Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Statistics & Machine Learning
2025 | LIU-IDA/STAT-A–25/002–SE

Investigating the Robustness of Visual Localization method Accelerated Coordinate Encoding under Fast Gradient Sign Methods Adversarial Attacks

Alan Cacique Tamariz

Supervisor : PhD Candidate Johan Edstedt, CVL, LiU
Examiner : PhD Johan Alenlöv, STIMA, LiU

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

Computer vision combines math and computer science to help computers understand images and perform tasks such as Image classification, Object detection, Visual Tracking, Semantic Segmentation, and Image Restoration [1]. Many software applications today use computer vision to perform these tasks. However, like any software application, computer vision systems are vulnerable to adversarial attacks. Adversarial attacks are deceiving a machine learning model, causing it to fail in its prediction by adding some imperceptible perturbation to the original image.

Several research papers analyze the adversarial robustness of Image classification [2, 3] and Object detection[4] models when in the presence of adversarial attacks. However, there is a lack of research on adversarial attacks in visual localization. This study aims to address this critical gap by investigating the adversarial robustness of visual localization models.

Visual localization is predicting the camera pose from a query image using some type of space representation, such as an image (database) and 3D point cloud, among others [5, 6]. There are three main ways to solve the task of visual localization. The model used in this study is Accelerated Coordinate Encoding (ACE), a learning-based model that functions by employing a scene coordinate regression system. Its architecture includes the first CNN layers, which extract high-dimensional feature vectors and have on top a Scene-specific Regression MLP (Multi-layer perceptron) that predicts the scene coordinates based on the feature vectors. Then, the 3D coordinates go into the solver g, conformed by a PnP and a RANSAC loop that estimates the pose.

The adversarial attacks in this study include a combination of white-box and black-box strategies. The adversarial perturbation was computed using the white-box method FGSM and its iterative version, I-FGSM. However, the projection of the loss function was not backpropagated to the entire ACE architecture. Instead, the 3D coordinates from the MLP head were utilized to compute two distinct loss functions $\mathcal{L}_1$ and $\mathcal{L}_2$. This second part represents a BPDA black-box attack.

To assess the robustness of the visual localization model under the adversarial attacks generated by FGSM and I-FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions, different values of the hyperparameters ϵ for FGSM and ϵ^I and N for I-FGSM were applied. The model was tested on the 7-scenes dataset from Microsoft on the Chess scene.

After testing the model with the adversarial examples, the average error pose was obtained, which refers to the mean error between the rotational and translational error. This was analyzed against the different ϵ values for FGSM and $\{\epsilon^I, N\}$ for I-FGSM. The results showed that the distribution for attack cases was skewed and more varied, while the distribution was centered and tight for no-attack cases.

Additionally, it was found that for all four types of adversarial attacks, there was a positive relationship that closely followed a polynomial trend—being the FGSM with $\mathcal{L}_1$ loss function, the model with the highest adjusted R^2 value of 0.71.

Another key finding was that the iteration step number N on the I-FGSM model does not affect the average error value. And that regardless of the method, the rotational elements for $\mathcal{L}_1$ and the rotational elements and the translational x component for $\mathcal{L}_2$ were the components of the pose with the most influence on the average error.

Acknowledgments

I want to express my gratitude to my supervisor, Johan Edstedt, for giving me the opportunity to work with the CVL despite my different academic background and for his guidance and patience. It showed me the correct research direction and allowed me to understand that my way can be within computer vision.

I want to thank my examiner, Johan Alenlöv, and my opponent, Shipeng Liu, for their clear feedback, patience, and explanations. I also want to thank my colleague Tugçe Izci for being a great team player in all the labs where we worked together and for clarifying the mathematical aspects of the subjects for me.

I cannot thank my mother, father, and sister enough. You are the most important part of my life; your support and love have fueled me, and this goal is yours, too. Finally, I am grateful to my girlfriend Karen. Thanks for always being there for me and helping me every step of the way since the beginning of this master's.

"Swimming the river ascending
Free of fear, filled with purpose
Nishikigoi fights through oceans of adversity
Against the current to obtain the highest goals
Swim through oceans of suffering
Climb to the falls of Dragon Gate
Become
Transform from koi into a beast
Dragon"
Becoming the dragon-Trivium

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Computer vision is considered one of the most important fields within the overlapping areas of mathematics and computer science due to its many real-life applications. This has marked a fast-paced development, which has been reflected in the features of everyday use products, such as Apple Face Detection [7] or the video game industry, with Niantic [8] as the company behind the Pokemon Go game.

A wide range of models cover tasks like Image Classification, Object detection, Visual Tracking, Semantic Segmentation, and Image Restoration [1]. These models, combined with computer vision techniques based on deep learning and non-deep learning, overcome the task of visual localization that aims to predict the camera pose from a query image. Some real-life applications of visual localization methods are *Live View*[9] on Google Maps in the area of augmented reality(AR), *Kiva* Amazon[10] warehouse robots in robotics,*Waymo*[11] cars in autonomous driving, and *UAV Navigation*[12] systems in unmanned aerial vehicles area, among others.

Like any other software application, Computer Vision systems are vulnerable to attacks, particularly *adversarial attacks*. Szegedy et al. [13] was the first to introduce the concept of adversarial perturbations; its objective is to "fool" the machine learning model so it fails its prediction. A simple example would imply having an image classification model whose purpose is to predict the class of the object or animal on an image, i.e., AlexNet [14], and adding a perturbation to the input image so small that it is imperceptible to the human eye to differentiate between the original image and the adversarial input (see Figure 1.1) to deceive the classifier. Eyholt et al. [15] present other scenarios of adversarial perturbations (see Figure 1.2), where the perturbations are noticeable but still small enough not to change for a human supervisor. This should not be confused with challenging the model with non-related inputs, i.e., inputting a dog picture to a cat image classifier.

Adversarial attacks can cause image classifiers to classify images incorrectly. Object detection models may not detect an object in different parts of the image. These attacks could provoke visual localization methods to compute a camera pose in a different location, de-

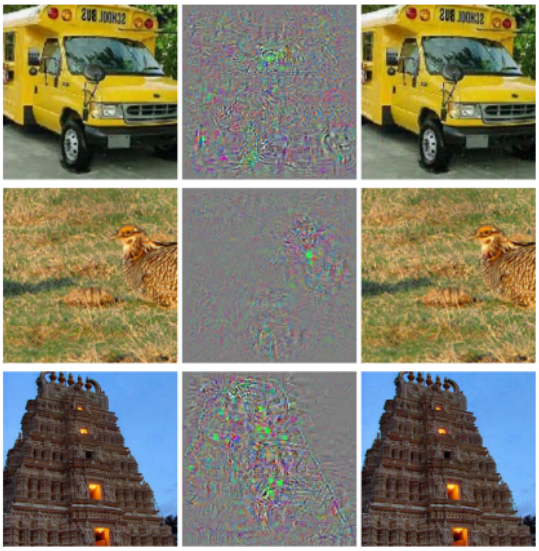


Figure 1.1: Column one images are correctly classified images(original), column two images are adversarial perturbations, and column three images are the adversarial examples, which are the result of adding the perturbation to the original images[13].

Figure 1.2: Attacks on autonomous vehicles. Road signs modified with adversarial perturbations(patches).[15].

creasing the model's accuracy.

Several research papers have reviewed and benchmarked the adversarial robustness of Image Classification [2, 3] and Object Detection[4] models. Adversarial attacks have also been introduced to motion estimation, particularly optical flow. Ranjan et al.[16] adopted the idea from [15] of introducing corrupted small patches into optical flow networks and showed an increase in the average end point error (EPE). Schmalfuss et al.[17] proposed and implemented an attack on motion estimation algorithm applied to snowflake renderer, which can provoke the networks to predict zero-flow even when the snowflakes are in motion.

Besides the mentioned research, to the best of our knowledge, none have investigated the *adversarial robustness* of visual localization methods. This refers to how vulnerable a model is to adversarial attacks and how it will impact the behavior of different performance metrics. Other related papers have benchmarked only the performance of visual localization (VL) methods [18, 19] but have not added adversarial perturbations. The current context showed us a crucial gap that is critical due to the presence of these systems in safety tasks (directly or indirectly) and represents a research opportunity. And it is the motivation to explore the topic.

1.2 Aim

The thesis aims to investigate the adversarial robustness of visual localization methods, first by conducting a literature review of visual localization methods and adversarial attacks. Exploring both subjects' different types and characteristics to identify a visual localization

model to adapt an adversarial attack(s). The following questions express the objective of the thesis:

1. What combination of adversarial attack(s) and visual localization method would be a good first approach? Which characteristics were taken into account when selecting these?

After starting the literature review, the following questions were formulated. They represent a more specific objective of the thesis due to identifying a visual localization method and attacks.

1. The adversarial attacks are focused on image classification models. How can the Fast Gradient Sign Method(FGSM) and Iterative Fast Gradient Sign Method (I-FGSM) be applied to the Accelerated Coordinate Encoding (ACE) model? Where in the architecture of the model can the attacks be created?
2. Is there a decline in performance from the ACE model when the adversarial attacks were injected? Does the loss and pose error present a change in magnitude?
3. Fast Gradient Sign Method and Iterative Fast Gradient Sign Method hyperparameters control the magnitude of the attacks. Will the change in the values of the hyperparameters affect the model's performance proportionally?

1.3 Delimitations

To start the discussion on implementing and evaluating adversarial attacks on visual localization methods. We mainly consider visual localization methods with scene coordinate regression and white box adversarial attacks, which refer to attacks where information about the model being attacked is known, such as architecture, training dataset, loss function, etc.

2 Previous Work

This chapter details the theoretical basis of the methods used in this thesis. In the first section, deep learning theory is reviewed, starting from the subjects of Neural Networks(NN) and the Neocognitron to Convolutional Neural Networks (CNN). The second section introduces Computer Vision concepts, starting with Visual Localization and its essential methods. The last section focuses on adversarial attacks, their different types, functionality, and their relation to computer vision.

2.1 Deep Learning

In a general sense, Deep Learning is defined as a division of Machine Learning (ML) whose architecture is based on multiple Neural Network (NN) layers [20, 21, 22, 23]. This multilayer approach is called Deep NN, and one of these precursors is the paper from Geoffrey E. Hinton et al. [24], where the authors elaborate on the Learning Algorithm for *Deep Belief Nets*. With the MNIST [25] dataset, they use complementary priors to learn "deep directed belief networks one layer at a time." This algorithm starts fine-tuning the weights to model the joint distribution of the digit pictures and their labels later with a three-hidden layers network.

