed of three different types based on how the data is used to train the model.

- *Supervised*
- *Unsupervised*
- *Reinforcement*

Supervised Learning - In supervised learning, the model is trained using the label of the data X given (i.e [X,"Woman"]). The model will learn to predict future value based on the knowledge obtained from the data. This method is usually applied to solve problem of classification (predict a discrete value) and regression problems (predict on continuous data). Example of algorithms based on this type of learning are *linear regression, logistic regression, Support Vector Machine (SVM), K-Nearest Neighbour.*

Unsupervised Learning - In unsupervised learning, the model is not provided the labels of the given data, and need to seek for meaningful connection and patterns in the data to find features, and manage to cluster the data. This method is usually applied to solve problem of association and clustering. Examples of algorithms based on this type of learning are $k-means$ and $NeuralNetworks$.

Reinforcement Learning - In reinforcement learning, the model learn to react to his environment with an agent, a state, a set of rules and rewards. The agent will explore his environment and based on his actions, a positive/negative reward is returned to the new state, and the agent will learn to optimize his choice of path based on the reward obtained. This method is usually applied to solve problem of exploitation and exploration. Examples of algorithms based on this type of learning are $Q-learning$ or $SARSA$.

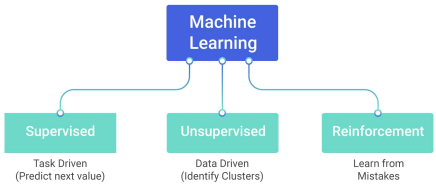


Figure 5. Supervised Learning vs Unsupervised Learning vs Reinforcement Learning (source)

Deep Learning - The Deep Learning system, is a subset of Machine Learning, attempting to mimic the human brain, and defined as a Neural Network composed of more than three set of layers (we will detail its structure and process in the next section). Compared to Machine Learning, Deep Learning algorithms does not need to work with structured data in order to make prediction, but can efficiently achieve their goal by determining features and dependency in the data. The learning phase of the Neural Network is then based on the use of gradient descent and back propagation to adjust the weights of the layers and improve the accuracy of prediction. Example of a deep learning algorithm is the convolutional neural networks.

2.2. Attacks against Artificial Intelligence

When training a model, the main objective to achieve is to minimize the loss of the results of the prediction of the training data x through the model h, with their label y. In the case of a Neural Networks, the model uses gradient descent in order to minimize this loss, applying multiples batches B and epochs E (usually the cross entropy loss is applied) and adjusts the weights and bias of the model accordingly.

$$\min_\theta \; l(h_\theta(x), y)$$

$$\min_\theta \frac{1}{m} \sum_{i=1}^{m} l(h_\theta(x_i), y_i)$$

$$\mathrm{l}(\mathrm{h}_\theta(x), y) = log(\sum_{j=1}^{k} exp(h_\theta(x)_j)) - h_\theta(x)_y$$

$$\theta = \theta - \frac{\alpha}{|B|} \sum_{i \in B} \nabla_\theta \; l(h_\theta(x_i), y_i)$$

Nevertheless, for several decades, AI models have been subject to the introduction of adversarial attacks, which objective is the miss-classification of the model. The idea is to apply the inverse technique originally used in order to train the model, and trying to maximize the loss of the model to miss-lead its behavior. The objective is to obtain an adversarial sample close to the original data based on a specific distance metric, usually to make it imperceptible to be detected by the human eye or understood. However, enough modified for the model to output an unexpected result. Therefore, this perturbation must be comprised between a certain value. And this perturbation value can be defined by the Pareto frontier, which can be defined on the maximum confidence attack or the minimum distance attack.

$$min \; L(x+\delta, y; \theta), \; ||\delta|| < \epsilon$$
$$or$$
$$min \; ||\delta||_p, \; L(x+\delta, y; \theta) < \epsilon$$

$$\Rightarrow min \; L(x+\delta, y; \theta) + c * ||\delta||_p$$

$$max_\delta \; l(h_\theta(x+\delta), y), \; with \delta = \{\delta \mid ||\delta||_\infty \leq \epsilon\}$$

The adversarial attack method cited earlier is categorized as a randomized one (also defined as error-generic), as it only targets to maximize the loss of the perturbed data far from the original class. However, we can define more concise attacks such as targeted attacks (also defined as error-specific), which try to produce an adversarial sample, but tracking on minimizing it to a chosen class.

$$max_\delta \; (l(h_\theta(x+\delta), y) - l(h_\theta(x+\delta), y_{target}))$$
$$\Rightarrow max_\delta((h_\theta(x+\delta), y_{target}) - (h_\theta(x+\delta), y))$$

From these observations, we can easily learn that the main objective of these various attacks is to find a perturbation δ, which applied on an input, is affecting the prediction of the model. From this perturbation, the attacker is able to achieve a targeted objective based on the attack intention towards the model.

- Security Violation (integrity, availability, privacy)
- Attack Specificity
- Error Specificity

NOTE Before pursuing, we require to define some definition that are being applied during the progression of this paper.
J(θ,x,y) is the optimization loss (adversarial loss), and represent the difference between the label predicted by the model and the actual label that should be predicted.
$x^{'} : D(x, x^{'}) < n, f(x^{'}) \neq y$ is defined as an adversarial example, which prediction should be different from the original one, but the data should be as close as possible to the original one.

Finally, the distance metrics between the benign sample and the adversary sample is defined by the following norms L_1 (Manhattan distance),L_2 (Euclidian distance),L_∞.

$$L_1 : ||X||_1 = \sum_{n=1}^{X} |x_n|$$
$$L_2 : ||X||_2 = \sqrt{\sum_{n=1}^{X} |x_n|^2}$$
$$L_\infty : ||X||_\infty = max(|X|)$$

There exists multiples ways in order to attack an AI model. Different types of models, information and structure, require alternated strategies based on the adversary capabilities knowledge of the model structure and information. An attacker have the choice of opting for three strategies:

- ❍ *White-Box*
- ❍ *Gray-Box*
- ❍ *Black-Box*

Figure 6. White-Box vs Gray-Box vs Black-Box (source)

White-Box - In the white-box model, the adversary has access to the structure and the information of the model. From this knowledge, the attacker is then capable to study the internal structure to gain an advantage. One possibility is to differentiate the model (only if the model algorithm uses the gradient descent in its structure), allowing gradient-based attacks to occur.

Attacks (those attacks are specific to individual benign samples, and do not transfer through them, the perturbation is applied on all element of the sample):

- ❒ *L-BGFS*
  Minimizing a loss, by finding adversarial perturbations with minimum L_p norm.

$$x^{'} = c||x - x^{'}||_p + J(\theta, x^{'}, y^{'})$$
$$min_{x'} ||x - x^{'}||_p$$
$$with\ f(x^{'}) \neq y'$$

- ❒ *FGSM (gradient attack)*
  Performs a one-step update on the direction of the gradient of the adversarial loss, increasing the loss.

$$x^{'} = x + \epsilon * sign(\nabla_x J(\theta, x, y))$$
$$x^{'} = x - \epsilon * sign(\nabla_x J(\theta, x, y^{'}))$$

- ❒ *BIM (iterative gradient attack)*
  Identical to FGSM, except it runs an iterative optimizer for multiple iterations. It projects the adversarial example in the $\epsilon - L_\infty$ neighbor and valid range.

$$x^{'}_{(t+1)} = Proj\{x^{'}_t + \alpha * sign(\nabla_x J(\theta, x^{'}_t, y))\}$$

- ❒ *MI-FGSM (integrate momentum memory)*
  It additionally integrates a momentum memory in the iterative process.

$$x^{'}_{(t+1)} = Proj\{x^{'}_t + \alpha * sign(g_{t+1})\}$$
$$g_{(g+1)} = \omega * g_t + \frac{\nabla_x J(\theta, x^{'}_t, y)}{||\nabla_x J(\theta, x^{'}_t, y)||_1}$$

- ❒ *DAA (attack on space of probability measures, optimizing over the potential adversarial distribution)*
  Instead of generating adversarial example independently for each benign example, we can perform optimization over the potential adversarial distribution μ and benign data distribution $\pi(x)$.

$$max_\mu \int_\mu J(\theta, x^{'}_t, y) d\mu + KL(\mu(x^{'})||\pi(x))\, dx$$

- ❒ *JSMA (use small L_0 perturbations, compute the logit outputs before the softmax layer identifying how input affect the logit outputs)*
  The JSMA is perturbing the DNN with small L_0 perturbations, by computing the Jacobian matrix of the logic outputs $l(x)$ before the softmax layer. The principle is that the Jacobian matrix is identifying how the input x affect the logit outputs of different classes. It perturbs the element $x[y]$ with the highest value of $S(x, y')[y]$ to increase/decrease the logit outputs of the class.

$$\nabla l(x) = \frac{\partial l(x)}{\partial x} = \left[\frac{(\partial l_j(x)}{\partial x_i}\right]_{i \in 1,..,M_{in}; j \in 1,\ldots,M_{out}}$$

- ❒ DF (find minimum L_2 adversarial perturbations on differentiable and non-differentiable classifier)
- ❒ Carlini-Wagner (CW) (generate L_0,L_2,L norm measured adversarial examples)

Attacks (does not perturb all the element of the benign sample, but only a restricted region):

- ❒ Adversarial Patch (optimize the adversarial loss of a patch based on benign images, patch transformations and patch locations)
  Select a few neurons significantly influencing the network outputs, then the pixel values in the region of adversarial patch are initialized to make the neurons achieve the maximum; universality is achieved by optimizing over the whole samples; robustness to noise and transformation is achieved with expectation over transformation EoT method.
- ❒ GAN-based (generative adversarial samples)

Black-Box - On the other hand, the Black-box model is non-differential, as the adversary has no access to the model structure. However an attacker can query the model with several inputs, and obtain an output for each of them. From these pairs inputs/outputs the adversary have the possibility to analyze the behavior of the model and adjusts the input accordingly to obtain an efficient adversarial sample.

Attacks (does not perturb all the element of the benign sample, but only a restricted region):

- ❒ *Surrogate Classifier*
  Mimic the operation of the real classifier being targeted by approximating its structure and behavior, in order to apply the gradient-based method.
- ❒ *Genetic Algorithm*
  Use the genetic principle, population, reproduction and mutation, in order to minimize a fitness function representative of the objective.

Finally, there exist four big categories of attacks that can be enumerated based on how the attack have been applied by the adversary. Indeed, the attacker have the possibility to attack the model, before and/or after it has been constructed, and/or by being passive or active. Allowing to conduct the following:

- Evasion Attacks
- Poisoning Attacks
- Inference Attacks
- Extraction Attacks

Evasion Attacks - The Evasion attacks consist of manipulating the input data to evade a trained classifier at run time. The attacker is modifying the input in order for the model to miss-classify/miss-predict the data, and outputting a wrong value. We can have error-generic or error-specific evasion.

Poisoning Attacks - The Poisoning attacks consists of increasing the number of miss-classified samples at testing time by injecting poisoned samples in the training data. With this method, the model will miss-learn his optimal weights, and will have more difficulty nay will not be able to classify/predict future data correctly.

Inference Attacks - The Inference attacks consists of probing the model with different inputs, and analyzing the corresponding outputs. The objective is to manage to retrieve information about the data that have been used to train the model, which might contains sensible information.

Extraction Attacks - The Extraction attacks are similar to inference attacks in the idea. Except, instead of stealing sensible information from the train data used by the model, the adversary is stealing the information about the structure of this last. With this method, the attacker is capable to progressively construct a substitute model with the same behavior of prediction.

NOTE There exists an isolated attack named adaptive adversarial attack, which aims to not manually tune the hyper-parameters used during the production of adversarial samples. Consequently, adversarial samples require more time to be produced since it depends on individual tasks, and is then technically impossible to be used for real-time systems. Nevertheless, the adaptive attack allows to improve the speed attacking leveraging multi-task learning, and since the generation of adversarial examples is a multi-task optimization problem, it can be approached.

There exist some challenges to the attacks that have been enumerated earlier, especially when applied in practice in real-time models and environment due to the diversity of scenarios.

One of the first challenge, is the noise and transformations of the environment in which the model is evolving, this modification is likely to destroy the adversary perturbations. This issue can easily be countered by

applying the expectation over transformation EoT method. Indeed, EoT applies random noise and transformation on the input, then takes the average over all the gradients with respect to these modified inputs, therefore the perturbation of the environment data on the adversarial sample is not impacted on it.

The second challenge is more specific to the images/videos processed models, in which the pixels has to be perturbed in real-time scenarios, due to the fact the background can be different from one to another. Nevertheless, this issue can be countered by applying a patch. We can even use a mask/patch transformation in order to separate the background from the object to be attacked itself, and apply the adversarial perturbations in the restricted region.

2.3. Defenses against Artificial Intelligence

We have seen that numbered of attacks have been presented in order to perturb the operation of the artificial intelligence model. From the knowledge of these attacks, defenses techniques have been designed to mitigate, nay eliminate the attack capabilities enumerated earlier. Now, that we have the knowledge of how a model can be attacked, we should be able to provide a defense against it. One of the most known defense goals is named min-max/robust optimization, providing adversarial learning. We are using the adversarial examples in order to robust the model against attacks.

$\min_\theta \; R_{dv}(h_\theta, Dtrain) =$
$min_\theta \frac{1}{Dtrain} \sum_{(x,y)\in Dtrain} \; max_{\delta\in\Delta(x)} \; l(h_\theta(x+\delta), y)$

with, $\theta = \theta - \frac{\alpha}{|B|} \sum_{(x,y)\in Dtrain} \nabla_\theta \; max_{\delta\in\Delta(x)} \; l(h_\theta(x+\delta), y)$

As, $\delta^* = argmax_{\delta\in\nabla(x)} \; l(h_\theta(x+\delta), y)$
$\Rightarrow \nabla_\theta \; max_{\delta\in\nabla(x)} l(h_\theta(x+\delta), y) = \nabla_\theta \; l(h_\theta(x+\delta^*), y)$

These defenses techniques can be categorized in two parts, *heuristic* and *certificated defenses*.

Heuristic Defenses - The Heuristic defenses are performing well in specific attacks, however they do not hold any theoretical guarantee to be successful. Adversarial training is considered as such defense mechanism. More heuristic defenses are applying input transformations/denoising in order to limit the adversarial effects.

Certificated Defenses - The Certificated defenses does provide the guarantee that an attack will not be able to perturb the model over a given threshold of lowest accuracy for a defined class of adversarial attacks. In other words, the model is guaranteed to be efficient over an accuracy even under attacked.

Defenses (for white-box based attacks):

- ❒ *Distillation*
- ❒ *Rejection/Detection*
- ❒ *Robust Optimization by adversarial training*
  Improve the robustness of the model by training it with adversarial samples.
- ❒ *FGSM adversarial training* (vulnerable to iterative attacks)
- ❒ *PGD adversarial training* (resistance against L attacks)
- ❒ *Ensemble Adversarial training*
  Use multiples pre-trained models with adversarial samples.
- ❒ *Adversarial logit pairing*
  Include the cross-entropy between the benign samples and the adversarial samples in the training loss evaluation.
- ❒ *Randomization*
  Randomize the adversarial effect on the model; by resizing and padding of the input; by adding a noise layer; using feature pruning and activations drops.
- ❒ *Input and Features Denoising*
  - ➢ *Conventional input rectification*
    Squeeze the model into two other models, one with bit reduction and one with image blurring, then compare the three models and apply the majority rule.
  - ➢ *GAN-Based input cleansing*
    Generate a distribution of benign images in the training of the model. During the test phase, an adversarial input can be cleansed by searching for an image close inside the learned distribution, and feeding this benign sample to the classifier, instead of the adversarial sample.

Defenses(for black-box based attacks):

- ❒ *Robust Optimization by adversarial training*
  Improve the robustness of the model by training it with adversarial samples.
- ❒ *Generative adversarial training*
  Train the model on benign samples, and on adversarial examples generated with a generative algorithm.
- ❒ *Randomize training data collection*
- ❒ *Difficult reverse-engineered models with ensemble models*
- ❒ *Randomize classifier output*

NOTE Some defenses are based on the obfuscation of the gradient, making it nonexistent or non-deterministic due to add-on operations such as quantization or randomization. For defense with non-differentiable add-on, we can use differentiable functions to approximate them. For defense with non-deterministic operations, we can use EoT. For defense vanishing the gradient with loops, we can change the variable for the optimization loop to be transformed in a differentiable function.

Based on the time space, in which we are willing to apply a defense during the processes for creation of the AI model, we can define two categories of defenses, *reactive* and *proactive*.