2.1.1 Neural Networks

Deep Learning architecture is *Neural Networks* or *Artificial Neural Networks* (ANN). The core idea is inspired by the anatomy of a human neuron cell, which forms Biological Neural Networks (BNN). Figure 2.1 shows two neurons interacting and their different parts; these are connected through the *synaptic terminals* located on the end of the *Axon* to the *Cell body* or *Dendrites*, and the so- called *Synapses* are produced. The neurons send signals (electrical impulses) through the axon to other neurons. This sounds familiar to ANN functionality[26] when one neuron receives the input signals from the last layer.

2.1.1.1 *The Neuron*: The cornerstone

The McCulloch-Pitts neuron model [28] brought the first step to an *artificial neuron* in 1943. The article proposed eight theorems that describe how the interaction between neurons can

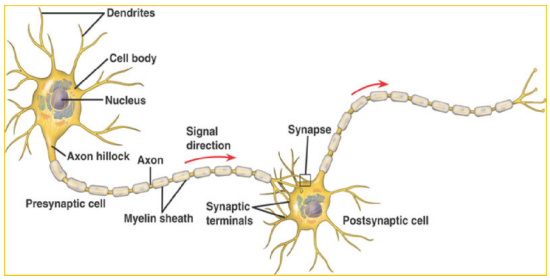


Figure 2.1: Two Neurons interacting [27].

be treated through propositional logic. It adopts the idea of neurons being in an on/off state. Furthermore, the flexibility concept from

ML models have their first appearance, as it is described that by applying the theorems, every network behavior can be described under the combination of logical computations, including operations such as OR, AND, and Negation.

The Perceptron model from Frank Rosenblatt [29] (Figure 2.2) is the second step toward modern Neural Networks. Compared with the McCulloch-Pitts model, the inputs are numbers instead of on-and-off values and have their respective weight; a weighted sum is performed, and the result goes into the activation function (step function). It is called a linear threshold unit because of the linear nature of the activation function. (Equation 2.2)

The Perceptron learns in a supervised setting by inputting multiple examples with their true label and comparing these with the predicted output to tune the weights to predict a correct label when inputting unseen data. To adjust the weights, Rosenblatt [29] proposed the learning rule or weight update. (Equation 2.1)

$$w_{ij}^{t+1} \quad = \quad w_{i,j} + \eta(y_j - \hat{y}_j)x_i \tag{2.1}$$

Where w_{ij}^{t+1} represents the weights of the next iteration between the i^{th} input neuron and the j^{th} output neuron. η the learning rate, y_i true Output of the j^{th} neuron, $\hat{y}$ predicted Output of the j^{th} neuron and x_i the i^{th} input value.

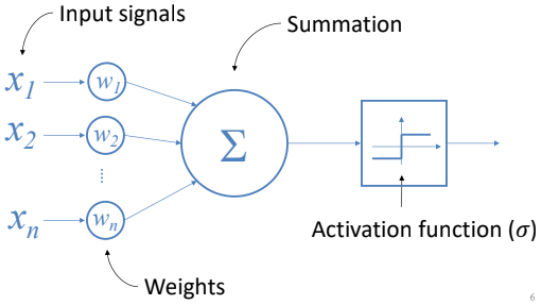


Figure 2.2: The Perceptron architecture [30].

$$\begin{aligned} \text{Inputs:} \quad & X = \{X_1, X_2, ..., X_n\} \\ \text{Weights:} \quad & W = \{W_1, W_2, ..., W_n\} \\ \text{Summation:} \quad & g = X^T \cdot W \\ \text{Activation function:} \quad & step(g) \\ \text{Output:} \quad & h_w(x) = step(X^T \cdot W) \\ & step(g) = \begin{cases} 1 & \text{if } g \geqslant 0, \\ 0 & \text{if } g < 0 \end{cases} \end{aligned} \tag{2.2}$$

The resemblance between the biological neuron and artificial neuron is evident when comparing their architecture side to side. The BNN Synaptic terminals are homologous to the Inputs of the ANN, and the Cell Body is the unit or neuron from ANN, where, in both cases, input signals or action potentials are summed. However, BN follows different types of summation mechanisms, such as *Spatial* and *Temporal* summation [31]. Figure 2.2 and Figure 2.3 illustrate these similarities.

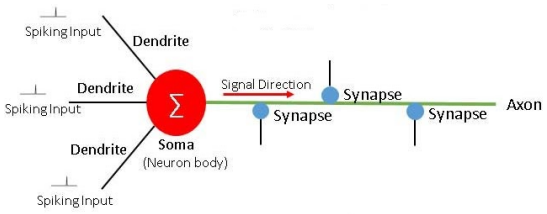


Figure 2.3: Biological Neuron structure[32]

As pointed out by Minsky and Papert [33], a single Perceptron has limitations and cannot solve the XOR classification problem. Several Perceptron units were assembled to overcome this, which defined the *Multilayer Perceptron*(MLP) architecture, proposed by Rosenblatt [34] in 1961. It comprises the *Input* layer, the *Hidden* layer, which can contain one or more stacked layers, and the *Output* layer(Figure 2.4). In addition, the Bias term is presented.

To have more flexibility, MLP architecture relies on the Bias term and the Covers Theorem [35] about the separability of pattern, which states that in the case of "*a complex pattern-classification problem treated in a high-dimensional space, nonlinearly would be more plausible to be linearly separable than in a low-dimensional space.*" (Haykin,1998,p.279)

The *Bias* term is a constant added to the multiplication between the features and the weights; it shifts the activation function to the negative or positive side; this shift represents the move of the decision boundary out of the origin and can approximate functions that do not pass through the origin. As a result, the output function in Equation 2.2 transforms to Equation 2.3.

$$h_w(x) = step(X^T \cdot W + b), \text{ where } b = 1. \tag{2.3}$$

The flexibility allows the MLP model to onboard more complex problems like the XOR. This problem occurs because the single Perceptron cannot deal with points that are not linearly separable. It can create a line, a plane, or a hyperplane in low (1D) or higher-dimensional (2D,3D) space, but all of these are linear.

Under the premise of Cover's theorem, non-linear functions were included in the activation function options to transform the problems into a high-dimensional space in a nonlinear

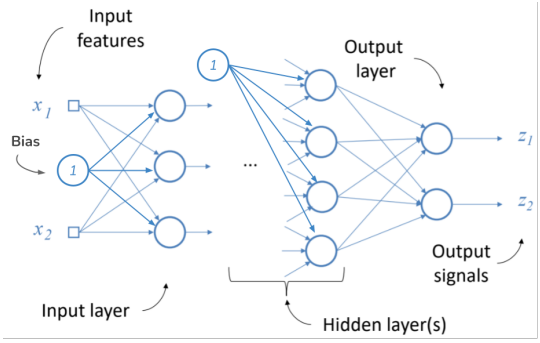


Figure 2.4: Multilayer Perceptron (MLP) [30].

form. This transformation is needed to create boundaries of non-linear shapes and project data that could be linearly separable in a high-dimensional space. Figure 2.5 shows common types of non-linear functions. Radial-basis function (RBF) networks present how non-linear functions can deal with a complex classification function. As shown in Figure 2.6, blue and red circles can not be divided or classified in a one-dimensional space with a linear approach. On the contrary, when a Gaussian activation function (RBF) is applied, it can be seen that the circles in a 2D space, the blue circles are found in an inner circle, and the red circles around the perimeter of these can easily separate these two types of circles by a radial boundary.

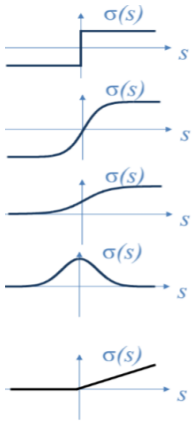


Figure 2.5: Non-linear functions [30].

$$\begin{aligned}
(a) \quad \text{Step Function} \quad & step(g) = \begin{cases} 1 & \text{if } p \geqslant 0, \\ 0 & \text{if } p < 0 \end{cases} \\
(b) \quad \text{Hyperbolic tangent} \quad & \sigma(s) = tanh(s) \\
(c) \quad \text{The Fermi-function} \quad & \sigma(s) = \frac{1}{1+e^{-s}} \\
(d) \quad \text{Guassian function (RBF)} \quad & \sigma(s;\gamma) = e^{-\frac{s^2}{\gamma^2}} \\
(e) \quad \text{Rectifying Linear Unit (ReLU)} \quad & \sigma(s) = max(0,s
\end{aligned} \tag{2.4}$$

MLP is part of Feedforward Neural Networks (FNNs) because the data, e.g., an image, moves in one direction, starting from the input layers, then the hidden layer, and ending at the input layers. Other types are Radial Basis Function Neural Networks (RBFNNs), Recurrent Neural Networks (RNNs) [36], Spiking Neural Networks (SNNs) [37], Deep Neural

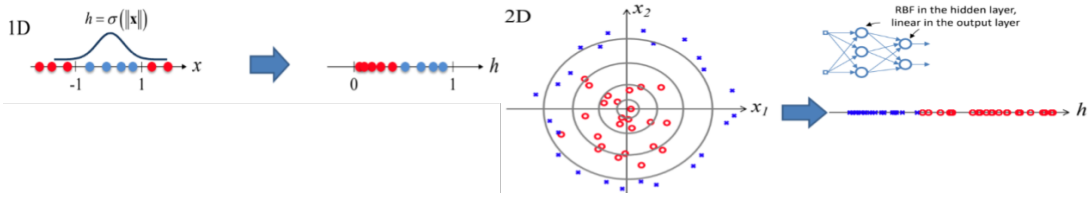


Figure 2.6: Classifiers Comparison - Lineal activation function vs RBF [30].

Networks (DNNs), Evolutionary Neural Networks (EVNNs) [38]. An ANN is a DNN, when it has many hidden layers, which can be interpreted as *deep* in hidden layers. The first three NN mentioned are considered "Old Generations" and the latter "New Generation" [39]. The exponential growth in research and applications of the different types of NNs was possible because of the backpropagation [40] training algorithm.

2.1.2 Backpropagation

In a supervised learning setting with the disposition of pairs of data and its true outcome (X, Y) for training, the main idea of the backpropagation algorithm is to adjust the value of the network's weights through the backpropagation of the error to minimize the loss function or error between the prediction of the network and the true outcome. Later, the tuned weights will be used to predict the outcome in data not seen by the model. Based on the Stochastic Gradient Descent (SGD) optimization method, it has two passes, Forward and Backward (Backpropagation), and the process is defined as follows:

Algorithm 1 Backpopragation [41].