Reactive Defense - The reactive defense is usually applied to counter past attacks which have already occurred, therefore the model have been subject to a successful attack. The process is based on detecting the new attacks, retraining the classifier against those attacks, and verify the efficiency of the classifier decision of the training data and the labels.

Proactive Defenses - The proactive defense aims to prevent future attacks to occurs. Instead of waiting for a successful attack to happen on the model, there is an iterative retraining of the classifier on simulated attacks, allowing the development of a robust optimization and rejection.

2.4. Convolutional Neural Networks

In this section, we are quickly detailing the structure of the Convolutional Neural Networks. As, along this research paper we require to work with road signs, we must be able to train a model capable to detect these signs on a given image and output the correct classes. We will observe that the Neural Network is an ideal choice, as we are interacting with image data. Precisely, we will train and introduce the Convolutional Neural Network [CNN] structure.

2.4.1. Structure of Neural Networks.

The Neural Network [NN] is a subset of Machine Learning, and is based on the use of neurons in order to simulate the functionality of the human brain. We usually define a Neural Network as a Deep Learning Artificial Intelligence, however it is only technically considered as such when the amount of layers is above the number of three. Indeed, a Neural Network is composed of multiples layers, themselves composed of multiples nodes, called the neurons. The structure of a Neural Network is separated in three distinct section.

- ❍ Input Layers
- ❍ Hidden Layers
- ❍ Output Layers

Input Layers - The Input layers are the entry point for the data to be fed to the model. Based on the size of the data, the number of neurons required in the input layer can variate. We can imagine this as a human, eating food with its mouth, this represents the entry point of the body. As example, with an image of $10x10$ pixels, the input would be each of the pixels representing the neurons. Exceptionally, and if the task requires it, multiples input layers can be required for the model. However, this is beyond our research in this paper.

Hidden Layers - The hidden layers represents the structure of the model. This is the area we design the operation of the system, and define how the model will behave based on the given input data. It is possible to add multiples different layers, each with/without different number of nodes.

Output Layers - The output layer represents the final output(s) obtained once the data have been processed through the model, and how it has been classified. The output layer can be composed of one or multiples nodes based on the tasks, such as a binary or multi-class classification.

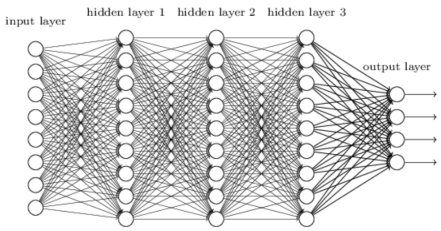


Figure 7. Structure of a Neural Network (source)

When feed-forwarding data in the model, each neurons n of each layer l will obtain a value.
This value is calculated from the sum of the values of the m neurons from the preceding layer, and the weights w of the model between each pair of neurons, plus a bias b. This process is called *fast forwarding*.
The weights are values calculated and modified during the training phase of the model, and are fixed once the training is finished. These weights determine how the model will behave and how the activation of certain neurons will influence the final output.

$$n_l^i = \sum_{j=0}^{m} n_{l-1}^j * w_{n_l^i, n_{l-1}^j} + b_l$$

As said earlier, the values of the weights are calculated during the training phased of the model. They are modified via the use of a *loss function* and the *back propagation* process.
The loss function is used in order to calculate the value difference of the output that have been obtained by the model given the input data, compared to the required target output; and is effectuated for each of the nodes of the model. And there exists different types of loss functions based on the output types, such as binary or multi classification.

$$Loss = \sum \frac{1}{2}(TARGET - output)^2$$

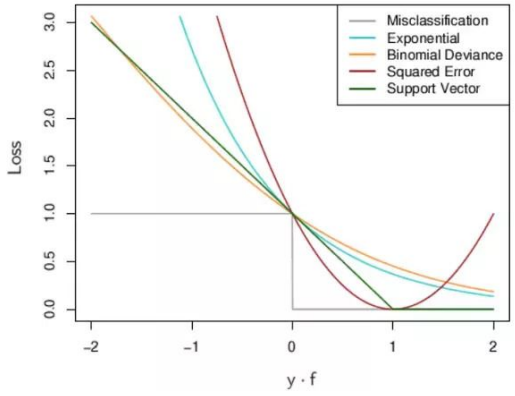


Figure 8. Graph of some loss functions (source)

In order to minimize the error of the loss function, the back propagation process is applied. The principle is to update the weights of the model using the gradient descent method based on the loss value obtained and the actual weights, to direct towards the optimal solution.
This method allows to understand how each neurons influence each others, and conclude if a specific one should be higher or lower in order to satisfy the minimization of the loss function value. Finally, each weights w is updated with a learning rate η, which can be visualized as the step size taken towards the optimal solution.

$$w_n = w_n - \eta * \frac{\partial LossValue}{\partial w_n}$$

$$\frac{\partial LossValue}{\partial w_n} = \frac{\partial LossValue}{\partial Output_{Node_x}} * \frac{\partial Output_{Node_x}}{\partial Input_{Node_x}} * \frac{\partial Node_x}{\partial w_n}$$

Special sets of functions can be applied to tune the model, and influence the inputs and outputs of the neurons. These are usually applied in order to avoid overfitting and/or underfitting of the model on the data. These functions can be used as a neurons deactivator and/or as a value restrictor.

- ❍ Dropout Functions
- ❍ Activation Functions

Dropout functions - The dropout is a function deactivating some neurons on given layers, and only enabled during the training of the model. It is applied in order to avoid the model to learn the data by heart, which would likely overfit it and lower the prediction power. This issue is mainly due to the structure of a full connected Neural Network. In such structure, each nodes are learning to be dependent to each other, which is alternating the individual power of each neuron to decide, and restrict the model the learn more robust features. Consequently, the model would have a low accuracy rate on the new data. To solve this issue, a dropout function can be applied, to de-activate some neurons and allows the model to learn more robust features from the input data.

Activations functions - The activations functions are used to determine the output of a neuron, and can be either linear or non-linear. The linear activations functions does not apply anything to the output of the node. However, the non-linear activations functions are usually used when we want to output, either probabilities and/or a specific values; by either killing neurons value and putting it to zero, or restricting their power based on their value obtained. Such functions are: *Sigmoid, Tanh, ReLU, Softmax*. As example, the Softmax is usually used in the output layer when dealing with multiples classes, as we wish to obtain the probabilities that the input belongs to a certain classes output.

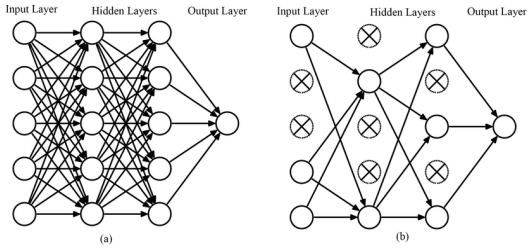


Figure 9. Visualization of the dropout function applied on a Neural Network (source)

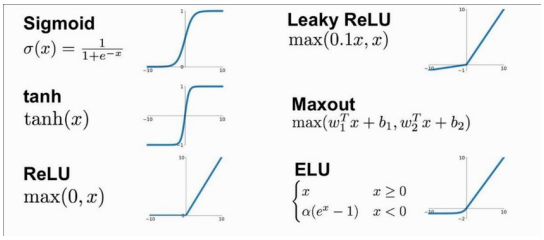


Figure 10. Visualization of few activation functions properties (source)

2.4.2. Structure of Convolutional Neural Networks.

The Convolutional Neural Network [CNN] is an extension of the existing Neural Network structure, and additionally applying a convolution to the input data before feed-forwarding to the Neural Network.

The idea is quite straightforward, instead of feed-forwarding the raw data directly in the model, this last will learn to detect the features present in the data, and evaluate the output based on these.
This method is mainly used in order for the model to be more scalable and efficient towards different types of data. Indeed, with the original Neural Network structure, if we assume an image with a resolution of *1920x1080* as the data, the input layer would require $2,073,600$ neurons for each of the pixels. If the output layer have 20 nodes, the number of weights would be of $41,472,000$. As reminder, each of these weights need to go through the gradient descent process in order to be reevaluated and minimize the loss function, which requires a lot of computational power and time. Moreover, with this structure, the model is extremely sensible to the environment around the subject and

each of the pixel as they each represents a neuron, therefore a modification of the background would likely perturb the model prediction power.

In order to solve these issues, an alternative neural net structure named Convolutional Neural Network arised, and is based on three distinct layers.

- ❍ Convolutional Layer
- ❍ Pooling Layer
- ❍ Fully-Connected Layer

Convolutional Layer - The Convolutional Layer is the main process of the convolution, and where lies most of the calculation. This process is composed of the *input data*, the *filter(s)*, and output(s) called *feature map(s)*. The filter, as well called feature detector or kernel, is used to detect the features present in the data. Different types of filters can be used on the same data in order to detect different kind of features, outputting more or less features maps. For the process of the features maps creation, the kernel(s) are applied on the data and a dot product is calculated for each area. The kernel is given a certain size and certain values, and sweeping through the image at different position based on the stride given, applying the dot product between the image and the filter. The stride represents the distance and/or number of pixels the kernel is moving through the image matrix, and can either be small or big.

Pooling Layer - The Pooling layer is applied just after a convolution occurred and is used as a dimension reductor. Even though some information are lost during this process, it improves the efficiency of the model training by minimizing the complexity, and could possibly avoid an overfitting of the model to occurs. Such as the convolutional layer, the pooling layer is applying a similar method and is sweeping a filter across the output of one of the convolution obtained. However, instead of applying a dot product, a dimensionality function reduction is effectuated. Two of the most known pooling function are *max pooling* and *average pooling*. The max pooling is outputting the highest value pixel in the area of the image covered by the the kernel, while the average pooling is calculating the average pixel value using all the pixels of the image covered by the kernel.

Full Connected Layer - The Full connected layer is a Neural Networks, which instead feed-forwarding the original data, is inputted the features obtained by the convolutions and pooling processes.

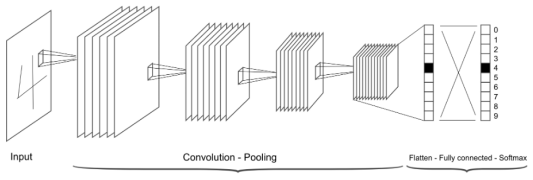


Figure 11. Structure of the Convolutional Neural Network (source)

2.5. Related Work in Adversarial Patches

Many researches papers about adversarial patches have been produced in order to perturb and attack the deep learning models. Each of these researches provided different types of results based on the situation, the scenarios and the techniques used to produce the adversarial patches. In this section, we are enumerating few papers highly related to the work improvement of this project we want to apply, and the work the authors have managed to apply.

We are beginning with **Ru.,Men.,Nin.,Fan. et Xia.[]**, where the authors have managed to provide a Trojan attack, which aims to attack deep neural networks model by triggering specific patterns, adversarial patches, maliciously inserted in the model training by model poisoning. Based on the specific triggers, the deep learning model is miss-classifying inputs when presented the adversarial patch that have been used during the poisoning. The authors state that the Trojan attacks are hard to detect in a given deep learning model. As once trained, the model readability is hard, and seems to act such as black-boxes, consequently incomprehensible by humans. The purpose of their research was to poison a deep learning model with specific triggered, defined by adversarial patches. And during testing, the model would naturally work as intended, but would miss-classify when presented the right triggers.
From another paper by **Fran.,Mak.,Nam.,Nic. et Mat.[]**, the authors present a frameworks *Sparse-RS* allowing to provide multiples types of attacks in the l_0,l_2 and l norms. Among these attacks, they provided a perturbations based on the use of adversarial patches in square based form, where their constraints of input domain was just the localization of the adversarial patch. The purpose of their research was to apply a black-box attack on unseen image based on adversarial patches applied at random locations on the attacked image.
Another paper from **Sharif et al.[]** presented the possibility to use glasses, where the frame of the glasses are applied as patches. The purpose was to foolish a face recognition model to impersonate other people, when the glasses are wear by a specific person which have been used to produce the frame. Moreover, **Duan et al.[]** proposed *AdvCam* to compute unrestricted perturbations that would take effect in real world, such that the patterns produced are claimed to be stealthier than other adversarial patches techniques. Finally, more recently, **Ani.,Aksha.,Koni. et Hamed[]** presented the possibility to provide adversarial patches based on the spatial context of the deep learning model. They claimed their attack to work only when the detection of a given object is influenced by the object itself but as well by the surrounding, therefore the context, since their adversarial patches does not overlap with the attacked object.

3. ADVERSARIAL PATCHES TRANSPARENCY

In the section above, we have taken the time to enumerate numbered of researches effort around the adversarial patches based attacks. We had the possibility to observe a multitudes of different techniques - *glasses frames, data poisoning, localization pattern,...* - in order to successfully miss-classify an input, by the generation and the appliance of a specific patch on this input, wrongfully triggering the model.

In this section, we are discussing about the objective of this project. As alternative researches, we want to improve

the established attacks based on adversarial patches, with new concepts that has not been introduced yet, and be able to use the adversarial patches in **real-time** and a **real-world environment**. In order to achieve this, we need to observe the researches that have been achieved up to now, and study the limitation, lack of creativity and/or improvement that could be done.

Target Position - First, as we have observed in the last section, some of the researches have not limited the domain of appliance of the adversarial patches on their *target position*. Indeed, these attacks are not specific to the target localization, as the adversarial patch can lies outside the target to miss-classify it. This technique is efficient, but accordingly the usability of this attack lays in the theoretical section, as it would not be usable outside of the computer usage context. Therefore, for our attack, we should rely on producing an adversarial patch applicable directly on the target intended to be miss-classified.

Black-Box - Secondly, most of the researches on adversarial patches are based on *white-box attacks*, where the data, parameters and the structure of the model attacked are retrieved. Based on mathematical analysis, it allows to study and directly interact with the flow of the gradient descent of the model to influence the output result. This is an efficient technique, however it does mean that the attacker must have access to the model structure. In industrial applications, the possibility for an attacker to be aware, understand what model have been used and/or how it has been structured is rarely possible, therefore this technique lies in the theoretical attack possibilities. Consequently, for our attack, we should rely on the usage of black-box attacks, with access to pairs of input/output, where we can only observe the output of the model given a specific input, and provide modification to this input until we obtain a required output.

Patch Form - In third, we must discuss about the patch generation itself. Generally, the adversarial patches are limited by their forms and structure variations, only producing one type of patch. Indeed, the adversarial patches generated observed are usually defined in a square form, but this limitation restricts the attack extend-ability and usability in different domains, as the environment in which they are being used may require a specific form of patch to be correctly applied. This form restriction could be lifted to allow a better diversity of the adversarial patch usability and localization in diverse environment. Nevertheless, few exceptions have as of now been observed, with researches which already worked on this aspect, where we can find adversarial patches under the form of butterfly; separated in multitudes of parts; or even on a pair of glasses. For our attack, we will likely work on providing adversarial patch under two different forms : *square and circle*.

Patch Structure - Lastly, a similitude we can observe on the adversarial patches attacks, is the contents of the patch. We can see that the content of the patch itself is rarely introduced and usually not limited by the attack. Indeed, the attack is based on the efficiency of providing an adversarial patch capable to miss-classify the model, but the color, mixture and appearance - *not the form* - of the patch is not discussed. This issue have to be raised, as the human factor have to obviously be taken into consideration during an attack and the use of an adversarial patch in a real environment is suitable. A human reasoning remains superior than an artificial intelligence, where the AI is commonly applying the task it has been trained to do, the human can intervene if it manage to recognize that a possible attack is about to occur, mainly due to the visibility of the adversarial patch observed. Consequently, for our work the adversarial patch transparency has to be raised in order to limit the visibility of the patch for the human factor, and the adversarial patch will require a restriction of its modification capabilities in the color domain.

3.1. Industrial Applications Attacks

In this section, we are discussing about the domain of applications of the research. Different domains providing different solutions, we must define the purpose of this project, the ideas behind the work provided, and the results we have to obtain at the end.

Nowadays, the world of self-driving cars is rising, and progressively more and more autonomous vehicles are being introduced on the public roads. Nevertheless, in order to be efficient and operational in those conditions, these autonomous vehicles must to be based on the usage of *Artificial Intelligence* technologies, and the integration of *sensors systems*. These lasts, allows to the self-driving vehicles the capabilities to become aware of their environment; detecting lines on the roads, detecting surrounding area and the entities it contains, the road paths and the traffic signs; in order to respect the driving rules and keep the passengers of the self-driving vehicles and the others road users safe. Obviously, these sensors systems may work together in order to be efficient at detecting objects, obstacles or signs.