Start:

1: Weights are initialized.

2: Choose a Loss function (Regression or Classification)

Forward Pass: Compute the output of each layer (hidden and output) from the output of each neuron. (Vectorized form)

3: Input the X vector into the input layer:
Each neuron output is $z_j = W_j \cdot X + b_j$ and activation function output $a_j = \sigma(z_j)$

4: Compute the output $z^l = w^l a^{l-1} \quad + b^l$ and the output through the activation function $a^l = \sigma(z^l)$ for the Layers $l = 2, 3, .., L$

5: Compute the Loss (e.g. MSE): $C = \frac{1}{2}(\hat{y} - y)^2$ $\hat{y}$ is the predicted output, and y is the true value.

Compute the output error (output layer):

6: $\delta^l : \frac{\partial C}{\partial a^L} \odot \sigma'(z^L)$

$\frac{\partial C}{\partial a^L}$: Partial derivative of the cost function C(Loss function)

$\sigma'(z^L)$: The derivative of the activation function

Backward Pass (Backpropagation):

7: Backpropagate the error by computing the error of each layer

$\delta^{x,l} : ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^L) \qquad l = L-1, L-2, ...2$

$(w^{l+1})^T$: Weight matrix for the $(l+1)^{th}$ layer. $\delta^{x,l+1}$: error at the $(l \quad + \quad 1)^{th}$ layer.

8: Weights: $w^l = w^l - \eta \Sigma \delta^{x,l} (a^{x,l-1})^T$

9: Bias: $b^l = b^l - \eta \Sigma \delta^{x,l}$ η: Learning rate. $l = L, L-1, L-2, ..., 2$

2.1.3 Visual Cortex: the bridge to Computer Vision

The fundamentals of Deep Learning relevant to computer vision are inspired by the biological mechanisms of the striate cortex cells found in mammals. Three fundamental studies[42, 43, 44] conducted by David H. Hubel and Torsten Wiesel, between 1959 and 1968, describe how the cells of cats and monkeys react to certain light stimuli with different planes, colors, movements, and orientations.

In the first study, they found some units (single cells) remained without a response when changes in the background illumination were performed. On the contrary, units that respond to stationary light spots also react to horizontal light movements; however, some only respond to movements in one direction and some to both. The second study presents that to drive a unit effectively, a specific stimulus is needed in orientation, position, form, and orientation. Again, some units reacted stronger to a direction of movement rather than both (like going from positive to negative on the X-axis), originating from the different arrangement of the receptive field. Additionally, the two receptive fields of the binocular unit, excitatory and

inhibitory areas, interacted.

The third study presents that the respective field of the units responds to a particular region of the visual field. They classified the neuron into simple, complex, and hypercomplex. Simple neurons have 'on' and 'off' activations and are triggered by the orientation of edges and to specific parts of the visual field. Complex neurons do not have 'on' and 'off' activations but are triggered by the movement of the light. Hypercomplex neurons have the characteristic of having a fall-off response when the stimulus passes the activation part of the receptive field, which means they react to the length of it. Another key finding is some units respond to the vertical or horizontal light stimulus. In addition, the neurons seem to be organized in columns that run vertically on the layers of the cortex. The neurons over these columns react similarly to specific stimuli, such as the ones mentioned above.

The studies established that single cells react to a specific respective field and that some of these units receptive fields interact with each other. The units are triggered by particular movements(or none) and the direction of it, in addition to the color or shape of the stimulus.

2.1.4 Neocognitron

Fukushima [45] approach was to create a neural network for pattern recognition able to mimic how a human would do it; this to understand better the biological neurons. To achieve a pattern recognition capability similar to a human being, the architecture was intended to address the problem of shifting the position or/and distortion of the input patterns that models, such as Perceptron(Rosenblatt) and Cognitron(Fukushima) had i.e., we as humans can recognize the number one written on a sheet of paper regardless of its position and/or if the shape has some level of distortion(different hand writings).

The network is a multilayered architecture that follows an unsupervised learning method to recognize the different patterns based on their similarities in geometry. The model has a self-organization process (unsupervised), and it has to be inputted several times stimulus patterns. The result of this process is the emulation of the biological hierarchical model of the visual nervous system, recovered from Wiesel and Hubel papers [43, 44], with Fukushima's addition of *grandmother cells*. (See Figure 2.8). These refer to neurons that react to more complex features than hypercomplex cells, like triangles, squares and the contour of a monkey's hand.

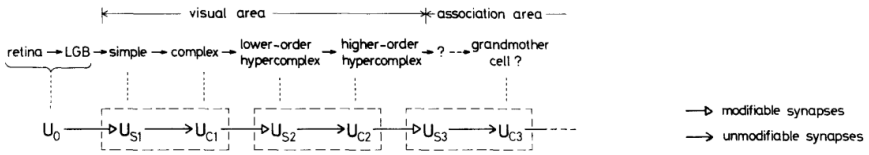


Figure 2.7: Hierarchical model that mimics the pattern recognition of human vision relation with the Neocognitron model[45].

The Neocognitron overview structure (Figure 2.7) represents the hierarchical model. It is a chain of modular structures that starts with the input layer U_0, which is a photoreceptor array that plays the role of the retina and the lateral geniculate body. Each rectangle contains a pair of layers, one with "S-cells" (simple) and the other with "C-cells" (complex). The simple layer S of the lth module is denoted as U_{Sl}, in the same way complex layers are U_{Cl}. Both types of cells in their respective layer are arranged into sub-groups depending on the stimulus features of their receptive field. These subgroups are called cell-planed and are specified as "S-plane" and "C-plane" according to the type of cells.

Cells from a single cell plane have the inputs of the same spatial distribution. And presynaptic cells are shifted from cell to cell. This arrangement mechanism produced that the cell from a plane has, from different positions, the same function receptive field.

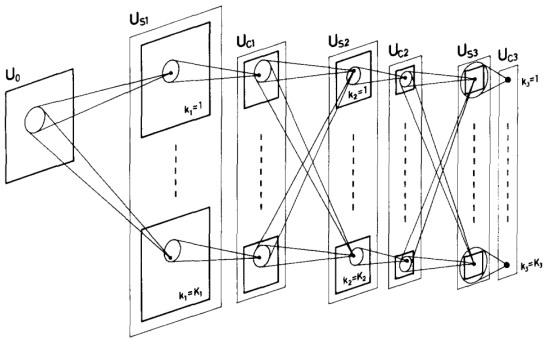


Figure 2.8: Neocognitron architecture [45].

The interconnections between layers and planes are shown in Figure 2.9. It is important to point out that *" the deeper the layer is, the larger becomes the receptive field of each cell of that layer."* The model takes into account the inhibitory cells $v_{Sl}(n)$ and $v_{Cl}(n)$. The numerical expressions of the cell outputs are below in Equation 2.5

$$u_{Sl}(k_l, \mathbf{n}) = r_l \cdot \varphi \left[\frac{\left(1 + \sum_{k_{l-1}=1}^{K_{l-1}} \sum_{\mathbf{v} \in S_l} a_l(k_{l-1}, \mathbf{v}, k_l) \cdot u_{Cl-1}(k_{l-1}, \mathbf{n} + \mathbf{v})\right) - 1}{1 + \frac{2r_l}{1+r_l} \cdot b_l(k_l) \cdot v_{Cl-1}(\mathbf{n})} \right] \tag{2.5}$$

where

$$\varphi[x] = \begin{cases} x & x \geqslant 0 \\ 0 & x < 0 \end{cases} \tag{2.6}$$

From Equation 2.6 $a_l(k_{l-1}, \mathbf{v}, k_l)$ and $b_l(k_l)$ are the efficiencies of the excitatory and inhibitory synapses. r_l is the efficacy of the inhibitory input; the larger the value, the more selective the response to certain features.
Equation 2.7 represents the inhibitory cell and has connections to the S-cell. C_{l-1} is the efficiency of the unmodifiable excitatory synapses.

$$v_{Cl-1}(\mathbf{n}) = \sqrt{\sum_{k_{l-1}=1}^{K_{l-1}} \sum_{\mathbf{v} \in S_l} c_{l-1}(\mathbf{v}) \cdot u^2_{Cl-1}(k_{l-1}, \mathbf{n} + \mathbf{v})} \tag{2.7}$$

Equation 2.8 shows the output of a C-cell. It depends of the inhibitory cell v_{sl}, which is described by Equation 2.9.

$$u_{\mathrm{Cl}}(k_l, \mathbf{n}) = \psi \left[1 + \frac{\sum_{\mathbf{v} \in D_l} d_l(\mathbf{v}) \cdot u_{\mathrm{Sl}}(k_l, \mathbf{n} + \mathbf{v})}{1 + v_{\mathrm{Sl}}(\mathbf{n})} - 1 \right], \tag{2.8}$$

$$v_{\mathrm{Sl}}(\mathbf{n}) = \frac{1}{K_l} \sum_{k_t=1}^{K_l} \sum_{\mathbf{v} \in D_l} d_l(\mathbf{v}) \cdot u_{\mathrm{Sl}}(k_t, \mathbf{n} + \mathbf{v}). \tag{2.9}$$

Figure 2.9 represents how patterns go through the interconnections between cells. In this case, the pattern is the letter A. From layer U_{S1} in the S-plane with k=1, have a 2-D array of S-cells that pull out " & -shaped features." The following plane is a complex C-plane in layer

U_{C1}, and the inputs come from the S-cells of the S-plane within the circle. Then, the outputs of the C-plane go into the next layer, which is again composed of an S-plane. Furthermore, the receptive fields or circles within each plane overlap with other planes from that layer. Allowing the hierarchical approach, the model can learn how to identify different features of a pattern in different positions.

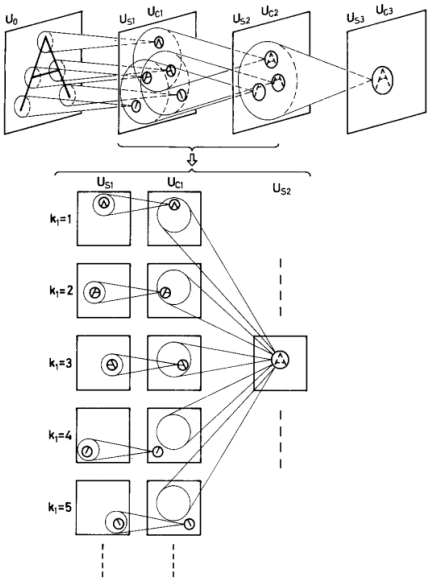


Figure 2.9: Neocognitron model with interconnections between simple and complex cells[45].

2.1.5 Convolutional Neural Networks

As reviewed in this thesis, the development of CNN neural networks has come a long way from the Perceptron, the MLP, the Wiesel-Hubel hierarchical model, and the Neocognitron could be considered the concept's origin. From the late 80's until the end of the 90's[46, 47, 48], several studies conducted primarily by LeCun set the base architecture for modern CNNs, particularly the well-known LeNet-5 [49].

The *convolutional* and *pooling layers* are aggregated to the full-connected layers and activation function (i.e., sigmoid)

2.1.5.1 Convolutional Layer

Taking into consideration an image as the input. On the contrary, with fully connected layers architectures(i.e., MLP), the neurons from the first convolutional layer are only connected to the portion of the pixels under its receptive field. Consecutively, the second-layer neurons are also connected to neurons within their receptive field. The first layer focuses on low-level features from the image; each additional layer assembles it into high-level features [26]. These cascade connections continue and follow the hierarchical model of Wiesel-Hubel and the skeleton from Fukushima's architecture. Figure 2.10 illustrates how the correspondence of receptive fields between layers.

Figure 2.11 shows the zero padding technique used to maintain the same dimension over all the layers; this is achieved by adding zeros around the input layer. Furthermore, it can be seen that a neuron located in (i,j) is connected to the neuron of the last layer located in (i+f_h-1,j+f_w-1), where the height(f_h) and width(f_w) are parameters. The stride on this layer is 1, as this is the separation between receptive fields. If a different stride $s_{h,w}$ is chosen, then the current neuron location (i$\times$ s_h,j$\times$ s_w) and its input would be neurons located in (i$\times$ s_h

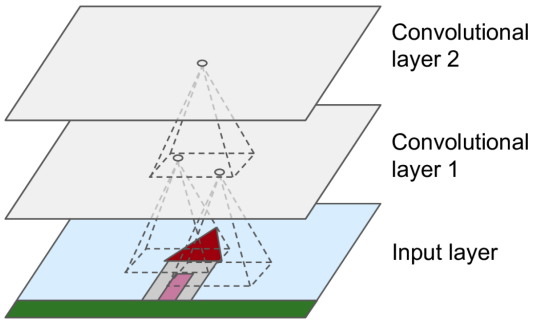


Figure 2.10: Convolutional layers receptive fields [26].

$+f_h$-1,j$\times$ s_w+f_w-1). The strides parameter allows us to connect the input layer into a smaller layer like the one in Figure 2.11

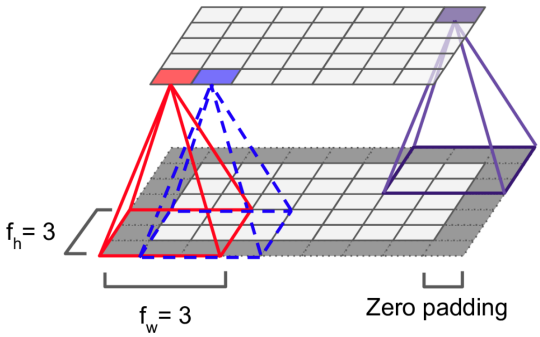


Figure 2.11: Receptive and stride [26].

Convolutional kernels are the weights of the neurons. These act like a filter because, according to their configuration, they will respond to specific features like the cortex's units(cells) in cats and macaques [43, 44, 42]. A kernel of dimension $i \times j$ with a vertical column of ones and the rest full of zeros would respond to vertical lines or edges; on the other hand, a kernel with a horizontal row of ones, the rest full of zeros will react to horizontal lines or edges (Figure 2.12). This kernel applies to all the neurons of a layer and is called a *feature map*.

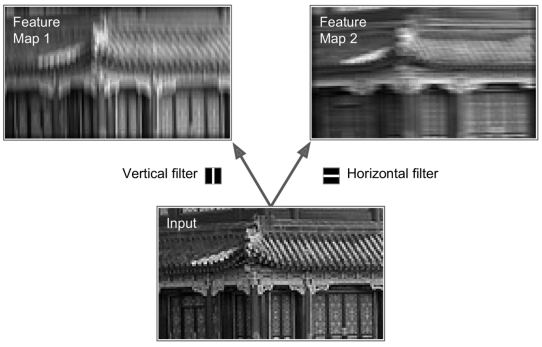


Figure 2.12: Vertical and Horizontal filters result [26].

The overview of a CNN focus on image application has three channels corresponding to the Red, Green and Blue (RGB). The layers can have stacks of feature maps that transform

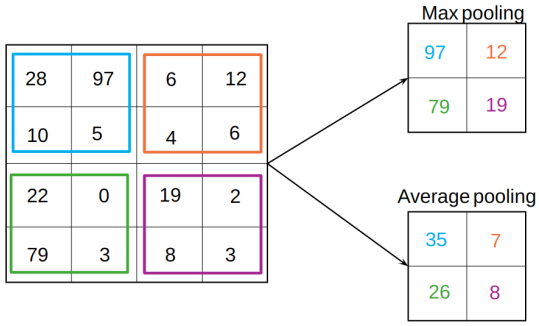


Figure 2.13: Common types of pooling layers.

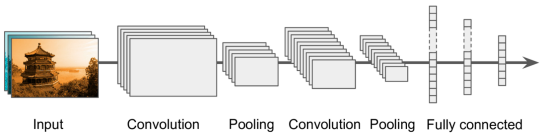


Figure 2.14: CNN architecture composed of a combination of convolution and pooling layers, and at the end fully connected layers[26].

them into a 3D array instead of a 2D array. The output of a neuron in a convolutional layer is shown in Equation 2.10

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \quad \begin{matrix} i' = i \times s_h + u \\ j' = j \times s_w + v \end{matrix} \tag{2.10}$$

Where $z_{i,j,k}$ represents the neuron output from row i, column j, in the feature map k, s_h, s_w the strides f_h,f_w the receptive field dimensions, $f_{n'}$ how many feature maps are in the previous layer, $x_{i',j',k'}$ the output from the neuron on the last layer, b_k the bias term of feature map k and $w_{u,v,k',k}$ the weight.

2.1.5.2 Pooling layer

Pooling layers are designed to downsample the input and decrease the size of the feature maps and, consequently, the computational resources needed. In addition, the chances of overfitting are reduced. It is achieved by aggregating the output from the convolutional layer by applying the max or average functions over some of the output. As shown in (Figure 2.13) from the convolutional layer output of size 4x4, the max function is applied over the four matrices of 2x2 size within this, resulting in taking the maximum value of each sub-matrix and downsampling from a 4x4 to a 2x2 matrix.

Assembling all the layers of a typical CNN architecture is presented in Figure 2.14: the image gets smaller as the networks get deeper, and it is connected to the end of a fully connected network that works as the classifier with a softmax activation layer.

2.2 Computer Vision

Computer vision could be described as the conjunction of techniques applied to allow machines or computers to see the world around as a human would, and in some particular applications better than humans. To "see" as or better than a human implies that the machines involved can do tasks such as object and person recognition, compute the 3D geometry of the environment, "enhance" images, improve photographs, object and face tracking, among others [50, 51].

2.2.1 Camera Pose

The camera pose describes the position and orientation of the camera in a 3D space relative to a reference coordinate system. It is composed of the translation (position) and the rotation (orientation). The translation or the displacement along the axes x, y, and z is represented by the translational vector **t** in Equation 2.11. Different methods, such as rotation matrix, quaternion, Euler angles, and axis angles, can represent the rotation. This section introduces only the homogeneous coordinates and quaternion representations because these are the methods used by the 7scenes dataset and the visual localization method(ACE).

$$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{2.11}$$

The rotation matrix (**R**) represents the rotation in a 3D space containing three degrees of freedom, corresponding to the rotation over the x, y, and z-axis. It has a dimension of 3x3, and its determinant equals 1, as shown in Equation 2.12.

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.12}$$

The individual rotation over the axes x, y, and z can be obtained by multiplying the rotation matrix **R** by the rotation matrices in Equation 2.13. To do this, one must know the order of the rotations applied to obtained **R**.

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, \quad \mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \quad \mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

The homogenous transformation matrix representation, shown in Equation 2.14, integrates the rotation matrix **R** and the translational vector **t**.

$$\mathbf{M} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.14}$$

where the upper left corner 3x3 matrix is the rotation matrix **R**, and the translation vector **t** is in the upper right corner.

In conjunction with the translation vector **t**(Equation 2.11), the *quaternion* (Equation 2.15) is another representation of the *pose*. The quaternion is of $\mathbb{R}^4$; composed of one real part (scalar, q_w) and three imaginary parts (vector, $[q_x, q_y, q_z]$). Quaternions of length *1* are called unit quaternions and represent rotations in a 3D space [52]. By considering α and **v** from the axis-angle representation, rotations can be linked to unit quaternions if the scalar part is represented as $\cos\frac{\alpha}{2}$ and the imaginary part as $\mathbf{v}\sin\frac{\alpha}{2}$ [6].

$$\mathbf{q} = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix} \tag{2.15}$$

2.2.2 Visual Localization

Visual localization is the task of estimating the 6 *degrees of freedom* (DOF) camera pose from a query image using some type of space representation, such as images (database) and 3D point cloud, among others [5, 6]. There are three main approaches to solving visual localization tasks, Learning- based, Structure-based, and Image-based [19]. All require several

components within their architecture to estimate the camera pose, making the workflow, in most cases, non-linear and sometimes implies doing tasks in parallel [53].

2.2.2.1 Learning-based

Learning-based models learn from the data to predict the pose or scene 3D coordinates. Such models can be *Absolute Pose Regression* (APR) or *Scene Coordinate Regression* (SCR), and primarily are structured with *CNN architecture* [54] for end-to-end learning or, in a non-Deep Learning manner, with *Regression Forests* [19]. If the model predicts the coordinates, a loop of Perspective-n-Point (PnP) and Random sample consensus (RANSAC) is normally incorporated to estimate the pose.