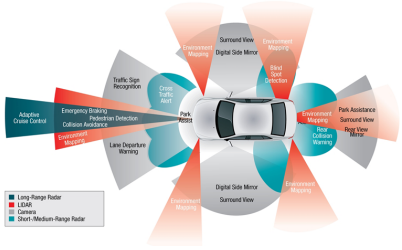


Figure 12. Autonomous Vehicles Sensors Ranges of Actions

To understand the complexity of self-driving components, here is an exhaustive list of the types of sensors we can encounter in a self-driving cars:

Lidar - The lidar (light detection and ranging) is a 3D laser scanner located at the top of the self-driving cars on the roof, used to scan the surrounding environment. Lidar functionality is straightforward, and is based on sending thousands of laser beams in all the directions, each of these beams are lately reflected on the surrounding objects and sent back to the sensor. It does create a point clouds, allowing to be aware of the objects, and environment, position and distance from the self-driving cars at any moment.

Radar - The radar, located around the car, has the same purpose as the lidar, detecting its surrounding information; but it is focused on defining the state of the objects it detects such as its angle, speed and orientation. The radar functionality is similar to the lidar, and based on sending radio waves, lately reflected on the environment and sent back to the sensor, allowing to obtain information about the surrounding objects.

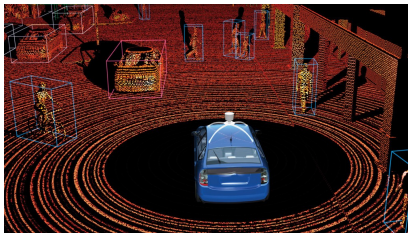


Figure 13. Operation of the LIDAR sensor on an Autonomous Vehicle (source)

Sonar - The sonar, situated around the car, has the same purpose as the last two sensors, allowing to detect its surrounding environment. However, compared to the lidar and radar based on the speed of light, the sonar is based on the speed of sound and is accurate in short distance. Nevertheless, the functionality of the sonar is similar, emitting echoes that are lately reflected to the sensor.

Camera - The camera, is a vital element to the self-driving vehicles, based on computer vision, this sensor is responsible of detecting and classifying the traffic signs as well as capturing visual data. Multiples cameras might be used in different position on the self-driving car, to approximate the distance of the objects and environment, and to certify that the detection by the AI model under different angles provide the same results.

IMU and GPS - There exists two more sensors systems situated inside the car, out of the scope of this project, with the purpose of measuring the vehicle movement, position and state at any moment. The IMU, is a gyroscope an accelerometers allowing to track the speed and angle of the vehicle; and the well-known GPS, used to track the position of the vehicle based on Earth.

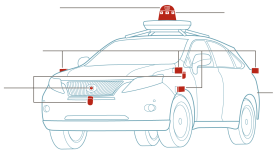


Figure 14. Autonomous Vehicles Sensors Positions (source)

As we can observe, self-driving cars are mounted with many sensors systems used in order to keep awareness of the vehicle state and environment at any moment. However, what would happen in case of an failure, if one of the sensor was miss-behaving or failing? Since the functionality of the lidar,sonar and radar are quite similar, if one of these sensors would happen to fail, the others sensors could compensate this loss, and work together in order to safely recover the autonomous vehicles back in a safe state. On the other hand, the camera sensor is an independent system, only capable to work on the vision of its environment, the other sensors would not be capable to compensate it, limited by their functionality. Therefore, if a failure happens with the camera, the other sensors are incapable to detect it and would simply keep on their operation.
The critical part is that we must be aware that these self-driving vehicles are transporting passengers, and consequently these passengers could be put into immediate danger if the camera operation would happen to fail during one of the trip.

For this reason, we have decided to work on an attack based on the camera sensor system of autonomous vehicles. Mainly, we are focusing on attacking the detection of the traffic signs (not traffic lights), where the AI model would miss-classify a sign (i.e Speed Up instead of Stop Sign). As explained in the last section, this attack would be based on the use of adversarial patches, that will be modified and improved for the sake of this research objective. As discussed, the camera and the human are both based on the visual factor,and one may be capable to detect that an attack is occurring by noticing the presence of a patch on a traffic sign. This research is focused on minimizing this detection factor, and keep obtaining an efficient adversarial patch.

3.2. Requirements

This research regarding the use of multiples domains of Computer Science and complex tasks, it is therefore required to define the essential requirements in order to be able to fully understand the process and the decision made during the evolution of this research. Consequently, this section of this paper allows becoming aware of the minimum knowledge required, and to auto-evaluate, such that each tasks in further section are fully understood.

We are beginning with the most important requirements related to the *Mathematical background*. Indeed, we have already observed in the state of the art, the necessity of such requirement, as mathematics govern the Computer Science domain in order to define different algorithms universally readable; therefore the need of understanding medium level mathematical is required. Math is necessary as, by pursuing the project, we are defining the different techniques used and defined, based on mathematical reasoning, and it did allow to solve many issues encountered. Moreover, we will eventually conclude providing the universal new mathematical formula defining the algorithm implemented such that it can be used for further researches.

The second requirement is obviously knowledge in the *programming domain*. Naturally, in order to solve this research, we had to use a specific programming language to allow the implementation of efficient algorithms to be designed. Undoubtedly, we will discuss in the following section, about the programming language used, and the reason behind this selection. Therefore, awareness in the concept of programming language, either at best syntax comprehension, is required to understand this project.

Finally, but not the least, knowledge in the *Artificial Intelligence domain* is required, especially Deep Learning. Certainly, we took some time to redefine few of the main concepts of the Artificial Intelligence in the state of the art. However, this requirement is actually necessary as the fundamental idea is to understand the reason of having worked on a project based on self-driving cars, and not related to any other domain. As discussed in earlier section, we have defined the dangerousness of the possible attacks in the industrial applications that could occur on the autonomous vehicles, and consequently justified this choice. Moreover, to have the ability to attack self-driving cars components, we must be aware about the operation of the Artificial Intelligence inside, and

to apprehend the possibilities to manipulate such that we obtain wanted results and an efficient attack.

4. INSTALLATION AND SET UP

In order to have a running project environment, and be aware of the functionalities needed, this section is dedicated to define the entire necessary tools to be installed. We are discussing the programming language selected, the installation and the reason of its use. It will be shortly followed by an enumeration of the necessary libraries to have the project fully operational. Finally, we will close this section, by introducing the simulator Carla, that we are using along this research and the necessity of its use.

4.1. Programming Language

For this research, we needed to operate with a programming language proficient in the development of Artificial Intelligence, and concurrently widely adopted for the sake of providing the necessary libraries supporting the project objective. Due to these reasons, we have determined that the programming language Python would suit these requirements, notably **Python 3**. Indeed, Python is providing support for abounding of libraries, and progressively gaining notoriousness among the community. Moreover, the syntax and usage simplicity provides an efficient approach to implement possibles attacks on Artificial Intelligence based systems.

Note: *The version of Python that have been used is the 3.7.4, and can be downloaded via this link Python 3.7.4*

4.1.1. Libraries. The project is based on the use of numerous libraries, necessary to be installed ahead, to avoid errors during the running process. A few of the libraries are related exclusively to the Artificial Intelligence component, while others to specific integration of pre-implemented algorithms.

The first library we are introducing is **Tensorflow**. Tensorflow is an open source library specifically designed for tasks evolving over Artificial Intelligence, allowing to perform on multitudes of machine learning techniques. The advantage of the Tensorflow library, is the fact it has been implemented over another known library, *keras*. Keras being a deep learning API available on Python, it grants the possibility to work on Machine and Deep Learning synchronously.
Tensorflow is principally used in this project for the production of the Deep Learning models, from scratch and pre-built.

The second library necessary for this research is **Pymoo**. Pymoo is a framework designed to be used in order to solve multi-objective optimization problems, it does provide diverse algorithms based on evolutionary behaviors, that can be modified for the sake of solving a specific optimization task. For this project, we have provided a scratch version of a genetic algorithm to understand its operation, but lately we switched on using the genetic algorithm provided by Pymoo allowing to obtain better performance of the attack, the results are discussed in the following section.

Note: *The Tensorflow library can be downloaded here Tensorflow 2.3.0. The Pymoo library can be downloaded here Pymoo 0.5.0*

5. EVOLUTIONARY ALGORITHMS

Nowadays, the Evolutionary algorithms are widely used to solve optimization problems based on approaches using meta-heuristics methods. These algorithms are designed based on observation from natural biological evolution among a population, allowing to produce different generation leading to an objective of optimization of a given problem. Of course, there exists different classes of evolutionary algorithms based on the approached used and the difference in the process, among these we can find the three most popular being Genetic algorithms, Evolution strategies and differential evolution. All classes provide different advantages that are going to be enumerated.

Genetic Algorithm - The genetic algorithms (GA) are the most basics and earliest type of evolutionary algorithms. This algorithm is based on the use of candidates, similarly called chromosome, belonging to a population, where each of the candidate carry as set of variables, similarly called gene, that need to be optimized.

Genetic algorithm process for one generation:

1) Population of Candidates
2) Apply fitness function on individual candidate, keep the best candidates as new population. (rank or tournament)
3) Reproduction by applying mutation and crossover on new population (create child and mutant)
4) Go back to *1)* for next generation

As we can observe, in the process of one generation, the genetic algorithm is proceeding the population candidates P over a fitness function F (specific to the problem to optimize), where each candidate C returns a value v, either leading towards the solution objective or not $F(C) = v, C \in P$. Based on the values returned by each individual, the GA can retrieve N set of candidates based on two methods; rank based, only the N best candidates are retained for the later population P'; tournament based, taking the best candidate b among a smaller set of candidates s, $b = Best(s), s \in C$.
Each candidate of this new population P' is further proceeding through the reproduction state, where the mutation and crossover phase are processed. The mutation phase is based on pairs of candidates, similarly called parents, using the crossover process (two points, multiples points, random points) retrieving part of their genes to produce a child candidate, where each of the child candidate, with a probability, have the potential to become mutated.
The next generations are pursuing the process until the optimization objective have been reached, or a stopping criteria have been met.

Evolution Strategies - The evolution strategies (ES) have been designed to process such as Genetic Algorithms, and are

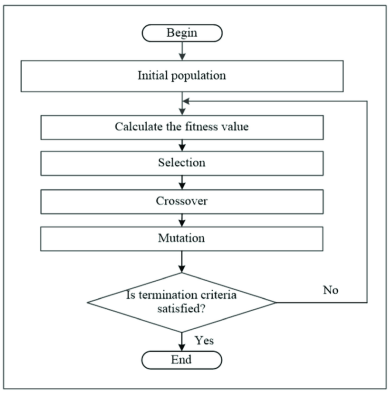


Figure 15. Structure of a Genetic Algorithm process (source)

similar in their fundamentals concept. Nevertheless, as the name suggests, the ES provides alternative methods based on *strategy parameters*, being used to determine the amount of changes allowed for the population, during the selection and reproduction phases, called *recombination*.
This addition grants an advantage of efficiency towards the search of the optimal solution for a specific problem, and consequently optimize the research domain.

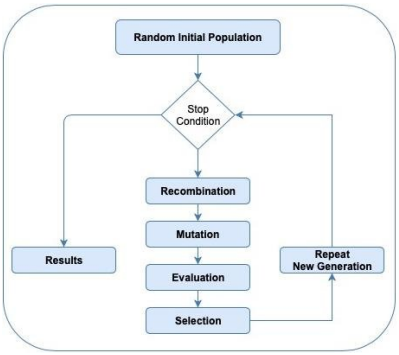


Figure 16. Structure of an Evolutionary Algorithm process

Among these methods, we may observe one based on increasing or decreasing the amount of changes allowed based on the number of successful mutations retrieved, the *1/5* rule. Another method described, is to apply these strategies parameters directly inside the evolution process, allowing to provide auto-settings to appropriate values for each of the generation.
However, in this group of methods, one that tends to be most widely used is the covariance matrix adaptation (CMA-ES). Using a similar mechanism to the gradient descent, it is based on estimating a productive direction towards the objective solution and apply the changes in this direction.
Another variation of the ES compared to GA, lies in the reproduction phase, where the ES is based on fitness ranking of the candidates instead of their fitness values. Indeed, instead of using a pair of parents as GA, ES provides the possibility to use one, two or more parents to create a single child. Moreover, it is possible that the parent does not participate in the next iteration where the children are replacing the parents $(mu, lambda) - ES$, while the common solution is to append the parents to the produced children $(mu + lambda) - ES$. From these variations, we may specify that the evolution strategies are further based on the deterministic aspect of the evolution than the probabilistic ones.
Similar to GA, the next generations are pursuing the process until the optimization objective have been reached, or a stopping criteria have been met.

6. METHODS

We have defined the problem related to the adversarial patches attacks, presented by recent researches, due to a lack of restriction. We have concluded that we wanted to work on the production of adversarial patches with a restriction in the color domain, such that their visibility towards the human is lowered. As stated, the attack is evaluated in the autonomous driving environment, principally the traffic sign detection, due to the impact that a successful attack could lead.
As we are evolving in the detection of traffic signs, we needed to be able to detect the road signs, and the production of an Artificial Intelligence model (*ResNet*) was required for the classification of these last. Regarding the attack implementation (*T-Patch*), we needed to be capable to produce efficient adversarial patches able to miss-classify the AI model defined earlier, and have adopted the use of a genetic algorithm, allowing generations of population to reproduce, in order to optimize a specific fitness function.

Note: *Python programming language have been adopted, allowing a simplification in the implementation and the access to numbers of libraries provided for data science.*

6.1. Model and DataSets

The first step was to define an Artificial Intelligence capable to detect traffic signs. To achieve this, we have applied a dataset provided by the *German traffic Sign Recognition Benchmark (GTSRB)* containing approximately fifty thousand of images of different traffic sign that may be needed to train our AI model. To train the model, we have applied the deep learning API *Keras*, providing multiples models based on different architectures that can be easily alternated for the purpose of this research, such as ResNet or VGG. We have focused on the usage of the *Residual Neural Network (ResNet)*, as its architecture has already been described to be efficient for images classification, and is based on the convolution operation, defining the ResNet as a Convolutional Neural Network (CNN) architecture. During the training phase, we have taken 60% of the dataset for the training, and 40% for the testing, with twelve epochs and thirty-two batches.
With these settings, we have obtained an accuracy of classification for the model near 98.6%, and will be able to be employed for the attack with adversarial patches.

Note: *Models trained with Keras are only allowed in order to avoid unexpected behavior.*

6.2. Color Restriction

We have defined that we wanted to produce an attack with a restriction of the adversarial patch on the color domain, in order to obtain a stealthier attack. Therefore, we have established a rule to procure a patch P with restricted modification $ResM$ in the pixels values p to obtain a color c nearby the original targeted image O color domain.

$$min\ Difference(ResM(P_p), c), c \in O_p$$

To achieve this objective, we have to analyze the content of the targeted image on the current pixels values, and produce a global method to restrict the pixels values of the patch based on the results obtained.

The first part of the method is to analyze the *mean value* of the pixels of the target image $Mean(O_p)$. However, we must take into consideration that the background pixels are influencing this analysis as it constitutes the image, therefore we restricted the target image based on the form f of the traffic sign attacked (*circle, square, triangle*), resulting in obtaining more accurate outcome for the mean value. The result of this mean is providing an overall analysis to define the main color of the targeted image (*red, blue or green*).

$$Mean(f(O_p))$$

The second part of the method rely on *counting and totaling* the amount of diverse set of pixels contained in the targeted image based on the three channels *RGB* values, and retrieving the one with the highest frequency (*we must still take into consideration the background*).

$$max\ Count(f(O_p))$$

With these results, we manage to obtain the main color MC and the main values MV for the RGB channels pixel of the target image, and can produce the initial adversarial patch : $AP = [MV^{height}, MV^{width}]$. We may now define a threshold, which abstain from changing the current color domain of the adversarial patch AP and restricting the possibilities of pixels values modification, while limiting the impact on the efficiency of the attack. To achieve this, we can for each of the channels set the maximum amount of deviation x or y, from the original pixel values, in the positive Pos and negative Neg direction. The deviation x being related to the channel holding the main color the deviation will be higher, while the y related to the others channels will be lowered.

$$\begin{gathered}[R, G, B] \in MV \Rightarrow \\ Pos(R + x, G + y, B + y), \\ Neg(R - x, G - y, B - y), \\ x > y, R \in MainColor\end{gathered}$$

6.3. Pymoo

In order to attack the AI model, we need a method to produce adversarial patches capable to miss-classify the model prediction once applied on the target image. For the implementation of this attack, we have introduced the principle of genetic algorithm, which is based on a set of population, generation and mutations, trying to optimize towards a specific objective function. In our research, the population would be defined as the set of adversarial patches, and the objective function would be the minimization of the square error between the prediction probabilities of the model on the target image with adversarial patch $M(T_{AP})$, and the target attack probabilities A.

$$min\ SquareError(M(T_{AP}), A)$$

To simplify this step, we have applied *Pymoo*, which is a multi-objective optimization framework providing multiples all-ready evolutionary algorithms, that can be altered for the sake of a distinct problem. In our research, we have re-use their implementation for the genetic algorithm, and modified it in order to handle images and adversarial patches inputs. An advantage of Pymoo is the fact it can optimize any input, therefore we can concurrently optimize the position (x,y) and the orientation (z) of the adversarial patches.