APR model PoseNet [55] regresses the 6 DOF camera pose from a single 224x224 RGB image. Where the pose is the vector $\mathbf{p}$, $p = [x, q]$, conformed by the 3D camera position x and the quaternion q. To learn the location and orientation simultaneously, the loss function in Equations 2.16 was implemented.

$$\mathcal{L} = ||\hat{x} - x||_2 + \beta||\hat{q} - \frac{q}{||q||}||_2 \tag{2.16}$$

Where x is the 3D camera position, $\hat{x}$ predicted camera position, q is the target quaternion, $\hat{q}$ is the predicted quaternion, and β is a scale factor. The architecture is a CNN pre-trained for image classification GoogleLeNet [56] with a few changes. It has 23 convolutional layers, and it removed the softmax layers and modified each final fully connected layer to output the pose vector to replace the softmax classifiers with regressors. A fully connected layer was added before the final regressor.

SCR method proposed by Brachman and Rother [57] couple a DNN with a pose solver (g) that includes a differentiable RANSAC $\tilde{h}$, called DSAC. The first part is composed of a CNN that includes eight convolutional layers of size [3x3] and eight convolutional layers of size [1x1], with three skip connections and a stride of two that sub-sample the input (gray RGB image) by a factor of 8, which predicts the scene coordinates. The 2D-3D correspondences C between the input image and the 3D space is represented as Equation 2.17.

$$C = (p_i, y_i | y_i \epsilon \mathcal{Y}) \tag{2.17}$$

where $\mathcal{Y}$ is the scene coordinates, y_i the ith scene coordinates, and p_i the 2D image coordinate of pixel i described on Equation 2.18.

$$p_i = Kh^{-1}y_i \tag{2.18}$$

where h is the hypothesis where the poses are sampled from and K the camera calibration matrix. Using this set of correspondences PnP solver g recovers the camera pose. The loss function used to train the model applies L_1 distance. (Equation 2.19)

$$\mathcal{L}\left(\mathbf{y}_i, \bar{\mathbf{y}}_i, \mathbf{h}^*\right) = \begin{cases} \hat{r}\left(\bar{\mathbf{y}}_i, \mathbf{h}^*\right) & \text{if } \mathbf{y}_i \in \mathcal{V} \\ |\bar{\mathbf{y}}_i - \mathbf{y}_i| & \text{otherwise.} \end{cases} \tag{2.19}$$

That replace the ground truth scene coordinate with a heuristic scene coordinate $\hat{y}_i$ obtained from the relation $\hat{y}_i = h\hat{e}_i$, that includes the ground truth camera pose h and hallucinated 3D camera coordinates $\hat{e}_i$ reprojected. The projection error $\hat{r}$ is also included and defined as Equation 2.20.

That replace the ground truth scene coordinate with y_i^* with a heuristic scene coordinate $\bar{y}_i$ obtained from the relation $\bar{y}_i = h^*\bar{e}_i$, that includes the ground truth camera pose h^* and

hallucinated 3D camera coordinates $\bar{e}_i$ reprojected. The projection error $\hat{r}$ is also included and defined as Equation 2.20.

$$\hat{r}(y_i, h) = ||p_i - Kh^{-1}y_i|| \tag{2.20}$$

Shotton et al. [58] proposed an SCR model that applies Regression Forest instead of convolutional neural networks. The regression Forest is conformed by several decision trees that map image patches to scene coordinates. The trees work independently, which means each tree is trained with a randomly selected subset of data. The regression forest associates scene coordinates with any 2D image pixel, using this information to estimate the optimal camera pose H^* by minimizing the energy equation $E(H)$ as in Equation 2.21.

$$H^* = argminE(H) \tag{2.21}$$

where H is the camera pose matrix, and $E(H)$ can be defined as in Equation 2.22.

$$E(H) = \sum_{i \epsilon \mathcal{I}} \rho(\min_{m \epsilon \mathcal{M}_i} ||m - Hx_i||_2) = \sum_{i \epsilon \mathcal{I}} e_i(H) \tag{2.22}$$

The energy equation (Equation 2.22)is minimizing the L2 distance between the predicted scene coordinates **Hx** and the truth coordinates **m**. Where i is the pixel index, ρ is a error function, $\mathcal{M}_i = \mathcal{M}(p_i)$ is the set of models estimated by the regression tree forest from the pixel p_i.

2.2.2.2 Structure-based

To compute the pose, structure-based methods [59, 60, 61, 53] rely on a 3D representation, as point clouds, from the image query environment. This representation is built from an image database, applying the Structure From Motion (SFM) [62] algorithm. The next step is extracting descriptors from the query and database to extract the 2D features. Then, the 2D features and the 3D representation are associated to obtain a set of the 2D features points extracted and the three-dimensional points x, y, z. The 2D-3D set is taken into a PnP solver within a RANSAC loop to compute the pose.

Hierarchical localization model [53] introduced HF-Net to compute the camera pose. HF-Net is based on a monolithic CNN that simultaneously computes global and local descriptors with its keypoint scores. To do this, it incorporates the CNN *MobileNet* [63], then take the output features into a SuperPoint [64] decoder that outputs the keypoints and local descriptors. From the output of the MobileNet, the Network bifurcates to a second MobileNet architecture, and the output features of this go into a NETVlad layer that produces the global descriptor. The loss used in the HF-Net training is shown on Equation 2.23.

$$\mathcal{L} = e^{-w_1}||d_s^g - d_{t1}^g||_2^2 + e^{-w_2}||d_s^l - d_{t2}^l||_2^2 + 2e^{w3}\text{CrossEntropy}(p_s, p_{t3}) + \sum_i w_i \tag{2.23}$$

Where $\mathbf{p}$ are the keypoint scores, $w_{1,2,3}$ the optimized variables, $\mathbf{d}^{g,l}$ the global and local descriptors. And the indices s and t refer to the student and teacher, because multi-task *distillation* technique is used.

The model takes a query image and extracts the global descriptor with HF-Net. At the same time HF-Net is used to extract the local descriptors and keypoints of the image database to apply SFM and create a 3D representation. The prior retrieval is done by matching the global descriptors of the query and the database by performing KNN. This will provide candidate locations called "places," narrowing the points on the 3D representation that must be checked. If two or more locations have overlapping parts of the same 3D points, a "place"

is located. The 2D keypoints extracted by the HF-Net on the query image are matched with the 3D points contained in the places. Having the sets of 2D-3D points, the RANSAC+PnP loop is applied to these, and the pose with a higher number of inlier points is the estimated pose.

The PixLoc [65] model proposed by Sarlin et al. follows a different approach to 2D- 3D matching and using a 3D model. It is composed of a CNN that extracts from the query image I_q and a reference image I_k D_l dimensional feature map $F^l \epsilon \mathbf{R}^{W_l \times H_l \times W_l}$ from each level $l \epsilon L, ..., 1$. The feature maps capture spatial context and semantic information as the levels get coarser, creating a better representation of both images. The loss function (Equation 2.24) compares at each level l the estimated pose with the ground truth.

$$\mathcal{L} = \frac{1}{L} \sum_{l} \sum_{i} ||\Pi(\hat{R}_l P_i + \hat{t}_l) - \Pi(R P_i + t)||_\gamma \tag{2.24}$$

where Π represents the reprojection function applied over $\hat{R}_l$ that represents the estimated rotation, R the rotation ground truth, P_i 3D points, $\hat{t}_l$ the estimated translation, t the ground truth translation and γ is the Huber cost to manage the noise of the reprojection error.

In a test setting with an image query, a reference image, and a sparse or dense 3D point cloud the objective of this is to find the pose (**R**,**t**) that minimizes the delta between the reference image and the query image. To achieve this, the total alignment error is implemented as shown in Equation 2.25.

$$E_l(R, t) = \sum_{i,k} w_k^i \rho(||r_k^i||_2^2) \tag{2.25}$$

The alignment error E_l is minimize by applying the Levenberg Marquardt optimization. Where the residual between the image query and the reference is taken into account as shown in Equation 2.26.

$$r_k^i = F_q^l[p_q^i] - F_k^l[p_k^i] \epsilon \mathcal{R}^D \tag{2.26}$$

Where F is the feature maps of the query q and reference k, and $p = \Pi(\hat{R}_l P_i + \hat{t}_l)$ the projection of i in the query based on the pose estimate.

2.2.2.3 Image-based

The image-based method relies on image retrieval methods. These are intended for place recognition tasks by retrieving the image most similar to the query image selected from an image database of the work environment [19]. To repurpose these methods for visual localization, the camera pose of each image is included and returned as a rough estimate of the image query camera pose.

The method is divided into two phases. The first phase is to represent the query image and database efficiently. The second phase measures the similarity or dissimilarity between the representations and chooses the most similar image from the database. There are several ways of doing the representations; the models to attack the task are mainly classified in *Descriptor Matching-Based*, *Hashing-Based*, and *Attention-Based*.

Descriptor Matching-Based retrieval commonly uses a variation of VLAD [66] to produce the descriptors. NetVLAD [67] takes a CNN architecture, cuts off the last convolutional layer, and returns a descriptor. A new pooling layer based on VLAD is aligned on top of the CNN. The final output is a descriptor with shape $(K \times D) \times 1$, where K is the number of clusters (feature space divided into clusters) and D the dimension of the descriptors (the depth of the last CNN layer feature map). After the descriptors are created, to measure the similarity of

the query image and the database, the *Euclidean* distance is computed. Then, the most similar images are top-ranked and returned.

Akihiko et al. [68] and Hyo et al. [69] follow similar pipelines. Akihiko et al. proposed *DenseVLAD*, a method that first computes SIFT descriptors at different scales (16,24, 32, and 40 px) to avoid, then the descriptors are aggregated using VLAD and the dimensionality reduced with PCA. The similarity is measured via the normalized product. Hyo et al. proposed *PBVLAD*, Per-Bundle VLAD is composed by first identifying Maximally Stable Extremal Regions (MSER) within the image and SIFT features from each region are detected. Then, the descriptors of each MSER region are aggregated using VLAD, and later, the similarity between the query image and the database images is measured by computing the dot product.

Haomiao Liu et al. [70] presented a model for *Hashing-based* retrieval that incorporates a CNN architecture of three convolutional layers, one maximum pooling layer, and two average polling layers. With the addition of two fully connected layers, the first with 500 nodes and the output layer with k nodes (nodes). The magnitude of k depends on how long the binary representation needs to be, as the user requires. To obtain binary values, it is coupled with a *sign* activation function to have output $= (-1, 1)$. To measure the similarity, the Hamming distance [71] between two binary codes is computed.

With the introduction of Transformers [72] and Visual Transformers (ViT) [73], *Attention-Based* models were introduced like Alaadeldin El-Nouby et al. [74] that trained ViT for image retrieval. The input image is divided into M patches of equal size (16×16); the patches are linearly projected into M vector-shaped tokens. A 1-D positional encoding vector is added to the tokens to include the location prior. Finally, a CLS (class) token is incorporated so the output token of the ViT represents a global descriptor of the image. The transformer has Multi-Headed Self Attention and Feed-Forwards layers, the global descriptor's output. Cosine similarity is the measurement of similarity computed between the descriptors of the query image and the database images.

2.3 Adversarial Attacks

This chapter introduced the concept of Adversarial attacks, their types and goals, and a review of some adversarial attacks of each type.

In 2014, Szegedy et al. [13] introduced the term *adversarial example* in an object recognition task context, which refers to a test image that has been added an imperceptible perturbation. To produce this *adversarial perturbation*, the input image is optimized to maximize the prediction error. Furthermore, the perturbations produced by one model are transferable to another model trained with other hyperparameters and architectures, meaning that it can *fool* it too.

To estimate the magnitude of the adversarial perturbation the l_p norm is applied, i.e., $l_\infty, l_0, l_1 \text{or} l_2$. Another term relevant to the topic is the *adversary*, that is, the *agent* that creates the *adversary example* or *image*. The adversary's goal is to fool the machine learning model, which can mean two things. To produce a *decline in the confidence score* from the prediction of the target model f. Or, in an image classification context, to produce a *misclassification of the label* or class predicted by the target model f.

Goodfellow et al.[75] paper expands on the Szegedy et al. work. It described that neural networks such as Long short-term memory (LSTM)[76], rectified linear units(ReLUs)[77], and maxout networks [78] are depicted to be linear and that non-linear networks like sig-

moid are tuned to spend the majority of time in the linear *regime*. This behavior reinforced the idea that perturbations based on a linear model will apply to neural networks, and the Fast Gradient Sign Method (FGSM) was introduced. In addition, it demonstrated that the direction of the perturbation is more important than the specific points in the space, which reinforced the idea that adversarial perturbations generalize across different models.

The type of adversarial attack depends on the knowledge about the target model, how the attack was built, the data used or not to compute the adversarial perturbation, its frequency, and its architecture, among others [2, 79, 80, 81]. The concepts presented here are to help understand the classification of adversarial attacks.

The *White-Box attacks* [82] refers to when the adversary has complete knowledge about the target model f to create the *adversary perturbation*, such as the training data (including the data distribution), the training algorithm, the architecture (type of the neural network, layers l, and activation functions σ), the parameters θ.

In *Black-Box attacks* [81], contrary to white-box attacks, the adversary has limited or no knowledge of the target model f. This restriction provokes the design of the attack to focus on the input query, the predicted labels, and its confidence scores. The following black-box attack variants that rely on having some information about the target model f are sometimes called *grey-box attacks*.

The *Non-Adaptative Black-Box attack* [82] is where the adversary only has access to the target model's data distribution f. To create the adversarial example using the data distribution, the adversary selects a new model architecture f' and how to train it. Now, with full knowledge of the target model f', the adversary can use white-box strategies.

The *Adaptative Black-Box attack* [82] strategy is to obtain a dataset with sets of (x, y) by querying arbitrary inputs $x = (x_1, ..., x_n)$ to the target model f and obtaining the predicted label y. The adversary follows to select a new model architecture f' and how to train it to apply white-box attack strategies then to create the *adversarial examples*.

Strict Black-Box attack [82] is a strategy where the data distribution is unknown, however as in the adaptative black-box dataset can be obtained by having a set of tuples (x, y), but here the adversary cannot modify the inputs $x = (x_1, ..., x_n)$ to watch how the output y changes.

Adversarial attacks can also be classified beyond the information available and focus more on the nature of the attack, the following terms can be considered as a classification within the white-box and black box attacks.

It is referred to as a *targeted misclassification attack* [79, 81, 82] when the strategy is to create adversarial examples that, when inputted to the target model f, force this to predict a specific output (label or class), i.e., any input image will cause the model to predict the same class or label every time. In contrast, an *untargeted or misclassification attack* [79, 81, 82] strategy is to produce the adversarial example and make the model misclassified without a specific target.

The *source/target misclassification attack* [82] is a strategy where the adversary produces a specific adversarial example to make the target model f to predict a specific output (label or class), i.e., the an adversarial example is the image of a dog, which is designed to make the target model predict a cat label.

The type of adversarial attack also depends on the steps required by the algorithm to produce the example. *One-shot attacks* [80, 79, 2] only have one calculation to create the adversarial examples, i.e., FGSM. In contrast, *iterative attacks* [80, 79, 2] need more than one iteration to create the adversarial examples, i.e., I-FGSM.

The attacks that imply adding a different adversarial perturbation to each original input are *specific perturbation attacks* [79]. In contrast, attacks that add the same adversarial perturbation to all the test inputs are *universal perturbation attacks* [83]. Moreover, the attack's time also defines the attack type; *evasion attacks* [82, 2] are performed when the target model f is in testing (prediction time). The *poisoning attacks* [82, 2] occur on the training data and in the training phase.

2.3.1 L-BFGS

Box-constrained L-BFGS attack proposed by Szedegy et al. [13] , is considered the first adversarial attack and is a White-box, targeted, and one-shot attack. It is based on the idea that there are small perturbations that, when added to a test image $\mathbf{x}_o \, \epsilon \, \mathbb{R}^m$, the machine learning model (DNN) will misclassify. To find this perturbation $\delta \, \epsilon \, \mathbb{R}^m$, an optimization problem is presented in Equation 2.27.

$$\min_{\delta} ||\delta||_2 \quad \text{s.t } f(\mathbf{x}_o + \delta) = l; \quad \mathbf{x}_o + \delta \, \epsilon \, [0,1]^m \tag{2.27}$$

Where f represents the the classifier (DNN) and l the label of the image. To find a solution where the label l is different from the ground truth label of the test image $\mathbf{x}_o$, it was proposed to use a box-constraint L-BFGS [84] algorithm and incorporates the loss function of the classifier $\mathcal{L}()$ as shown in Equation 2.28.

$$\min_{\delta} \mathbf{c}|\delta| \; + \; \mathcal{L}(\mathbf{x}_o + \delta, l) \quad \text{s.t } \mathbf{x}_o + \delta \, \epsilon \, [0,1]^m \tag{2.28}$$

The goal is to find the minimum value of c when $c > 0$, and the minimizer δ have to satisfied the condition $f(\mathbf{x}_o + \delta) = l$.

2.3.2 Deep Fool

Deep Fool [85] attack focuses on fooling deep learning models by creating more minor perturbations than other methods, i.e., FGSM [75, 86]. Securing a greater 'fooling' ratio makes it a more robust attack. Moosavi-Dezfooli et al. defined a multiclass classifier $f(x)$ based on a one-vs-all linear classification scheme, where $f(x) = \mathbf{W}^T x + b$, and the outcome of this scheme is $\hat{k}$ and the representation of the perturbation δ is shown in Equation 2.29.

$$\underset{\delta}{\text{argmin}} ||\delta||_2 \quad \text{s.t. } \exists k : w_k^T (x_{0+\delta}) + b_k \; \geqslant \; w_{\hat{k}(x_0)}^T (x_{0+\delta}) + b_{\hat{k}(x_0)} \tag{2.29}$$

Where k represent the number of column from $w_k \, \epsilon \, \mathbf{W}$. The representation of the problem established that when $f(x)$ is applied over x_0 or I_o, this exists within the area of a polyhedron P defined as in Equation 2.30.

$$P \; = \; \bigcap_{k=1}^{c} \{x : f(x)_{\hat{k}(x_0)} \geqslant f_k(x)\} \tag{2.30}$$

This area represents the space of the label $\hat{k}(x_0)$ and the objective change to measure the distance between x_0 and "the complement of the convex polyhedron P" (Moosavi-Dezfooli,

p.4). To find δ, $\hat{l}(x_0)$ is defined as the hyperplane of Pthat is closest to x_0 as in Equation 2.31.

$$\hat{l}(x_0) = \underset{k \neq \hat{k}(x_0)}{\operatorname{argmin}} \frac{|f_k(x) - f(x)_{\hat{k}(x_0)}|}{||w_k - w_{\hat{k}(x_0)}||_2} \tag{2.31}$$

Then the perturbation δ will be the vector that projects x_0 on the hyperplane $\hat{l}(x_0)$ as defines in the Equation 2.32.

$$\delta(x_0) = \frac{|f_{\hat{l}}(x_0) - f(x)_{\hat{k}(x_0)}|}{||w_{\hat{l}(x_0)} - w_{\hat{k}(x_0)}||_2^2}(w_{\hat{l}(x_0)} - w_{\hat{k}(x_0)}) \tag{2.32}$$

This method is considered a white-box, non-targeted, and iterative attack. In a more straightforward form, the strategy is to produce, in multiple steps, small perturbations that are added to x_0 to push it to the limit of the boundary from the polyhedron P.

2.3.3 AdvGAN

AdvGAN [87] is an attack composed of Generative Adversarial Networks (GANs). It has three parts; the generator $\mathcal{G}$, the discriminator $\mathcal{D}$, and the target model f (neural network). The generator $\mathcal{G}$ creates an adversarial perturbation $\mathcal{G}(x_0)$ by taking the original input x_0. The perturbation is added to the original input, $x_0 + \mathcal{G}(x_0)$ creating the adversarial example and goes into the discriminator $\mathcal{D}$ in conjunction with the original image x_0, to differentiate between the adversarial example and the original image; to create, with each new step, adversarial perturbations that make adversarial examples as similar as possible to the original image.