In our implementation, we have provided three distinct types of attacks, and therefore defined different objective functions (*still based on the square error*) : *target attack*, *untarget attack* and *best attack*.

Target Attack - In the target attack scenario, adversarial patches must be able to miss-classify the model to a specific target label. Therefore, the objective function is defined as maximizing the target label probability while minimizing all the others labels probability. As said earlier, the objective function is based on the square error between two sets of probabilities, the prediction by the model $M(T_{AP})$, and the target prediction A we want to obtain, therefore the goal is to obtain and objective function outputting zero.

$$\begin{gathered}ObjFunc = min\ SquareError(M(T_{AP}), A) <= 0, \\ A = [0, 0, 0, 0, ..., 1, ..., 0], \\ M(T_{AP}) = [0.03, 0.1, 0.0005, ..., 0.8, ..., 0.07]\end{gathered}$$

Untarget Attack - In the untarget attack scenario, adversarial patches must be able to miss-classify the model as far as possible from the original label. Therefore, the objective function is defined as maximizing all the labels probability while minimizing the original label probability. In comparison to target attack, in this situation we are not concerned about the class of label the model has classified the image into, as long as the original label is not predicted anymore.

$$\begin{gathered}ObjFunc = min\ SquareError(M(T_{AP}), A) <= 0, \\ A = [1, 1, 1, 1, ..., 0, ..., 1], \\ M(T_{AP}) = [0.3, 0.4, 0.04, ..., 0.00003, ..., 0.04]\end{gathered}$$

Best Attack **-** In best attack scenario, compared to the other attacks, the objective is to provide an adversarial patch capable to miss-classify the model in the label, different from the original one, with the highest probability. We can define the best attack as an evolving target attack, as the label with the highest probability will be prioritized at each generation and therefore A will be modified adequately.

$$ObjFunc = min\ SquareError(M(T_{AP}), A) <= 0,$$
$$A = [0, 0, 0, 0, ..., 1, ..., 0],$$
$$M(T_{AP}) = [0.03, 0.1, 0.0005, ..., 0.8, ..., 0.07]$$

6.4. Metrics of Evaluation

To determine the efficiency of our algorithm implementation, and the success in the resolution of the problem defined in this research about the visibility of the adversarial patches. we have defined several metrics for the evaluation of the algorithm, based on time, the visibility and the results obtained using out implementation.

Time **-** The time metric (TM) is contributing to evaluate the amount of time needed to produce the adversarial patches. To achieve this, we have evaluated the time needed to create exactly one adversarial patch (*uncomplete attack*), the time needed to create all the adversarial patches (*full attack*), and defined the existence of a relation based on the size of the image attacked.

Similarity Index **-** Multiples Similarity Index metric (SSIM,SCC, VIFP, UQI) is provided to evaluate the similarity value between two images, and is based on the image degradation change in its quality information. We are using these metric to evaluate the efficiency of color restriction defined and observe the adversarial patch transparency, by comparing the original image O, and the image integrated with the adversarial patch P.
To obtain accurate results, this metric is being evaluated simulating the distance from the attacked target traffic sign (*slowly resizing the image*), and calculated as soon as the traffic sign has been detected. A threshold for each of the similarity metrics is set, such that the value obtained should stay above an objective set. This metric will provide a simulation evaluation of the time needed for the human to visualize the attack on the traffic sign.

Attack Accuracy **-** The Attack Accuracy (AA) is evaluating the efficiency of the adversarial patches created by our implementation based on Pymoo. This metric is calculated based on the percentage of successfully created patches on the different types of attacks (*target,untarget,best*), observing the amount of labels which manage to change, amount of attack successfully over a specific probability, and finally based on the form of the adversarial patches. (*circle,square,triangle*).

7. "ADVERSARIAL_COLOR_PATCH" - A PATCHES TRANSPARENCY ATTACK

To recap all the sections that have been defined and discussed earlier, we have designed and implemented a new adversarial attack against artificial intelligence models, based on images camera treatment, applied for self-driving cars. This new attack is based on the appliance of adversarial patches, produced and supported by a genetic algorithm provided by Pymoo, such that the adversarial patches visibility concerning the humans awareness is minimized and in relation with the attacked images color domain, leading to the predicted results of the artificial intelligence model being successfully altered. We have consequently named this new attack $Adversarial\_Color\_Patch(ACP)$.

To understand the capabilities and functionalities of ACP, we are detailing in this section the full structure of the attack regarding the reasons behind the algorithms and the mathematical operations applied. In order to do so, we are firstly introducing the architecture of the ACP application for each of its components, providing their necessary information and usage, to have the capability to produce the final generation of adversarial patches, and be aware on how to operate the attack for alternative experiments. We are later continuing with $Pymoo$, where the details to determine the parameters and the structure of the algorithm applied will be explained further than in the last section [Pymoo], and necessary to understand how we found an ideal trade-off between efficiency and speed improvement for the attack. We will pursue by detailing the usage of a modified version of the ResNet architecture originally applied for image classification, in order to deploy our own artificial intelligence model capable to detect road signs, and able to be operated and later attacked using our developed attack ACP. Finally, we will provide a full explanation on how to manipulate efficiently ACP, to be either operated on our model with our images and/or other different models (*Keras, PyTorch*) with others images. Therefore an understanding of the structure of the application is necessary.

7.1. Adversarial_Color_Patch - Structure

As said, we are in this section enumerating the main components of the ACP attack application. Among the presented components, we will be introducing folders and python files, which have been required to design, and/or are required to control/parameter the ACP application. The importance of each component being necessary, the important ones to be aware of are **images_, Models, patches**, and obviously the launcher of the implemented ACP defined in $Pymoo\_patch.py$. Each of the components will be detailed and explained in the following section, regarding the summary provided by the graph structure of AdversarialColorPatch - PymooM Structure.

archive **-** The very first component introduced is the **archive**, representing an amount of around ten thousands images of different road signs, it has been used in order to train the artificial intelligence model required for the road signs classification process. Among all the images contained in the data, we may retrieve up to thirty-three different road signs classes, divided into training and testing phases, labeled for supervised learning. These data are later applied in the $model.py$ file detailed further in this section.

***images_** - Next, is the **images_** folder, and is important as it contains the starting point for the attacked images. Indeed, the ACP attack requires images to be attacked, and adequately the purpose of the folder is to retrieve the images contained inside it, which will be applied to the ACP. Obviously, any volume and/or types **.jpg/.png** of images may be used and are automatically retrieved from this folder at running time.

***Models** - An additional important folder to take into consideration is the **Models**, as it has similar utilities to **images_**. Indeed, APC requires an artificial intelligence model to be attacked, usually related to the type of images attacked and/or vise-versa. Obviously, ACP may be able to perform with any type of models types **Keras, PyTorch** and apply the corresponded attack on it. Therefore, we may provide all the necessary models to be attacked in this folder, and ACP will retrieve any models with the extension **.h5** at running time.

***patches** - With the attack applied using ACP, we must at a specific point in time, require the results obtained to be saved. To do so, the folder **patches** has been created to save all the graphic and images related outcomes. Indeed, for a single attack with ACP, the final adversarial patches obtained are saved in the corresponding sub-folders $circle$,$square$ and $triangle$ regarding of their forms chosen (*circle, square, triangle*). Moreover, in order to obtain a representation regarding the position and orientation of the constructed adversarial patches on the attacked images, we have additionally provided a fourth sub-folder, containing the adversarial patches applied on the attacked images. The purpose of this inclusion, is required in case of a real application use, as it would benefit to provide an easier reproduction of the attack for the appliance of the adversarial patch on the real object/image.

***Results** - We have just introduced the existence of the **patches** folder to preserve the results obtained for the adversarial patches and the attacked images, however as explained only images related ones. Indeed, we may need to retrieve additional details regarding the efficiency of the ACP attack, if it has been successful, partially successful, and/or understand in which alternative classes the attacked images may have been predicted to, with a certain probability. Therefore, we have provided a **Results** folder, where we managed to save for each of the attacks, their capabilities and efficiency written under multiples **.csv** files with the following information: the original class of the attacked image, the attack type applied (*target,untarget,best*), the corresponding adversarial patches forms and size details compared to the attacked image, and finally the position and orientation of the adversarial patches. With all these information, we may be able to track in the **patches** folder the corresponding adversarial patches regarding the results, and be used for a real application scenario.

Attacks_Comparaison.py - In order to be aware of the efficiency and improvement of ACP for the adversarial patches attacks, we must be able to compare attacks using identical model and images attacked, with others existing attacks based on adversarial patches. In $Attacks_Comparaison.py$, we are comparing the results obtained with an ACP attack with three others attacks which have shown their efficiency, namely $SquareAttack$, $AdversarialTexture$ and $AdversarialPatch$. The results of the comparison obtained are later detailed and discussed in DISCUSSION section of this paper.

model.py - As explained earlier, we have implemented a modified version of the ResNet model, required in order to be able to classify road signs. In $model.py$, we have provided a modified version of ResNet based on the use of the Convolutional Neural Networks structure, and is using the data which have been provided by the **archive** folder introduced earlier. Producing a trained final model, it may be retrieved in the **Models** folder under the name $Road\_Signs\_classifier\_model.h5$. A further detailed representation and modification of the ResNet structure is introduced later in Road Signs Detection - ResNet Modified.

SSIM_Metric.py - As a reminder, the objective of this paper is to provide a new attack based on adversarial patches, where their transparency is maximized as much as possible, while minimizing the impact on the efficiency of the attack. To be able to observe the fulfillment of this objective, we must apply several metrics regarding the visualization and observation of the original image compared to the attacked images. Used as a measure of similarity between two images, these metrics, each applying different methods of similarity algorithm, are the *SSIM, SCC* and *MSSIM*. The purpose of applying several of these metrics, is to be able to obtain an overall average of the transparency capability for the adversarial patches.

Pymoo_patch.py - Finally, the most important part of this structure is the ACP attack. The $Pymoo\_patch.py$ is representing the starting point of the ACP, indeed this is where we can apply the adversarial patches attack with the images provided from **images_** folder and the model from **Models** folder. Once launched, we may define several different parameters for the attack, such as the form of the adversarial patches applied, and/or the form of the images/object attacked , these parameters are important to be defined and understood in order to obtain an efficient attack. A further detailed, purpose and functionalities regarding the usage of $Pymoo\_patch.py$ is provided further in Adversarial Patch Generation.

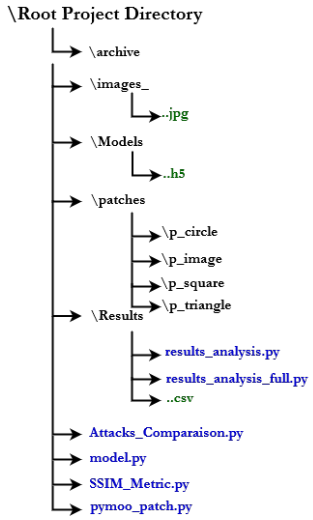


Figure 17. AdversarialColorPatch - Project Structure - Files

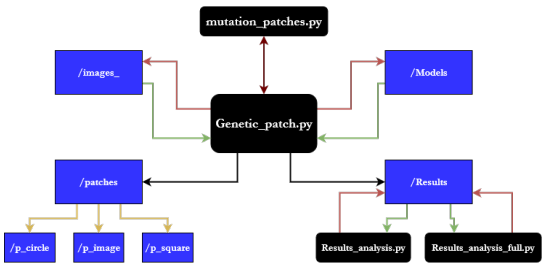


Figure 18. AdversarialColorPatch - Project Structure - Graph

7.2. Road Signs Detection - ResNet Modified

In this section, we are detailing the process manipulation of the ResNet (*Residual Neural Network*) architecture, in order to obtain an efficient model for road signs identification and prediction. Already been introduced in Model and DataSets, the ResNet architecture have been structured in order to skip connections in its structure, connecting activations of a layer to further layers by skipping some layers in between. This architecture have already been proven to act efficiently related to images classification for multi-classes prediction (1, 2, 3).

Applying the Python deep learning API *Tensorflow*, respectively *Keras*, capable to provide multiples models of different architectures, we have retrieved the ResNet50 architecture as shown in *Figure 18*, and modified its structure to produce an efficient model for road signs classification.
As observed in *Figure 19*, the modified version is based on splitting the convolution part of ResNet50 by two, followed by a fully connected neural networks, leading to a set of fourty-three classes representing multiples road signs classes.
As already introduced earlier, we have applied a dataset provided by the *German traffic Sign Recognition Benchmark (GTSRB)* containing approximately fifty thousands of images of different traffic sign. During the training phase of the model, we have applied 60% of the dataset for the training, and 40% for the testing, with twelve epochs and thirty-two batches.
With these settings, we have obtained an accuracy of classification for the model close to 98.6%, and is able to be employed for the normal classification of the road signs, additionally as the base model for the adversarial patches attack.

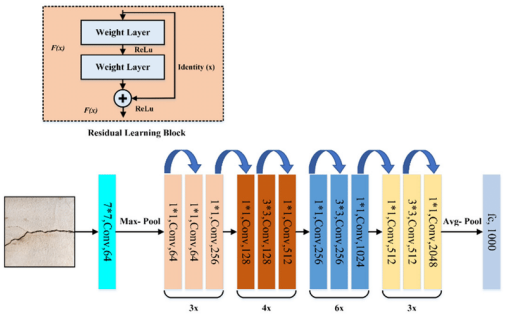


Figure 19. Architecture of the ResNet50 (source)

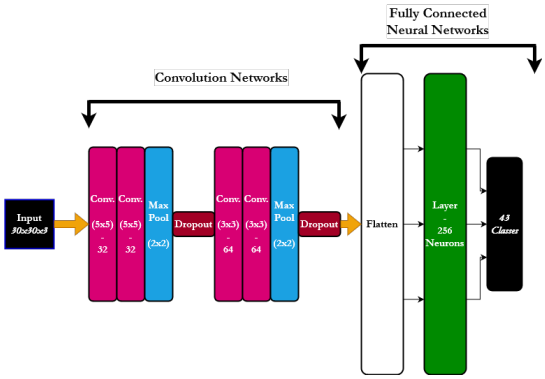


Figure 20. Architecture of the Convolutional Neural Network for traffic Signs Classification

7.3. New Adversarial Patch based on Attack Transparency

In this section we are detailing the research related to the maximization of the transparency for the adversarial patches, based on the color domain of an image. Yet, we have already introduced few of the functionalities and ideas in the Color Restriction section concerning this method. Nevertheless, we must detail the process that have been applied in order to find the optimal color restriction values, while keeping a balance on the efficiency of the attack and transparency maximization.

As explained earlier in the Color Restriction section, the color domain research for an attacked image is based on firstly determining the mean of the pixels values contained in the image $Mean(f(O_p))$. This is secondly pursued by evaluating the sets of pixels amount in the image based on their values, in order to retrieve the pixels with the maximum of appearance $max\ Count(f(O_p))$. Nevertheless, we must be aware that the background of an image have the capability to highly perturb this method of color domain calculation, as the pixels values would alter the final results. Similarly, as it would intervene on an artificial intelligence model prediction. Due to this fact, we have defined the presence of two restrictions for an attacked image: *position* and *form*.
The first restriction is based on the position of the attacked object on the image. Indeed, we may act on the manipulation of the image in order to assure that an attacked object is centered on the picture, and additionally near to it. This approach is allowing to minimize a considerable amount of the background.
The second restriction is about disposing the remaining of the background on the image, as it would surely still impact our method. Indeed, the road traffic signs have been designed with different forms (*circle, square, triangle*), and therefore some background may persists afterwards the first restriction. In this case, we have encountered this issue by additionally defining a parameter, that must be set during the attack, allowing to determine three different forms (*circle, square, triangle*) for the image. This procedure allows to remove the background by retracting the research area for the color domain based on the form provided.
Now that we have obtained the main color domain values appearing in our image $[R, G, B], x > y, R \in MainColor$, we have to set up the threshold around these values in the positive and negative direction $[x' - R + x, y' - G + y, y' - B + y]$ representing the capabilities and efficiency of our adversarial patch attack. Indeed, the objective of ACP is to manipulate the pixels contained in

Figure 21. Difference between a "Good" image and a "Bad" image for ACP

Figure 22. Example of an appliance of the "Circle" form on the *Stop Sign*

the adversarial patch for an artificial intelligence model prediction capabilities to be greatly altered. Moreover, the threshold that is declared must not be impacting the color domain difference between the adversarial patch and the attacked image, in order to keep the its transparency capability.
Assume, we would preserve a threshold with zero spread in the positive and negative direction $[0-R+0, 0-G+0, 0-B+0]$, no modification would be applied on the adversarial patch, and the attack efficiency and capabilities would adequately be extremely hard to be completed. Therefore, we have analyzed the impact of the threshold (*addition, subtraction*) for of the each individual pixel channel $[R, G, B]$ with different values spread, and obtained the following tables in Figure 21, Figure 22 and Figure 23.