The first phase uses a white-box attack strategy to make the target model f fail in its prediction. The target f takes as input the adversarial example $x_0 + \mathcal{G}(x_0)$ and outputs the loss function $\mathcal{L}_{adv}^f$ as in Equation 2.33.

$$\mathcal{L}_{adv}^f = \mathbb{E}_{x_0}\mathcal{L}_f(x_0 + \mathcal{G}(x_0), t) \tag{2.33}$$

Where the original loss function $\mathcal{L}_f$ used to train the model f is incorporated, and t refers to the target class. $\mathcal{L}_{adv}^f$ in a targeted attack type is the distance between the target class/label and the prediction. The second part is the adversarial loss function as shown in Equation 2.34.

$$\mathcal{L}_{GAN} = \mathbb{E}_x \log \mathcal{D}(x_0) + \mathbb{E}_x \log 1 - \mathcal{D}(x_0 + \mathcal{G}(x_0)) \tag{2.34}$$

As mentioned before the discriminator $\mathcal{D}$ task here is to identify which is the perturbation and which is the original image. The third part is the soft hinge loss function (Equation 2.35 based on the L_2 to restrict the magnitude of the adversarial perturbation.

$$\mathcal{L}_{hinge} = \mathbb{E}_x \max 0, ||\mathcal{G}(x)||_2 - c \tag{2.35}$$

Where the constant c is the bound specified by the user. The objective or loss function applied to train the model takes the three parts $\mathcal{L}_{adv}^f$, $\mathcal{L}_{GAN}$, $\mathcal{L}_{hinge}$ under consideration as shown in Equation 2.36.

$$\mathcal{L} = \mathcal{L}_{adv}^f + \alpha\mathcal{L}_{GAN} + \beta\mathcal{L}_{hinge} \tag{2.36}$$

And adds two constants, α and β, to give less or more importance to each objective, as the user defines. To obtain $\mathcal{G}$ and $\mathcal{D}$, the min-max "game" $\arg\min_{\mathcal{G}} \max_{\mathcal{D}} \mathcal{L}$ needs to be solved. Finally, the attack is considered a white-box, iterative, and targeted attack.

2.3.4 BPDA

BPDA stands for backward pass differentiable approximation; this attack is a black-box, targeted, and iterative attack, and its strategy is the *gradient approximation*. The target model, a neural network, is $f(\cdot) = f^{1,\dots,j}(\cdot)$ and $f^i(\cdot)$ is on of its non-differentiable layers. As the gradient of the target model $\bigtriangledown_x f(x)$ cannot be obtained, to approximate it, a differentiable approximation, $g(x) \approx f^i(x)$, must be found. The first step to approximate $\bigtriangledown_x f(x)$ is to perform a forward pass in the target model $f(\cdot)$; it could be seen as inputting a test image in an image classifier model. Then, the backward pass is performed with the approximation $g(x)$ instead of $f^i(x)$.

The Equation 2.37 can define the general representation of the idea.

$$\bigtriangledown_x f(g(x))|_{x=\hat{x}} \approx \bigtriangledown_x f(x)|_{x=g(\hat{x})} \tag{2.37}$$

This method requires several iterations to make the gradient approximation more similar to the original. A white-box attack can be performed with the approximate gradient of the target model $f(\cdot)$.

2.3.5 One-Pixel

The One-Pixel [88] is a black-box, non-targeted, and iterative attack. The main idea is to use a differential evolution algorithm to produce an adversarial perturbation of the size of a single pixel. In this context,there is a target model f that has a n-dimensional input $x = (x_0, ..., x_n)$, and has it corresponding class t. In addition, there is also an adversarial perturbation vector $\delta(x) = (\delta_0, ..., \delta_n)$, and the objective is to find the optimized solution to $\delta(x)^*$ as shown in Equation 2.38

$$\max_{\delta(x)^*} \quad f_{adv}(x + \delta(x)), \quad \text{s.t.} ||\delta(x)||_0 \leqslant d \tag{2.38}$$

Where d refers to the number of dimensions to be modified from vector $\delta(x)$, in this case $d = 1$, which means that only one dimension will be changed, and the rest of the vector $\delta(x)$ will be zeros.

The differential evolution (DE) algorithm is part of the evolutionary algorithms, aiming to reach the fittest individual in a population. This is done by generating sets of new solutions (children) based on the current individuals (parents); if the children are more fitted than the parents, these survived and now are the current population. This approach does not need the target model's gradient or loss function information f.

The adversarial perturbation is the candidate solution optimized to *evolve* in the DE algorithm. The candidate solution has a constraint on the number of perturbations, and the perturbations are tuples conformed by five elements, which are the value of the three RGB channels and the coordinates x and y. The DE algorithm starts with 400 candidate solutions and creates another 400 candidate solutions on each iteration with the Equation 2.39.

$$x_i(g+1) = x_{r1}(g) + F(x_{r2}(g) - x_{r3}(g)) \tag{2.39}$$

Where $r1, r2$ and $r3$ are random numbers that satisfied $r1 \neq r2 \neq r3$, x_i is an individual (children) of the candidate solution, F is a scale parameter set to 0.5 and g the current iteration. The stop criterion of the algorithm is 100 iterations or an early stop when the probability label of the ground truth class/label is lower than 5%. The ground truth label/class is compared with the most non-tru class/label to evaluate the attack's success.

3 Theory

This chapter explains the theory behind the visual localization method and the adversarial attacks selected. The ACE (Accelerated Coordinate Encoding) model, the Fast Gradient Sign Method(FGSM), and the Iterative Fast Gradient Methods (I-FGSM) were chosen to implement an adversarial attack on a visual localization context. Usually, VL methods are constructed from various modules, like hierarchical localization [53], which create a non-direct workflow, causing the implementation of the attack to be more complex. Conversely, ACE has an architecture that could be considered a direct workflow. In addition, FGSM is regarded as the first-ever adversarial attack, with a clever and easy-to-exploit approach; only access to the gradient is needed, and I-FGSM represents the following stage of FGSM. They are also representative (ACE is considered a state-of-the-art VL model) and relevant. In addition to the simplicity of implementation, this was the criterion for choosing the adversarial attack and the VL model mentioned.

3.1 ACE

ACE stands for Accelerated Coordinate Encoding[89]. It is a visual (re)localization method whose functionality relies on a *scene coordinate regression* system. This method is faster than other methods regarding training and relocating images. [53, 57, 90], without sacrificing accuracy. Figure 3.1 shows the model's architecture (and pipeline). It includes a CNN coupled with an MLP and train with a projection loss.

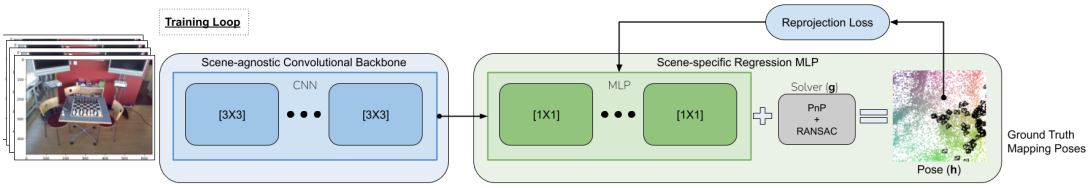


Figure 3.1: ACE architecture and pipeline.

The backbone is the CNN taken from the DSAC [57] model; this is constructed with several convolutional layers of dimension [3x3] and [1X1] that have a stride of 2, a ReLU activation function and some skip connection within. The ACE model takes the first layers of

the CNN , which extracts high-dimensional feature vectors and, as mentioned before, has on top a Scene-specific Regression MLP that predicts the scene coordinates based on the feature vectors.

The objective is to predict the camera pose **h** with only one RGB image I. The camera pose can be described as in Equation 3.1.

$$\mathbf{h} = g(C), \quad \text{with} \quad C = (x_i.y_i) \tag{3.1}$$

Equation 3.1 presents that the camera pose can be estimated if a set of 2D-3D correspondences contained in C is passed through the function g, a PnP solver within a RANSAC loop. The correspondences refer to the 2D pixel positions x_i and the 3D scene coordinates y_i. A scene coordinate regression is made with a neural network(f) to obtain the correspondence to predict the 3-D scene points for a 2D image location. 3-D scene points y_i are described as in Equation 3.2.

$$y_i = f(p_i; w) \quad \text{with} \quad p_i = P(x_i, I) \tag{3.2}$$

The parameters of the NN are the weights w and p_i that is an image patch drawn out from the pixel position x_i of the image I. These patches are 1024 randomly selected from the CNN output with their corresponding descriptor. The input of the network is the grayscale of the RGB image, which results in a patch with dimension $C_I = 1, H_P = W_P = 81PX$

Equation 3.3 is optimized to learn the function f. The optimization considers all images I_M, and the ground truth poses $\mathbf{h}_i$. Minibatch SGD is used. $\mathcal{L}_\pi$ is the projection loss.

$$\arg\min_{\mathbf{w}} \sum_{I \in \mathcal{L}_M} \sum_i \mathcal{L}_\pi \left[\mathbf{x}_i, \underbrace{f(\mathbf{p}_i; \mathbf{w})}_{y_i}, \mathbf{h}_i \right] \tag{3.3}$$

Equation 3.4 defines the reprojection training loss. It aims to do end-to-end training with a dynamic inlier threshold that starts loose and tightens as the training continues. This approach forces the network to focus on good 3D predictions and discard the ones the RANSAC loop would remove in the latter phase.

$$\mathcal{L}_\pi[\mathbf{x}_i, y_i, \mathbf{h}_i] = \begin{cases} \hat{e}_\pi(\mathbf{x}_i, y_i, \mathbf{h}_i) & \text{if } y_i \in \mathcal{V} \\ \frac{1}{\|y_i - \overline{y}_i\|_0} & \text{otherwise.} \end{cases} \tag{3.4}$$

Where $\hat{e}_\pi$ is the reprojection error for the valid coordinates predictions $\mathcal{V}$. These predictions are within 0.010m and 1000m and have a e_π smaller than 10000px. For the invalid predictions, the distance is subtracted against a dummy scene coordinate $\overline{y_i}$.
To define the reprojection error $\hat{e}_\pi$ of equation 3.4, tanh clamping is applied to the reprojection error e_π(Equation 3.6). Tanh is re-scale with the threshold τ, which changes with the training process(Equation **??**).

$$\hat{e}_\pi(x_i, y_i, \mathbf{h}_i) = \tau(t) \tanh\left(\frac{e_\pi(x_i, y_i, \mathbf{h}_i)}{\tau(t)}\right) \tag{3.5}$$

$$\tau(t) = w(t)\, \tau_{\max} + \tau_{\min}, \quad \text{with} \quad w(t) = \sqrt{1 - t^2} \tag{3.6}$$

3.2 RANSAC and PnP

The Random Sample Consensus (RANSAC) is a method to find the best set of data points within experimental data for fitting a model [91]. There is a set of data points P, from which a subset S_1 with n data points is randomly selected, where the number of data points on P is greater than n. The model M_1 is fitted with the subset S_1, then M_1 is used to know the subset of points $\hat{S}_1$ in P that fall into the error tolerance. This subset $\hat{S}_1$ is a consensus set.