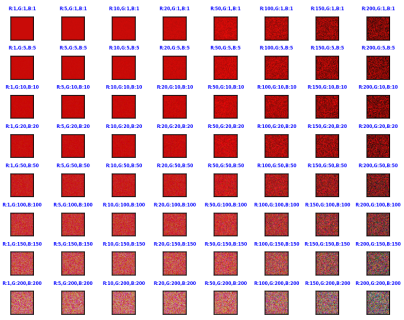


Figure 23. Different threshold restriction for the pixels values of the color domain of the *Stop Sign*

As we may observe based with the tables obtained, in order to balance the maximization of the adversarial patches transparency, while allowing the pixels threshold to be spread to keep an efficient attack, we have determined that the optimal values for the threshold would be defined as $x' = 50, x = 100, y' = 30, y = 30$, therefore $[50 - R + 100, 30 - G + 30, 30 - B + 30]$. If those values are not respected, the color difference from the adversarial patches and the attacked image would be too high.

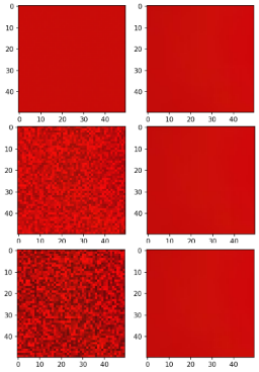


Figure 24. Difference between the usage of *minimum*, *average* and *full* threshold.

Figure 25. Difference between the usage of *minimum*, *average* and *full* threshold on the *Stop Sign*

7.4. Genetic Algorithm Applied - Pymoo

We have detailed in the last section how we managed to determine the optimal color values restriction, to obtain the color domain of the attacked image for our adversarial patches. As we have acquired this information, we may now have to apply it with the use of a genetic algorithm, in order to implement an adversarial patch attack that would behaves efficiently on provided images and adversarial patches. To achieve this, we have applied one of the algorithm provided by *Pymoo*, which have been introduced in Pymoo, and have adequately modified the algorithm to allows the usage of adversarial patches images in its algorithm, while introducing the usage of multiples objective functions based on the attack types defined.

The following section is divided in two parts, and is detailing the technical aspect of the application implemented. The first section is defining the modification applied to the Pymoo genetic algorithm in order to be used for the adversarial patches optimization. While, the second section is defining how we have the attack set up, where we apply the color restriction based on the image color domain, and how this information is used by Pymoo for its optimization. The overall algorithm is defining a new adversarial patch attack titled $AdversarialColorPatch$.

PymooM

As said, this section defines the genetic algorithm modification of Pymoo, which have been titled **PymooM**. PymooM is using all the data that will be provided to produce adversarial patches and optimize a given objective function. One of the first modification is related to the creation of the adversarial patches, as Pymoo is not aware of what is an image and is simply optimizing providing values based on an objective function, it only allows one dimensional arrays of values as input. Indeed, as an adversarial patch is an image, it is considered as a three dimensional array and unusable with this state, therefore, before being operated by Pymoo, we have to reshape each of the adversarial patches from three dimensional (*[height, width, channels]*) to a single dimensional array (*[height x width x channel]*). Additionally, Pymoo must be aware of the restriction color that have been defined in the last section, in fact the Pymoo algorithm is based on three parameters to produce an optimized array X, an upper maximum XU, a lower minimum XL and the amount of population, representing the amount of arrays x in X, therefore the amount of adversarial patches that will be produced by Pymoo . For an adversarial patch, the upper maximum XU represents the maximum color restriction values for each of the pixels as defined earlier $[R + 100, G + 30, B + 30]$, and therefore represented in a one dimensional array $[R + 100, G + 30, B + 30, R + 100, G + 30, B + 30, ...]$. Similarly, the lower minimum XL represents the lowest color restriction values for each of the pixels $[R - 50, G - 30, B - 30]$, therefore in a one dimensional array $[R - 50, G - 30, B - 30, R - 50, G - 30, B - 30, ...]$. XU and XL actually represents the upper and lower bound of a value contained in an array x of X, and Pymoo is required to stay between the provided bounds, while optimizing an objective function, to produce the values for x.

Additionally, we may not wish to restrict the adversarial patches position on the image, and allow the optimization to take into consideration the position and the angle of these last. Therefore, we have provided at the end of XU and XL, three additional parameters representing the *height*, the *width* and the *angle* of the adversarial patches (*[(height x width x channel)+posX+posY+angle]*). The parameter $posX$ and $posY$ values are based on the image form restriction provided (*circle,square,triangle*), representing the positions the adversarial patches are allowed to evolve on the image (*target the attacked object on the image*), reducing the area of research for Pymoo. Consequently, Pymoo is able to optimize the adversarial patches pixels values, while optimizing their position and angle on the image.

Once applied, Pymoo is outputting X, representing a set of arrays x respecting the restrictions that have been applied with XU and XL. Nevertheless, we must specify if the X provided are efficient and sufficiently optimized for our objective function. To achieve this, we must evaluate for each x in X the attack efficiency on the image. Therefore, each x (*withdraw of their last three values*) being a one dimension array, is reshaped to return into an adversarial patch image of three dimensions, and obtain four optimized parameters, the *adversarial patch*, the *height*, the *width* and the *angle*. With the provided information, we may for each of the adversarial patches apply them with their respective position and angle on the image to obtain an attacked image i. Lastly, for each gathered attacked image i, we may finally evaluate our objective function, based on the *SquareError* of the prediction obtained with i compared to the objective prediction, and send this information to Pymoo, in order to re-optimize the values of x in X for the next generation, to reach the objective prediction.

Algorithm 1 PymooM

```
Input :
attack ← (number,nc,u)
image ← Image to Attack of size h x w x c
model ← Model to Attack
form ← (circle,square)
formImage ← (circle,triangle)
N ← (1 ... 100)
Generation ← 200
XU, XL ← arrays of threshold variables of size 1 x ((h + w) + L + H + A)

for n in Generation do
    X ← GeneticAlgorithmPymoo(XU,XL, population)
    patches ← Reshape(X.3)
    heights ← X.2
    widths ← X.1
    angles ← X.0
    for individual in patches do
        patch ← CombinePatchFormtoImage(image, form, height, width,
angle, individual)
        preds ← Append(preds, model.predict(patch))
    end for
    attackEfficiency ← SquareError(preds, attack)
end for
Output : X[: N]
```

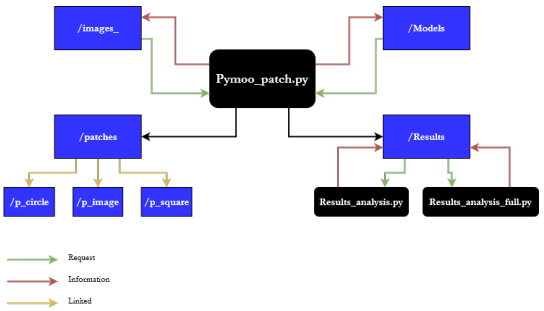


Figure 26. AdversarialColorPatch - PymooM Structure

T-Patch

In this section, we are defining the setting up of the parameter applied by *PymooM* defined earlier, titled **T-Patch**.

When the *Adversarial Color Patch* application is started, we are prompted with multiples parameters (*type of attack, image to attack, model to attack, form of the adversarial patch, form of the image object attacked*) necessary to be specified, and is purposely used by *T-Patch* to create the initial adversarial patch. With these information, the first operation to occur is the determination of the color domain of the adversarial patch with $mean(image)$ based on the methods described in New Adversarial Patch based on Attack Transparency. Additionally, as we may require adversarial patches of different sizes based on the attack requirements, adversarial patches of multiples sized *PatchSizes* are produced based on the attacked image size.

The color domain and the adversarial patches sizes established, the determination of the values related to the parameters XU and XL for each adversarial patch size, as detailed in the $PymooM$ section, are based on the restriction defined in New Adversarial Patch based on Attack Transparency. With the gathered information (*patchsize, model, image, form, XL, XU*), the $PymooM$ algorithm is able to successfully operate the attack, and produce the final optimized adversarial patches.

Algorithm 2 T-Patch

```
Input :
attack ← (number,nc,u)
image ← Image to Attack
model ← Model to Attack
form ← (circle,square)
formImage ← (circle,triangle)
N ← (1 → 100)

colorMain, meanR, meanG, meanB ← mean(image)

for size = 3...10 do
   PatchSizes ← Append(PatchSizes, image.shape / size)
end for

for patchSize in PatchesSizes do
   if colorMain == X then
      modMinX ← max(0, meanX − 50)
      modMinY ← min(0, meanY − 30)
      modMinZ ← min(0, meanZ − 30)
      modMaxX ← min(255, meanX + 100)
      modMaxY ← meanY + 30)
      modMaxZ ← meanZ + 30)
      XL ← (modMinX,modMinY,modMinZ,minX,minY,0)
      XU ← (modMaxX,modMaxY,modMaxZ,minX,minY,360)
   end if
   patches ← PymooM(XL, XU, patchSize, model, image, formImage, attack, N)
   Output:Patches
end for
```

Note: The source code and application of AdversarialColorPatch can be found at https://github.com/SeanAchtatou/AdversarialColorPatch

7.5. Adversarial Patch Generation

Being aware of the structure and the functionalities of the $AdversarialColorPatch$ application, we can pursue by detailing the use of ACP in order to generate the adversarial patches, and attack a distinct image. Therefore, in this section to ease this description, we are providing an example of the application process in its actual state, based on how to use the several provided folders and set up the parameters to start an efficient attack.
We will observe that the process of APC is separated in three parts, each detailed further below:

1) Setting up the APC application
2) Starting the attack
3) Retrieve the adversarial patches and results

Setting Up
Ahead of starting an attack, we must be aware about what we wish to attack. Indeed, the objective and capabilities of ACP is to attack an artificial intelligence model regarding its predictability efficiency towards a chosen image. Therefore, for our example, we have chosen to attack the following model $Road\_Signs\_classifier\_model.h5$, which have been implemented in order to be able to detect up to forty-three different road signs. To be used by APC, we must place the chosen model in the location folder regarding the models, which will be retrieved during the attack, namely **Models**.
Once the model has been correctly set, we can now require an image to be attacked. In our case, regarding the model which have been chosen, it is requiring a picture related to road signs. Reminder that the attacked image must be centred, we have chosen a road sign representing a *Stop sign*. In order to be usable during the attack process of APC, we are placing the image at the following location **images_**.

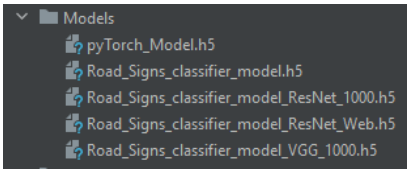


Figure 27. Adversarial Color Patch - Models Folder

Figure 28. Adversarial Color Patch - "Stop Sign" image chosen to be attacked

Note: *All the models must be of the extension .h5 in order to be read by ACP. On the other hand, all type of images (.jpg,.png) may be used.*

Launching an Attack
Once we have set up our model and image to be attacked, we may now be able to start an attack using the $AdversarialColorPatch$ application. In order to launch an attack, we must start the following python file $Pymoo\_patch.py$, either via an IDE (*Integrated Development Environment*) and/or a console via the required command *python Pymoo_patch.py*. Once launched, we will be prompted with multiples parameters necessary to be understand and set, as the attack details and efficiency will be altered regarding the model and the images attacked.

- Models types: Based on how the model have been trained, we must firstly specify the type of models attacked, limited to either **Keras** and/or **PyTorch**. This step is necessary as $AdversarialColorPatch$ requires to retrieve information data regarding the model input (*size*) and/or output (*number of classes*), as *Keras/PyTorch* must be read differently to access this data at running time. In our case, the model $Road\_Signs\_classifier\_model.h5$ have been trained on Keras, and therefore *Keras* is chosen has parameter.

- Images selection: We may now have the possibility to access the image to be attacked. If the setting up part have been correctly completed, we must be prompted with our image selected earlier, in our case the *Stop sign*. As multiples images may be used at the same time, each image can be selected via their corresponding numbers as displayed, and selecting the corresponding one.

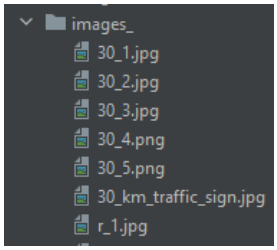


Figure 29. Adversarial Color Patch - Images Folder

Figure 30. Adversarial Color Patch - Models Types

- Model selection: Once the image selected, we may now have the possibility to access the model to be attacked. Identical to image selection, if the setting up part have been correctly completed, we must be prompted with our model. Once more, we may be prompted with multiples models at the same time, therefore each model may be selected via their corresponding numbers as displayed, and selecting the corresponding one.

- Attacks selection: The *Adversarial Color Patch* is providing three different adversarial attacks types based on the preferences needed.
  Indeed, we are provided the possibility to produce adversarial patches with a **target attack [number]** (*the adversarial patch produced target a specific class to be predicted by the model*), **untarget attack [u]** (*the adversarial patch produced tries to minimize the original label probability to be predicted by the model*) and **best attack [nc]** (*the adversarial patch produced maximize the prediction probability of the best class different from the original class*). For our example, we will provide all three attacks types to observe the differences and impact on the adversarial patches produced.

- Adversarial patch form: We may have the possibility to produce adversarial patches of different forms, respectively **circle**,**square** and/or **triangle**. We can choose the adversarial patch form based on our preferences and necessary requirements, that will be used during the attack. Once again, for the purpose of this example, we will provide an attack with each of the forms.

- Shape of the image object: Identical to the adversarial patch form step, we must as well choose the form of the image object attacked, respectively **circle**,**square** and/or **triangle**. This is important to be understood and completed, as it is required in order to efficiently produce the ACP attack. Certainly, the selected shape will be exploited during the color restriction and selection step of the application, used in order to retrieve all the pixels colors contained on the image in this shape, to finally produce the color domain later used for the adversarial patch production. Therefore, based on the shape chosen, the patch color will be adequately altered. For our case, the *Stop sign* being between a circle and square shape, in order to assure the efficiency of the attack, we applied the circle shape.

Figure 31. Adversarial Color Patch - Image Selection

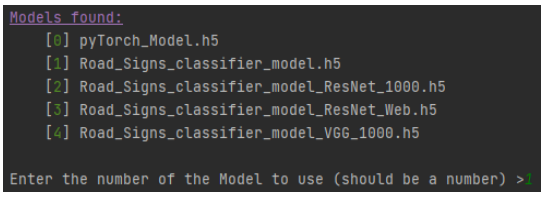


Figure 32. Adversarial Color Patch - Models Selection

- Amount of adversarial patches: Finally, for last parameter, we may define the amount of adversarial patches to be produced between one and ten. Actually, during the attack, one adversarial patch will bu default produce seven different adversarial patches, each representing different sizes based on a ratio division of the image attacked. However, we may if necessary, retrieve additional adversarial patches of identical size by defining in an amount parameter, which is by default of one.

We are now able to apply the $AdversarialColorPatch$ attack provided all the parameters which have been set. In order to observe the evolution of the attack, we are provided a progress timeline regarding the numbers of evolution and generations. Additionally, we are provided information regarding the attack efficiency, such as the original classes with their predicted probability, and the attacked classes with their predicted probability.

Note: *As stated in the installation section, Python version 3 must be used, and the required libraries must have been installed in order for ACP to operate adequately.*

Note: *The amount of time for an attack is based on the image size. The bigger it is, the longest the attack time and power will be required. This limitation will be addressed in the DISCUSSION section of the paper.*

Retrieve the results

We have just applied an attack with $AdversarialColorPatch$,

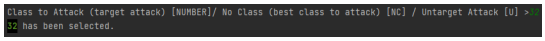


Figure 33. Adversarial Color Patch - Attacks selection

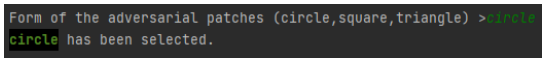


Figure 34. Adversarial Color Patch - Adversarial patch form

and it successfully been completed. We want now to retrieve the adversarial patches which have been produced, observe an example of its application on the attacked image, and obtain a summary of the attack efficiency.

Adversarial patches which have been produced by the ACP attack are located and saved in **patches** folder, with respective sub-folders regarding the form of the adversarial patch which have been defined **p_[form]**.
Each of the individual adversarial patch which have been produced, are provided information about their details in order to retrieve the respective attacked images and the possibility to apply the adversarial patches for a real application. (*i.e final_patch_12S_38X54Y_248A_1661256182T*). Similarly, in order to retrieve the adversarial patches applied on the attacked image, ACP is saving these in **p_images**, and information about the details of the applied adversarial patch on the image are provided (*i.e patch_image_12S_38X54Y_248A_1661256182T*).

- Size of the patch [S]
- Position of the patch on the image [X,Y]
- Angle of the patch on the image [A]
- Time in milliseconds the patch have been produced [T]

For our example regarding the *Stop Sign*, we are retrieving the images produced for the small and big adversarial patches.

Having managed to retrieve the adversarial patches produced by ACP, we may require to be aware of the efficiency of the attack, to observe the capabilities of the adversarial patches selected for a possible real application.
To achieve this, the application is saving the data located at the **Results** folder in *.csv* files, each being titled regarding the attack types and the form of the adversarial patches applied (*i.e Results_[form][attack].csv*). Each of the *.csv* file are providing a further detailed summary of the individual adversarial patches information (*Patch_Size, PositionX, PositionY, Rotation, Class_Attack, Probability, Class_Original, Final_Probability*) for the different sizes produced.
As we have detailed in the structure of the application, we are provided a python file, titled $result\_analysis.py$. The file has the purpose to provide a visual graph of the information data contained in each *.csv* files, regarding the prediction probability related to the size of the adversarial patch. The python file must be located in the same location as the *.csv* files and launched via an IDE (*Integrated Development Environment*) and/or a console via the required command *python result_analysis.py*.

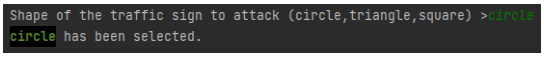


Figure 35. Adversarial Color Patch - Shape of the image object

Figure 36. Adversarial Color Patch - Amount of adversarial patches

8. RESULTS

This section is re-introducing the research questions defined during the introduction, detailing the results that we have obtained based on the use of the methods in the last section.

***R1** - How to restrict the visibility of the adversarial patches from the human factor?*

The amount of visibility for an adversarial patch is based on restricting the pixels colors (*channels values*) of this last, as close as the ones from the original image. To achieve this, we applied the color restriction method on a set of plain colors (*red,blue,pink,black,white*), followed by the evaluation on a "**Stop Sign**", and applied diverse threshold values, in the positive and negative direction, for x (*main color*) and y (*other channels*).
Based on the results obtained from this experiment, we determined that the ideal threshold for the parameter x is $[-50, +100]$, and for the parameter y is $[-30, +30]$. As x is related to the main color, a larger modification of the color channel is allowed without actively impacting the color domain. If the threshold is not respected, we can observe that the color domain is deviating from the original one, and introducing too much contrast between each of these.
An idea to obtain a perfect restriction of visibility on the color, would be to set the threshold to $[-1, +1]$ for all the channels. Nevertheless, we should keep in mind that the adversarial patch must provide efficient attack results, and a threshold with that restriction would impact the attack capabilities of the patch. Therefore, we have set the threshold in order to balance the performance between the visibility and the efficiency.

We are aware that a pixel value interpreted by any computer is extending from 0 to 255, therefore we need to consider the possibility that there exists extreme values such as $[30, 0, 0]$ (*black*) or $[255, 240, 348]$ (*white*). Indeed, on the black color, if we follow our threshold rule set, we obtain an entirely different color, due to the fact that the implementation performs the following operation $30 - 100 = -70 = 255 - 70 = 185$, and we obtain the red color. A similar issue is obtained with the color white. Therefore, we have defined two exceptions based on the extreme white color and black color, and manage to define the ideal extreme threshold as $[-50, +40]$ for each for the parameter x_E (*the parameter $y_E = y$ is unchanged*). Moreover, we restricted the extreme value to not go below 0 or over 255.

Direction/Parameters	x	y	x_E	y_E
Positive	100	30	40	30
Negative	50	30	50	30

TABLE 1. THRESHOLD SETTINGS FOR PARAMETERS x,y,x_E AND y_E

***R2** - Is producing adversarial patches of different forms and size*

Figure 37. Adversarial Color Patch - Results small adversarial patches

Figure 38. Adversarial Color Patch - Results big adversarial patches

delivering similar attack efficiency?

Associated to the design of the adversarial patch instead of the content (*pixels values*), this experiment is essential as there exists traffic signs with multitudes forms such as *circle, square, rectangle, triangle, octagon*, and the adversarial patch design is adequately participating in the transparency of the attack once applied onto the corresponding traffic sign form. To achieve this, we have applied three different form restriction onlooking the entire traffic signs types - ***square,circle,triangle*** - along diverse patches sizes, and retrieved the probabilities of the attack efficiency for target attack on two traffic signs: **Stop** and **Speed**.

Sign *"Speed"* [circle] - In $Table2$, we can observe the existence of an inconsistency due to the fact that in genetic algorithm, we may reach a local minima (*not the optimum result*) resulting in less efficient attack (*solution is to re-run the attack to obtain better attack probabilities*). Nevertheless, we can observe that based on the size of the adversarial patch, specific forms may be more efficient than others (*circle* $>$ *square in P1, circle* $<$ *square in P2, circle* $>$ *square in P3*). But, the triangle form probabilities is majoritarily lower in all phases, due that the amount of pixels modification is lowered, and consequently the attack capabilities is lowered as well.

Moreover, we surely observe that further the phases (*smaller adversarial patches*), lower is the probability and attack efficiency. This result is straightforward explained and is based on a similar principle as the form, since the attack is allowed fewer pixels modifications, the attack efficiency is consequently lowered.

Forms/Patches Sizes	33% (P1)	25% (P2)	20% (P3)
Circle	0.99	0.87	0.86
Square	0.99	0.97	0.84
Triangle	0.90	0.99	0.79

TABLE 2. RESULTS FOR FORMS EXPERIMENT WITH *Stop* SIGN

Sign *"Stop"* [octagon] - In $Table3$, we can observe that we obtained consistent results. Indeed, we can notice that the adversarial patch with the *circle* form is superior in every phase to *square* and *triangle*, and is explained due that the traffic sign form is corresponding to the form of the adversarial patch. In fact, we must be aware of every parameter that might influence the attack probabilities : **form of the traffic sign** (*hyper-parameter*), **color of the traffic sign** (*handled by the application*), **model trained** (*hyper-parameter*), **local or optimal minima** (*handled by the application*).

Forms/Patches Sizes	33% (P1)	25% (P2)	20% (P3)
Circle	0.69	0.11	0.16
Square	0.50	0.09	0.11
Triangle	0.075	0.11	0.14

TABLE 3. RESULTS FOR FORMS EXPERIMENT WITH *Speed* SIGN

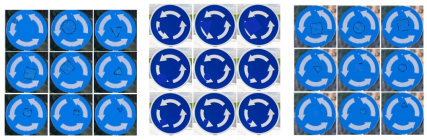


Figure 39. Adversarial Color Patch - Results obtained for "RoundAbout" Road Sign

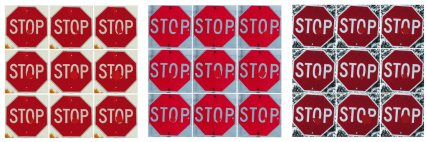


Figure 40. Adversarial Color Patch - Results obtained for "Stop" Road Signs

Figure 41. Adversarial Color Patch - Results obtained for "Speed 30km/h" Road Signs

R3 - *Can we apply Pymoo to produce adversarial patches fulfilling the restriction in visibility and attack position, but still being efficient?*

We have implemented an adversarial patch attack based on the application of the genetic algorithm from Pymoo, and altered it such that the adversarial patch is optimized in its content (*pixels*) and its position (*x,y,z*). In order to determine the genetic algorithm parameters to set, we analyzed the amount of time needed for the probability of an attack to reach over 95%, and similarly the efficiency of convergence of the adversarial patches into an optimal minima. After multiples experiments, the optimal settings for the genetic algorithm settled for 200 generations, with 256 of population (*adversarial patches*), and 256 mutations per generation.

In order to obtain flexible attacks, we have defined in our implementation multitudes hyper-parameters which once set allow

to provide improved attack efficiency, and therefore knowledge on the type of traffic sign attacked is needed.

Attack Parameters:

- Image Attacked (.jpg)
- Model Attacked (.h5)
- Type of Attack (*Target attack, Untarget Attack, Best Attack*)
- Adversarial Patch Form (*circle,square,triangle*)
- traffic Sign Form (*circle, square, triangle*)
- Amount of Adversarial Patches (*between* 1 *and* 10)

Forms/Patches Sizes	P1	P2	P3	P4	P5	P6	P7
Circle [T]	0.99	0.87	0.86	0.79	0.67	0.44	E
Square [T]	0.99	0.97	0.84	0.83	0.84	0.68	0.46
Triangle [T]	0.90	0.99	0.79	0.41	0.30	E	E
Circle [U]	0.98	0.99	0.99	0.9	0.99	0.99	0.99
Square [U]	1.0	0.99	0.99	1.0	0.99	0.99	0.99
Triangle [U]	0.99	0.99	1	1.0	0.99	0.99	0.99
Circle [B]	1.0	1.0	1.0	0.99	0.99	0.99	0.99
Square [B]	1.0	1.0	1.0	1.0	0.99	0.99	0.99
Triangle [B]	1.0	1.0	1.0	1.0	0.99	0.99	0.99

TABLE 4. RESULTS FOR *Target Attack, Untarget Attack, Best Attack* ON SPEED TRAFFIC SIGN USING PYMOO

Forms/Patches Sizes	P1	P2	P3	P4	P5	P6	P7
Circle [T]	0.69	E	E	E	E	E	E
Square [T]	0.50	E	E	E	E	E	E
Triangle [T]	E	E	E	E	E	E	E
Circle [U]	0.98	0.98	0.99	0.8	0.66	0.93	0.66
Square [U]	0.99	0.99	0.99	0.93	0.82	0.91	0.91
Triangle [U]	0.99	0.99	1	1.0	0.99	0.99	0.99
Circle [B]	1.0	0.99	0.99	0.99	0.99	0.99	0.96
Square [B]	1.0	0.99	0.99	0.99	0.99	0.99	0.99
Triangle [B]	1.0	1.0	1.0	1.0	0.99	0.99	0.99

TABLE 5. RESULTS FOR *Target Attack, Untarget Attack, Best Attack* ON STOP TRAFFIC SIGN USING PYMOO

Note: *E represents the values in the attack which do not manage to reach the target label.*

9. DISCUSSION

In this section, we are comparing the results obtained regarding the metrics defined earlier in our research, helping to determine the efficiency of the **ACP** algorithm and application usability. Furthermore, we are analyzing the **AdversarialColorPatch** compared to the operation of three existing adversarial patches attacks defined and enumerated earlier in our state of the art, respectively *AdversarialPatch*, *SaliencyMap* and *CarliniMap*. Later on, the limitations of the current **ACP** attack implemented will be discussed, enumerating the restrictions, and possible improvements to be achieved. This section will be closed with an explanation of the future work regarding the current project progression.

9.1. Results and Metrics

As we may observe in both tables $Table4$ and $Table5$ for ***R3*** in the result section, we have obtained straightforward results allowing to define more understanding of the **AdversarialColorPatch** efficiency. Indeed, at first sight we can see the presence of a difference in the success probability of an adversarial patch for the type of traffic sign applied *Stop sign* or *Speed Sign*, specifically the types of attack which have been applied. We can notice that the attacks, *Untarget attack* and *Best attack*, have produced excellent probabilities of attack success for both traffic signs, due to the fact that these attacks intent to precisely go toward the direction of the ideal solution, as long as being far from the original starting point (*original label of traffic sign*) . Nevertheless, on the opposite, the *Target attack* is restricting the direction possibilities to the ideal solution, as it must reach a specific label to be attacked. Consequently, the direction for the best solution is fewer straightforward, prone to extra search through the objective function domain, and additionally the possibility to end up in a local minima. Indeed, the *E* in both tables are representing values which did not manage to obtain a specific label (*target label for Target attack; not original label for Untarget attack and Best attack*), and is mostly present in the *Target attack* part of the **AdversarialColorPatch**. Another observation, is related to the size of the adversarial patch applied. Indeed, the patch size is only covering a percentage of the attacked image $(33\%, 25\%, 20\%, 15\%, 13\%, 10\%)$, therefore the amount of pixels participating in the attack is each time lowered, with the consequence of lowering the number of allowed pixels modification on the image and impacting the probability of success for the attack. This outcome may well be observed for *Target Attack* applying a Square adversarial patch for the *Speed Sign*, where the probability of success is lowered each time the adversarial patch is downsized.

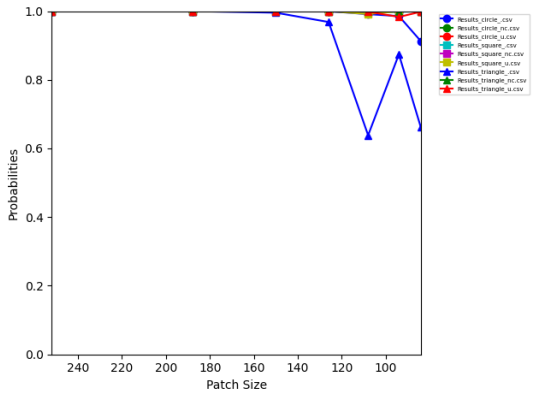


Figure 42. Adversarial Color Patch - Graph obtained for .csv files of "Speed 30km/h"

Based on the tables results, we can define a global metric defining the attack accuracy and success rate for an image applying the **AdversarialColorPatch**. Indeed, for *Untarget attack* and *Best attack*, we can define two success metrics: the success modification based on the amount of adversarial patches which managed to shift from the original label to an attacked label (*U1*); and the success rate of attack based on the probability of the attacked label to be above 95% (*U2*). For the *Target attack*, we may define three success metrics: the success modification based on the amount of adversarial patches which successfully switched the prediction from the original label to the targeted label (*T1*); the minimum success rate of attack based on the probability that the targeted label is at least 50% related to the original label (*T2*); the success rate of attack based on the probability of the targeted

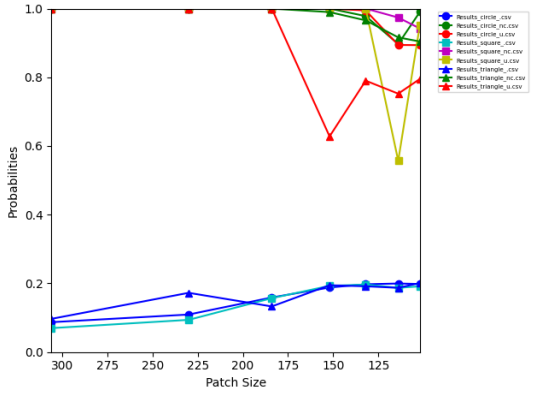


Figure 43. Adversarial Color Patch - Graph obtained for .csv files of "Stop"

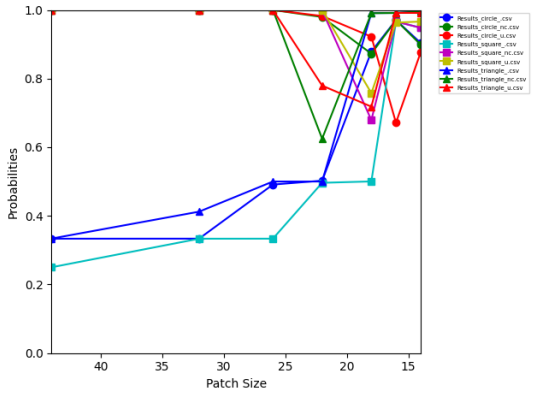


Figure 44. Adversarial Color Patch - Graph obtained for .csv files of "Roundabout"

label to be above 95% (*T3*). These metrics are providing a global success evaluation of the **AdversarialColorPatch** capabilities and efficiency applying different types of attacks and adversarial patches sizes.

***Note**: Each of the metrics $U1, U2, T1, T2$ and $T3$ have been evaluated for each of the different adversarial patches forms. Respectively $Square$, $Circle$ and $Triangle$.*

Success Metric/traffic Sign	Speed Sign	Stop Sign
U1	100%	100%
U2	100%	100%
T1	100% - 85.7% - 71.4%	14% - 14% - 0%
T2	85.7% - 83.3% - 60%	14% - 14% - 0%
T3	28.5% - 16% - 20%	0% - 0% - 0%

TABLE 6. RESULTS METRICS FOR *Target Attack, Untarget Attack, Best Attack* ON TRAFFIC SIGNS TABLES 4 AND TABLES 5

If we observe the results of $Table6$, we can notice a significant gap of performance between both traffic signs "Speed" and "Stop".

Indeed, for $T1$, $T2$ and $T3$ related to *Target attack*; for the attack on the *Speed Sign*, some of the adversarial patches created by **AdversarialColorPatch** are capable to provide satisfying probabilities, as we can notice for $T1$ and $T2$ where most of the adversarial patches are acceptable to be applied, defining the possibility of a successful attack to occurs if applied on the related traffic sign. However, $T3$ is providing quite poor results, as it shows awareness that only one or two adversarial patches may be applied against the related traffic sign.
Furthermore, for the *Stop Sign* only few, nay none, of the adversarial patches created by **AdversarialColorPatch** are capable to provide satisfying probabilities, as we can notice for $T1$ and $T2$, especially $T3$ with null results, therefore performing poorly.
For $U1$ and $U2$ related to *Untarget attack*, we can distinctively observe excellent performance from the **AdversarialColorPatch** to create adversarial patches for both the *Speed Sign* and *traffic Sign*, as it is non restricted on the targeted label, and the probability is reaching a hundred percent for all.

Consequently, we may assert that the **AdversarialColorPatch** is exceptionally performing for *Untarget Attack*, while intermediately and poorly for *Untarget Attack*.

Similarity metrics for visibility evaluation

We have just evaluated the metrics related to the *AdversarialColorPatch* attack accuracy and time to process a single adversarial patch. Nevertheless, a last metric to be observed is defined by the calculation of the similarity indexes. Indeed, as reminder, the objective of this research is to produce adversarial patches with color domain restriction, in order for the transparency towards the human visibility to be maximized. Therefore, an evaluation of the structural similarity between an original road sign, and an attacked version with an adversarial patch applied to the image, must be provided. Among these, we are evaluating four similarities metrics that have been introduced in Metrics of Evaluation, based on different similarity algorithms, respectively **SSIM**, **SCC**, **VIFP** and **UQI**. The usage of multiples similarity metrics allow us to obtain an efficient average comparison of the transparency efficiency of a specific adversarial patch with an attacked image.

For each of the similarity metrics, the calculation is providing a defined value according to how two evaluated images are similar. Indeed, the values are restricted between the two extremes *0.0* and *1.0*, such that *1.0* refers to two identical images (*can be obtained by evaluating twice the same image*), and *0.0* referring to two totally different images. For our evaluation of an adversarial patch transparency efficiency, the value must be higher than *0.95* for all similarity metrics in order to be defined as a successful attack fulfilling the visibility requirement. Finally, we are evaluating the similarity metrics for two adversarial patches (*smallest and largest*) produced with three different road signs (*stop sign, roundabout, speed*) to observe a possible difference regarding the visibility based by the size of the adversarial patches applied.

$$M \in \{\texttt{SSIM}, \texttt{SCC}, \texttt{VIFP}, \texttt{UQI}\}$$
$$x \in \texttt{image}$$
$$x' \in \texttt{image attacked}$$
$$M(x, x') >= 0.95 \Rightarrow \texttt{SUCCESS}$$
$$M(x, x') < 0.95 \Rightarrow \texttt{FAIL}$$

***Note:** The similarity metrics evaluation have been produced and observed via the usage of SSIM_Metric.py found in the*

AdversarialColorPatch application.

Figure 45. Adversarial Color Patch - Similarity metric for *Stop Sign*

For the *Stop Sign*, we have obtained for each of the similarity metrics, the following values:

- Small adversarial patch [*SSIM: 0.99, SCC: 0.98, UQI: 0.99, VIFP: 0.98*]
- Large adversarial patch [*SSIM: 0.90, SCC: 0.88, UQI: 0.95, VIFP: 0.87*].

Figure 46. Adversarial Color Patch - Similarity metric for *Roundabout Sign*

For the *Roundabout*, we have obtained for each of the similarity metrics, the following values:

- Small adversarial patch [*SSIM: 0.98, SCC: 0.97, UQI: 0.99, VIFP: 0.97*]
- Large adversarial patch [*SSIM: 0.91, SCC: 0.88, UQI: 0.96, VIFP: 0.93*].

For the *Speed Sign (white)*, we have obtained for each of the similarity metrics, the following values:

- Small adversarial patch [*SSIM: 0.99, SCC: 0.98, UQI: 0.99, VIFP: 0.96*]
- Large adversarial patch [*SSIM: 0.89, SCC: 0.89, UQI: 0.97, VIFP: 0.84*].

For the *Speed Sign (black)*, we have obtained for each of the similarity metrics, the following values:

Figure 47. Adversarial Color Patch - Similarity metric for *Speed Sign*

Figure 48. Adversarial Color Patch - Similarity metric for *Speed Sign*

- Small adversarial patch [*SSIM: 0.98, SCC: 0.95, UQI: 0.98, VIFP: 0.98*]
- Large adversarial patch [*SSIM: 0.92, SCC: 0.87, UQI: 0.90, VIFP: 0.86*].

Regarding the results obtained, we can observe that for each of the road signs, the evaluation of the similarity metrics are consistently successful (*according to our requirements*) with an attack applying a small adversarial patch. While, on the contrary, the similarity metrics are consistently failing (*according to our requirements*) with an attack applying a large adversarial patch. These outcomes are explained, as opposed to an attacked efficiency based on the adversarial patches sizes. Certainly, we have already detailed earlier in this research that an attack efficiency may be related to the size of the adversarial patch applied to the image, as if amount of allowed pixels to be modified is larger, the efficiency of the attack will be relatively higher and vice-versa. However, consequently the transparency of the adversarial patch is lowered, as additional pixels produce less similarity between the original image and the attacked image. On the contrary, if the amount of allowed pixels to be modified is lowered, the efficiency of the attack will be relatively lowered and possibly less successful. However, consequently the transparency of the adversarial patch is increased, as less pixels produce more similarity between the original image and the attacked image.

9.2. Comparison with State of the Art

In order to obtain a comparison of the **ACP** attack efficiency and usability, we have operated with three well known adversarial patches attacks approaching the operation of our *ACP* attack : *AdversarialPatch[AP]*, *SaliencyMap[SM]* and *CarliniMethod[CM]*. These attacks are allowing us to observe the possible assets and disadvantage of each individual attack, and define an accurate efficiency map for the **AdversarialColorPatch**. By all means, for each of the attacks, we are applying a set of traffics signs (*stop, roundabout, speed*) and analyzed the obtained results based on three major attributes : *time* - necessary time to apply the attack, *attack efficiency* - rate of success of the attack based on if the prediction have been altered, and *patch visibility*.

The results obtained are defined in $Table7$ (*time*), $Figure49$ (*time*) and $Table8$ (*efficiency*).

Traffic Sign/Patch Attack	AP	SM	CM
Stop	2.1 (s)	61.2 (s)	0.53 (s)
RoundAbout	2.08 (s)	79.77 (s)	1.86 (s)
Speed	2.01 (s)	32.38 (s)	1.86 (s)

TABLE 7. RESULTS FOR ATTACKS COMPARISON BASED ON AVERAGE *Time*

Traffic Sign/Patch Attack	AP	SM	CM
Stop	(0.2,0.3,0.4,0.5)	SUCCESS	FAIL
RoundAbout	(0.3,0.4,0.5)	SUCCESS	FAIL
Speed	(0.4,0.5)	SUCCESS	FAIL

TABLE 8. RESULTS FOR ATTACKS COMPARISON BASED ON AVERAGE *Attack Efficiency*

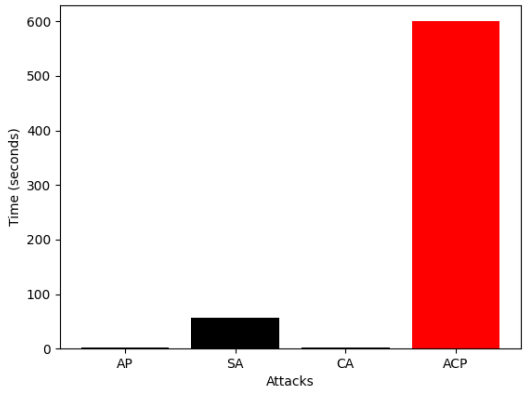


Figure 49. Comparison of time consumption for attacks with *AP,SA,CA,ACP*

9.2.1. AdversarialPatch[AP].

We began with the *AdversarialPatch* attack. As explained, we have applied the attack on a set of traffic signs (*stop, roundabout, speed*) and observed the output received for each individual runs. For this attack, we were able to define the size of the adversarial patches to be used, allowing us to apply patches with different ratio representing *10%*(*small*), *30%*(*medium*) and *50%*(*big*) of the image. However, on the other hand, we were unable to define the forms of the patches, singularly only allowing the circle form to be applied.

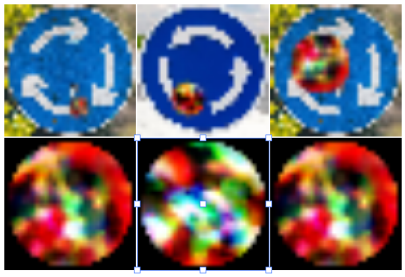


Figure 50. Examples of AdversarialPatch attack on "Roundabout" sign

Figure 51. Examples of AdversarialPatch attack on "Stop" sign

Figure 52. Examples of AdversarialPatch attack on "Speed" sign

Early, we can observe with the results obtained in $Table7$ and $Figure49$, that the *AP* attack time (*3 seconds*) is substantially better than for *ACP* (*10 minute*), thus likely two-hundreds times faster to operate a single attack. This advantage is relatively significant as it allows sequentially attacks to be achieved in a short amount of time.

Nevertheless, we may observe with $Figure50 - 52$ that vast amount of the adversarial patches produced have a great tendency to be represented as an association set of flashy colors. This issue may easily be observed due to the difference of context color between the patch and the attacked image, allowing a straightforward detection of the presence of an adversarial patch in the image. Throughout the ACP implementation, one of the main objective, that has been achieved, was to take into

consideration this detection issue of the patch, and produce adversarial patches sensitive to the attacked image color context, consequently allowing the visibility of the patches to be drastically reduced.

Finally, we may observe $Table8$ that the attack efficiency of the AP may be defined as acceptable, as it allows to successfully produce an attack for each of the traffic sign at least once in a run, applying patches of medium and big sizes. However, we may compare and observe with $Table4$ and $Table5$ that the attack efficiency for ACP allows to essentially produce a successful attack with an extended amount of different sizes for the adversarial patches, consequently improving the attack efficiency compared to the AP.

9.2.2. SaliencyMap[SM].

We pursued with the *SaliencyMap* attack. Once again, we have applied the attack on a set of traffic signs (*stop, roundabout, speed*) and observed the output received for each individual runs. For this attack, we were this time not able to define the size of the adversarial patches to be used, neither to define the forms of the patches.

Figure 53. Examples of SaliencyMap attack on traffic signs

Early, we can observe with the results obtained in $Table7$ and $Figure49$, that the *SM* attack time (*1 minute*), similar to the *AP*, is substantially better than for *ACP* (*10 minute*), thus likely ten times faster to operate a single attack. This advantage is relatively significant as it allows sequentially attacks to be achieved in a short amount of time.

Nevertheless, we may anew observe with $Figure53$ that vast amount of the adversarial patches produced have a great tendency to be represented as an association of colors, located and noticeable via a set of multiples tiny points representing the full patch. This issue may easily be observed due to; firstly the difference of context color between the patch and the attacked image, allowing a straightforward detection of the presence of an adversarial patch in the image; and second the disposition and the distance between each of the points produced, drastically decreasing the patch manageability. Throughout the ACP implementation, one of the main objective, that has been achieved, was to take into consideration the flexibility and applicability of the adversarial patch, by producing adversarial patches based on a solid approach (*one single solid patch*), consequently allowing the patches to be easily manipulated and applicable in a real environment.

Finally, we may observe $Table8$ that the attack efficiency of the SM may be defined as acceptable, as it allows to successfully produce an attack for each of the traffic sign. However, we may compare and observe once again with $Table4$ and $Table5$ that the attack efficiency for ACP allows to essentially produce a successful attack with an extended amount of different sizes for the adversarial patches while operating with a compact adversarial patch (*all attacks points are together*), consequently improving the attack efficiency, expend-ability and usability compared to the CM.

9.2.3. CarliniMethod[CM].

We concluded with the *CarliniMethod* attack. Once again, we have applied the attack on a set of traffic signs (*stop, roundabout, speed*) and observed the output received for each individual runs. Once again, we were not able to define the size of the adversarial patches to be used, neither to define the forms of the patches.

Figure 54. Example of CarliniMethod attack on "Stop" traffic sign

Early, we may observe with the results obtained in $Table7$ and $Figure49$, that the *CM* attack time (*2 seconds*), is substantially better than for *ACP* (*10 minute*), along with each attacks considered earlier. This advantage is relatively significant as it allows sequentially attacks to be achieved in a short amount of time.

Nevertheless, surprisingly, based on each of the runs for the *CM* attack, we may observe that at no time it managed to successfully provide a functioning patch to be applicable on the traffic sign. Indeed, observable with $Table8$ and $Figure54$, the patch is surely ideally unnoticeable, nonetheless it is lacking the efficiency to be able to alter the prediction of the traffic sign, as it is constantly failing. Notwithstanding, with ACP, the main objective achieved was to implement an attack capable to produce adversarial patches ideally unnoticeable, as much as being able to efficiently alter the prediction of the attacked image.

9.3. Limitations

The **AdversarialColorPatch** current implementation and efficiency, is restricted by a few limitations that have been observed during the experiments. In this section, we are enumerating the most impacting limitations discovered, detailing their causes and consequences.

The first limitation is the efficiency of the *Target attack* with **AdversarialColorPatch**. Indeed, we have observed in the last section a negligence for the capability of the attack to provide adversarial patches with the expected requirement (*target label*). After a few experiments, it seems to be due to the type of image attacked compared to the chosen targeted label; as we have seen, *Speed traffic* obtained correct few adversarial patches applicable, while the *Stop Sign* did not at all. However, *Untarget attack*

was constantly successful for both traffic signs, and the simple difference was related to the attacked label found, therefore when applying the *Target attack* we must be conscientious of the target label chosen. Indeed, if the attacked image is a *Stop Sign* (*octogon, red, letters*) and we wish to target a label to a *Pedestrian Sign* (*triangle, blue, stickman*), the attack will likely fail due to a great difference between both of the signs. This limitation is quite critical as it does not allow us to freely choose the target label.

The following limitation is related to the method applied to tackle the issue of defining the color threshold, to be applicable on an adversarial patch and the attacked image color domain. Notwithstanding that **AdversarialColorPatch** may be applicable to any Keras models, and attack images not related to traffic signs, the objective of this research was to essentially define a universal threshold for visibility limitation based on the color domain of an image. In order to simplify this process, the efficiency of the algorithm is based on a specific placing of the image attacked, additionally to a restriction of the form (*square shape frame*), this approach allows us to focus on the desired attacked object (*i.e traffic sign*), as it is situated in the middle of the image. Moreover, this research evolving around traffic signs, we restricted the type of forms to *circle*,*square* and *triangle* for the adversarial patches, related to the different types of traffics signs we can encounter on public roads. This limitation is fairly critical as, based on the evolution of these traffic signs forms (*rectangle*,*multiples signs*) and/or attacks with other models, the flexibility of the adversarial patches would require further improvement and modification availability to fulfill the attacked object contour.

The next limitation is related to the time required for **AdversarialColorPatch** to produce a single adversarial patche and/or a set of different adversarial patches sizes. Indeed, during the comparison of our attack with SquareAttack, AdversarialPatch and AdversarialTexture, we have observed that our attack required between 10 and 30 minutes to create a single adversarial patch (*based on the size of the attacked image*) for an image. This time is relative to the size of the image attacked, as the bigger it is, the more pixels modifications on the adversarial patch and research area domain is required, consequently elongating the process time. This critical feature of **AdversarialColorPatch** is due to the fact we are evolving in the black-box attacks domain, and applying a random search using a compact adversarial patch (*all the modified pixels are together*), therefore the convergence towards the objective function optimum value may require some time to find the optimal route, as adversarial patches modification may or may not improve the attack efficiency at each generation. Additionally, in order to maximize the attack efficiency and robustness, the amount of generation (*i.e* 200) is set to relatively high, and keep operating despite the fact we may already have reached an ideal probability of attack. This method allows to minimize the impact of the original label on the image.

9.4. Future Work

Based on the limitations that have been detailed in the last section, and the current progress of the application *Adversarial Color Patch* attack, we are determining in this section the possible future work to be achieved, in order to either improve the attack efficiency and/or usability for a real-world application.

As observed in limitations, we have defined that the *untarget attack* was behaving efficiently on the contrary to the *target attack*, due to the type of road signs, adversarial patch color and/or the size of the adversarial patch applied, resulting in distinct efficiency. As the main objective of this research was to introduce an alternative approach to generate adversarial patches, based on the color domain restriction of the attacked image, we have focused on first defining the restriction, defining the genetic algorithm and observe the results obtained. Therefore, one of the future work to accomplish is to provide an efficient target attack, regardless of the adversarial patches sizes. As explained earlier, the current adversarial patches attacks produced patches with colors distant from the color domain of the attacked image; with our method, the patches color is close to the original color, hardening the attack efficiency (*pixels modification restriction*), exceptionally when the adversarial patch is small. Consequently, in order to solve this issue, additional observation of the attack capabilities by tuning of the generation algorithm parameters might be introduced.

Nevertheless, in order to be able to tune the genetic algorithm, applying further generations and/or population, one of the issue enumerated that has been detailed in the limitations was the time. Indeed, based on the size of the attacked image, the time to execute a complete attack is greatly extended. Consequently, it did not allow the experiments to operate under extended genetic algorithm parameters, impacting the attack efficiency.

Finally, at last but not least, one of the future work to be accomplished is related to the use of the *AdversarialColorPatch* in a real scenario. Indeed, during this research, we have not introduced the usability of the *AdversarialColorPatch* attack for a self-driving car scenario, and simply provided metrics related to the attack efficiency, adversarial patch color visibility percentage and time to operate a full attack. Consequently, the next step of this research is the introduction of a method to observe the attack efficiency in a defined scenario. Among the potential methods, the usage of a simulator and/or apply the adversarial patches in real-life would allow to define the usability efficiency of the generated adversarial patches, passing on from the digital world to the real world.

10. CONCLUSION

Autonomous vehicles are nowadays subject to numbered of attacks, which are prone to have a dangerous impact, as it could lead to threat the human's life due to a miss-usability of the vehicle. We have observed that, among these attacks, adversarial patches were possible to be applied in order to fool an autonomous vehicle. Adversarial patches, may be defined as an image/pattern applied (*as a glue would do*) to an original image to produce an attacked image. The objective is to manipulate an Artificial Intelligence model, responsible for image recognition, to miss-predict the class of the original image. Indeed, autonomous vehicles are equipped with multiples sensors (*lidar, gps, radar, camera*) to help the autonomous driving to behave efficiently, where the image recognition is mainly responsible to predict the road signs, in order to be able to respect the road rules.

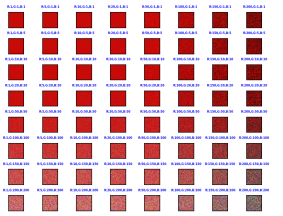
For this research, we have decided to implement an original attack, titled *AdversarialColorPatch*, based on the use of the

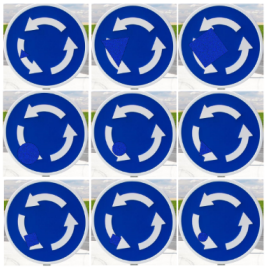
framework *Pymoo*, to produce adversarial patches with specific requirements, and attack road signs. We have observed that recent attacks related to adversarial patches conferred issues regarding the transparency of the patch and/or its location on the attacked image. Certainly, the adversarial patches visibility is a requirement to take into consideration, as the human factor may greatly impact the outcome. A human, more logical than an Artificial Intelligence blindly processing an operation, may have the possibly to detect an attack occurring, by observing the road situation, and noticing the existence of an adversarial patch due to a greater sensitivity of the eyes to color contrast. From this fact, *AdversarialColorPatch* is providing adversarial patches respecting the color domain of the attacked image (*similar color*, while defining a specific restriction of the color to be able to produce an efficient attack.

We have observed that the *AdversarialColorPatch* have produced curious results regarding the possibility to apply adversarial patches with color restriction, and successfully fulfilled the requirements of this research. Despite the fact that the time metric to produce an single attack is poor, *ACP* is providing efficient results during the production of an adversarial patch for *untarget* attacks and best attacks (*model is predicting any road sign, different from the original sign*), while behaving averagely for *target* attacks (*model is predicting any road sign defined by the user, different from the original sign*).
Based on these results, we have therefore defined that it would further be required to apply the attack in a real case scenario and observe the outcomes using *AdversarialColorPatch* as we only evolved in the computer/theoretical application of the attack. Additionally, an improvement of the target attacks efficiency, by re-evaluating the genetic algorithm handled by Pymoo, would need to be applied to provide a full efficient *ACP* attack.

Special thanks to Salah GHAMIZI for helping me along the production of the Master Thesis, and Yves LE TRAON for authorizing to work among the Security and Trust (SnT) members during the research period.

APPENDIX





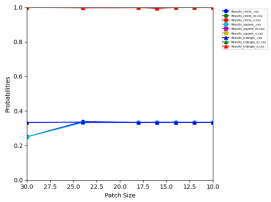


Figure 55. Adversarial Color Patch - Graph results for *Roundabout Sign* 1

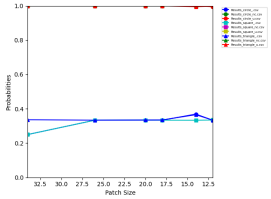


Figure 56. Adversarial Color Patch - Graph results for *Roundabout Sign* 2

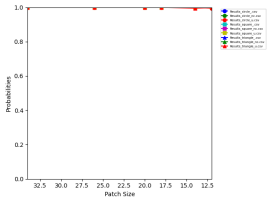


Figure 57. Adversarial Color Patch - Graph results for *Roundabout Sign* 3

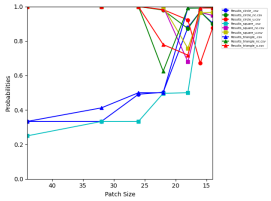


Figure 58. Adversarial Color Patch - Graph results for *Roundabout Sign* 4

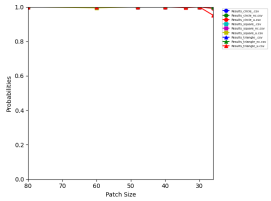


Figure 59. Adversarial Color Patch - Graph results for *Speed Sign* 1

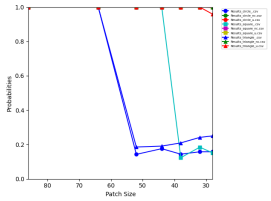


Figure 60. Adversarial Color Patch - Graph results for *Speed Sign* 2

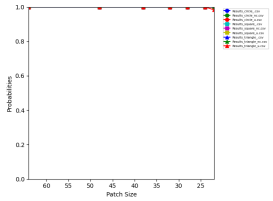


Figure 61. Adversarial Color Patch - Graph results for *Speed Sign* 3

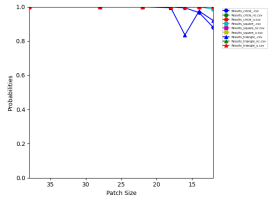


Figure 62. Adversarial Color Patch - Graph results for *Speed Sign* 4

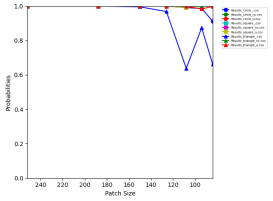


Figure 63. Adversarial Color Patch - Graph results for *Speed Sign* 5

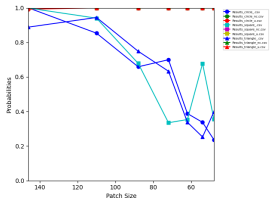


Figure 64. Adversarial Color Patch - Graph results for *Speed Sign* 6

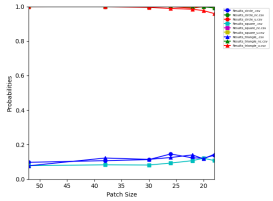


Figure 65. Adversarial Color Patch - Graph results for *Stop Sign* 1

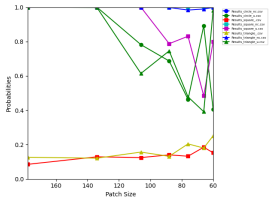


Figure 66. Adversarial Color Patch - Graph results for *Stop Sign* 2

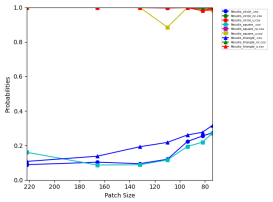


Figure 67. Adversarial Color Patch - Graph results for *Stop Sign* 3

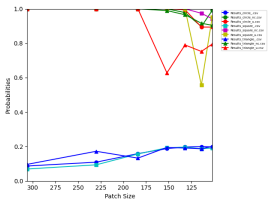


Figure 68. Adversarial Color Patch - Graph results for *Stop Sign* 4

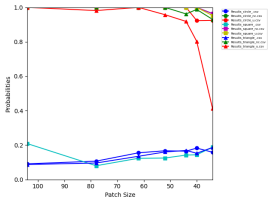


Figure 69. Adversarial Color Patch - Graph results for *Stop Sign* 5

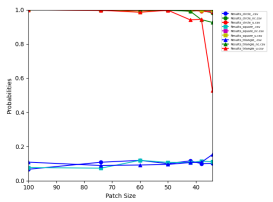


Figure 70. Adversarial Color Patch - Graph results for *Stop Sign* 6

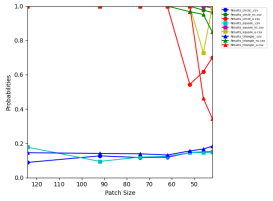


Figure 71. Adversarial Color Patch - Graph results for *Stop Sign* 7

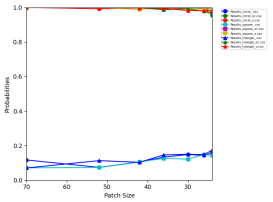


Figure 72. Adversarial Color Patch - Graph results for *Stop Sign* 8

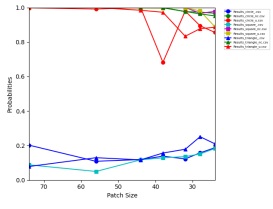


Figure 73. Adversarial Color Patch - Graph results for *Stop Sign* 9

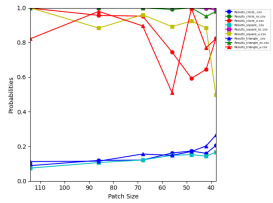
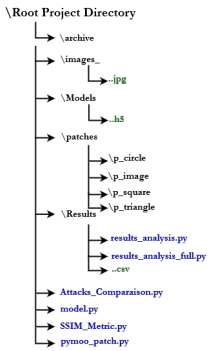
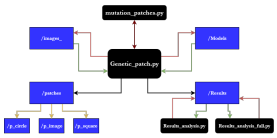
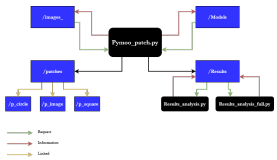
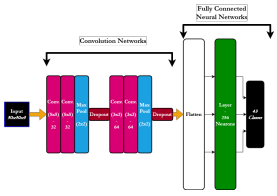


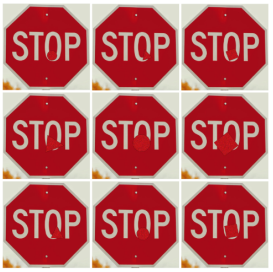
Figure 74. Adversarial Color Patch - Graph results for *Stop Sign* 10

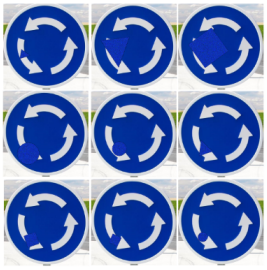


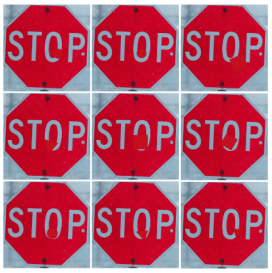












References

[1] Autonomous Vehicles: Levels, Technologies, Impacts and Concerns. https://www.ripublication.com/ijaer18/ijaerv13n16_42.pdf.

[2] Autonomous Vehicles Market Size, Share, Trends Analysis Report By Application. https://www.grandviewresearch.com/industry-analysis/autonomous-vehicles-market.

[3] Autonomous Vehicles Market Size, Share, Trends Analysis Report By Application. https://www.grandviewresearch.com/industry-analysis/autonomous-vehicles-market.

[4] Carla Simulator. https://carla.org/.

[5] Pymoo. https://Pymoo.org/.

[6] Python. https://www.python.org/downloads/.

[7] Python Project on traffic Signs Recognition with 95% Accuracy using CNN Keras. https://data-flair.training/blogs/python-project-traffic-signs-recognition/.

[8] Tensorflow. https://www.Tensorflow.org/.

[9] CNN Architectures from Scratch. https://medium.datadriveninvestor.com/cnn-architectures-from-scratch-c04d66ac20c2, 2020.

[10] Build ResNet from Scratch With Python. https://www.analyticsvidhya.com/blog/2021/06/build-resnet-from-scratch-with-python/, 2021.

[11] Residual Networks (ResNet) – Deep Learning. https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/, 2022.

[12] Alexis Myrick Alison Jenkins, Vinika Gupta and Mary Lenoir. Variations of Genetic Algorithms. https://arxiv.org/pdf/1911.00490.pdf, 2019.

[13] Koninika Patil Hamed Pirsiavash Aniruddha Saha, Akshayvarun Subramanya. Role of Spatial Context in Adversarial Robustness for Object Detection. https://arxiv.org/pdf/1910.00068.pdf, 2020.

[14] Fabio Roli Battista Biggio. Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning. https://arxiv.org/abs/1712.03141, 2017.

[15] David W. Corne and Michael A. Lones. Evolutionary Algorithms. https://arxiv.org/ftp/arxiv/papers/1805/1805.11014.pdf, X.

[16] Naman D. Singh Nicolas Flammarion Matthias Hein Francesco Croce, Maksym Andriushchenko. Sparse-RS: a Versatile Framework for Query-Efficient Sparse Black-Box Adversarial Attacks. https://arxiv.org/pdf/2006.12834.pdf, 2022.

[17] Zhan Qin Xue Liu Kui Ren, Tianhang Zheng. Adversarial Attacks and Defenses in Deep Learning. https://www.researchgate.net/publication/338408111_Adversarial_Attacks_and_Defenses_in_Deep_Learning, 2019.

[18] Nicolas Flammarion Matthias Hein Maksym Andriushchenko, Francesco Croce. Square Attack: a query-efficient black-box adversarial attack via random search. https://arxiv.org/pdf/1912.00049.pdf, 2020.

[19] Navid Kardan Naveed Akhtar, Ajmal Mian and Mubarak Shah. Advances in adversarial attacks and defenses in computer vision: A survey. https://arxiv.org/pdf/2108.00401.pdf, 2021.

[20] WANG Bin PAN Jun WANG Jia-Min CHEN Jian-Hai ZHOU Wu-Jie LEI Jing-Sheng QIAN Ya-guan, MA Dan-feng. Spot Evasion Attacks: Adversarial Examples for License Plate Recognition Systems with Convolutional Neural Networks. https://arxiv.org/ftp/arxiv/papers/1911/1911.00927.pdf, 2019.

[21] Ninghao Liu Fan Yang Xia Hu Ruixiang Tang, Mengnan Du. An Embarrassingly Simple Approach for Trojan Attack in Deep Neural Networks. https://arxiv.org/pdf/2006.08131.pdf, 2020.

[22] Xiaolin Li Dapeng Wu Xiaoyong Yuan, Pan He. Adaptive Adversarial Attack on Scene Text Recognition. https://arxiv.org/pdf/1807.03326.pdf, 2020.