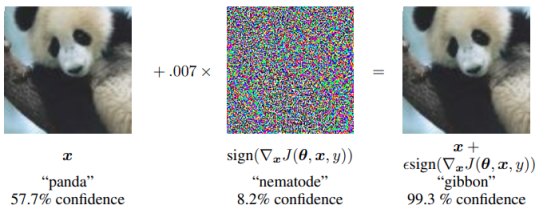


Figure 3.2: Adversarial example applied to model GoogLeNet[56, 75]. The left image is an image correctly classified as a panda, the center image is the adversarial perturbation added to the left image, and the right image is the adversarial example composed by adding the perturbation to the correctly classified image.

Subsets S of data points are selected iteratively. A new model $\hat{M}_1$ is computed for each subset. The threshold t is defined and used as criteria to determine if a data point is considered an inlier for the computed model $\hat{M}_1$. Several models are computed with different subsets of data points, and the model with the highest number of inliers is selected.

In visual localization methods, 3 or 4 sets of point correspondences, depending on the PnP solver [92], of 3D-2D points correspondences are selected randomly and sent to the PnP solver to estimate the pose. The next step is to project the predicted 3D scene coordinate back to the plane of the image using the estimated pose. The projection error is computed by comparing the 2D derived from the projection and the actual 2D points from the set of 3D-2D correspondences. It is an inlier if the correspondence has a reprojection error smaller than a previously defined threshold. Multiple poses are estimated within the RANSAC loop and are ranked concerning the number of inliers.

3.3 Fast Gradient Sign Method (FGSM)

The Fast Gradient Sign Method relies on the linear nature of the Neural Networks in high-dimensional spaces [75]. The general formula of the attack is shown below. δ is refer as the perturbation, L is the cost or loss function that depends on, $f(\mathbf{x})$ the models prediction and y the target of the model. ϵ is a constant that changes the magnitude of the perturbation. Figure 3.2 presents the image x of a panda; with the addition of the adversarial perturbation, this perturbation would be classified as a nematode. The result is an image with imperceptible changes to the original, $\mathbf{y}$, that the model classifies as "gibbon" with high confidence. The adversarial perturbation is defined in Equation 3.7.

$$\delta = \epsilon sign(\nabla_x L(\theta, f(\mathbf{x}), y)) \tag{3.7}$$

3.4 I-FGSM

The Iterative Fast Gradient Method (I-FGSM) is an extension of the Fast Gradient Sign Method introduced in section 3.3. The idea is to apply several times the FGSM using a small step size α, starting by defining the adversarial example 0 as $\mathbf{X}_0^{adv} = \mathbf{X}$, where $\mathbf{X}$ is the original image. The next iteration that produces the following intermediate adversarial example is given by Equation 3.8.

$$\mathbf{X}_{N+1}^{adv} = Clip_{X,\epsilon}\{\mathbf{X}_N^{adv} + \alpha \text{sign}(\bigtriangledown_X \mathcal{L}(f(\mathbf{X}_N^{adv}), y))\} \tag{3.8}$$

Where the prediction of the model $f(\cdot)$ based on the adversarial example from the last iteration, $\mathbf{X}_N^{adv}$ and the true target y are needed to compute the sign function of the gradient from the adversarial example input. Kurakin et al. [86] used $\alpha = 1$, and the number of

iterations was selected heuristically, $(\epsilon^I + 4, 1.25\epsilon^I)$, but it can be defined as the user decides. The clip function is defined as in Equation 3.9.

$$Clip_{X,\epsilon^I}\{\hat{X}\}(x,y,z) = \min\{255, X(x,y,z) + \epsilon^I, \max\{0, \mathbf{x}(x,y,x) - \epsilon^I, \hat{X}(x,y,z)\}\} \quad (3.9)$$

Where $\mathbf{X}(x,y,z)$ refers to the value of channel z from the last image at the coordinates (x,y), with the primary objective of maintain the resulting tensor within the $\mathcal{L}_\infty\epsilon -$ neighborhood of the image **X**.

3.4.1 Interquartile range

The box plot represents the interquartile range[93] to analyze the value change from the components of the quaternion and translational vector of the pose. The interquartile range (IQR) is the distance between the upper and lower quartiles, which the Equation 3.10 presents.

$$\text{IQR} = q_n(0.75) - q_n(0.25), \quad (3.10)$$

where $q_n(0.75)$ is the 75th percentile (upper quartile) and $q_n(0.25)$ is the 25th percentile (lower quartile).

The box plot in Figure 3.3 helps to visualize the IQR, and it is composed of the median, the first quartile (Q1), the third quartile (Q3), the lower and upper whiskers, whose lengths are computed with Equations 3.11 and 3.12. These help identify all the data points outside the range as outliers.

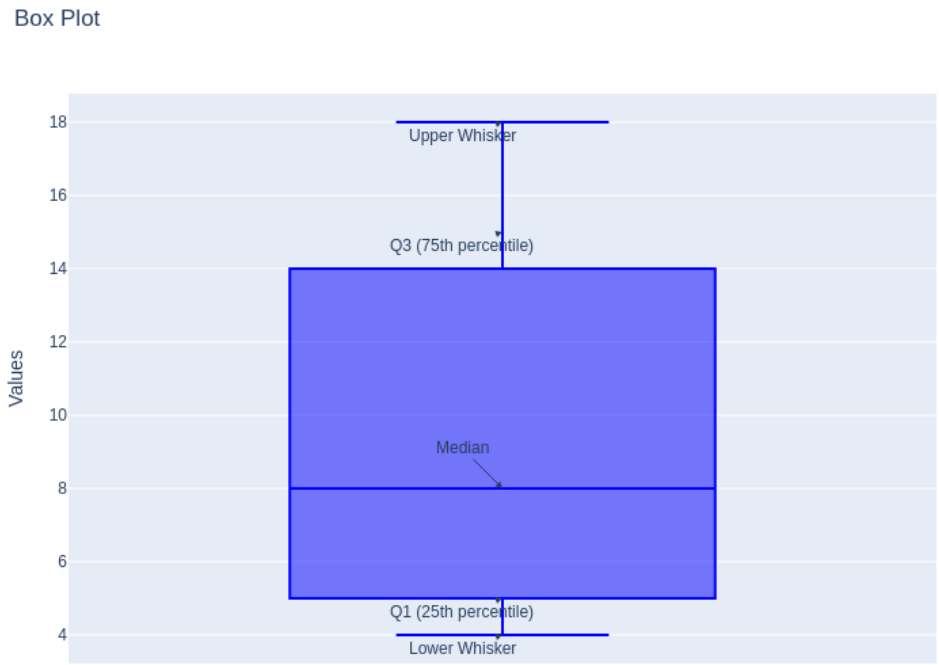


Figure 3.3: Box plot with the five-number summary obtained through a python script.

$$Lower\ \ whisker = Q1 - 1.5 \times IQR \quad (3.11)$$

$$Upper\ \ whisker = Q3 + 1.5 \times IQR \tag{3.12}$$

3.4.2 Multiple Linear Regression

The multiple linear regression [94] is introduced to interpret how much *damage* the adversarial examples induced in the ACE model by analyzing what variables describe the average error's behavior. In particular, to review if the epsilon's ϵ, ϵ^I and the iteration step N magnitude change affects the average error. In addition, it will reveal whether the model's error correlates to the attacks performed.

Scenarios where there is only one predictor variable correspond to the simple linear regression [94] shown in Equation 3.13.

$$Y \ = \ \beta_0 \ + \ \beta_1 X_1 + \epsilon \tag{3.13}$$

Where β_0 is the intercept to the origin (Y when X=0) and β_1 is the average increase of Y with respect to the increase of one unit of the predictor variable $\mathbf{X}$, also called the slope, and the error is $\epsilon \sim N(0,1)$. The thesis approach considers multiple linear regression, as there is more than one predictor variable; for this, the equation changes as shown in Equation 3.14.

$$Y \ = \ \beta_0 \ + \ \beta_1 X_1 \ + \ \beta_2 X_2 \ + \ ... \ + \ \beta_i X_i + \epsilon \tag{3.14}$$

Where X_i is the ith predictor and the coefficient β_i is the average increase of Y when there is a one unit increase in X_j, maintaining the other predictors fixed.

The values of the coefficients $\beta_{1,...,i}$ are chosen to minimize the sum of squared residuals in Equation 3.15

$$RSS \ = \ \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.15}$$

By substituting the prediction $\hat{y}_i$ the Equation 3.16 is obtained.

$$RSS \ = \ \sum_{i=1}^{n} (y_i - \hat{\beta_0} \ + \ \hat{\beta_1} X_1 \ + \ \hat{\beta_2} X_2 \ + \ ... \ + \ \hat{\beta_i} X_i)^2 \tag{3.16}$$

The coefficients $\hat{\beta}_{1,...,i}$ refer to the multiple least squares regression coefficient estimates.

The p-value is a metric used to determine if there is an association between the response and the predictor variable; the convention is that if $p - \text{value} < 0.05$, it indicates that there is a relation between Y and $\mathbf{X}_{0,...,i}$. In addition, there are more support metrics to know if there is a relation between the predictor variables and the response. Supposed the hypothesis test with the null hypothesis is that all the coefficients are equal to 0 as shown in Equation 3.17.

$$H_0 : \beta_1 = \beta_2 = ... = \beta_i = 0 \tag{3.17}$$

And the alternative hypothesis is shown in Equation 3.18

$$H_a : \text{at least one } \beta_i \text{ is} \neq 0 \tag{3.18}$$

The F-statistic comes into play to know which hypothesis is true by applying Equation 3.19

$$F = \frac{(TSS - RSS)/p}{RSS/(n-p-1)} \tag{3.19}$$

Where the total sum of squares is $TSS = \sum (y_i - \bar{y})^2$ and the residual sum of squares is $RSS = \sum (y_i - \hat{y})^2$. TSS considers the sample mean $\bar{y}$ and RSS the population mean $\hat{y}$. If H_0 is

true, which means that there is no relation between the predictors and the response variable, then the value of the F-statistic is around one and would imply that Equation **??** holds.

$$\mathbb{E}[\frac{TSS - RSS}{p}] = \sigma^2 \tag{3.20}$$

If H_a is true, then at least one of the predictor variables is non-zero, and the value of the F-statistic is greater than one. Therefore the Equation 3.21 is been satisfied.

$$\mathbb{E}[\frac{TSS - RSS}{p}] => \sigma^2 \tag{3.21}$$

The criteria for how close to or greater than one the F-statistic should depend on the number of samples n and predictors p. If n is large, then a value slightly larger than one will mean to reject H_0; if n is small, then the F-statistic would need to be a larger value to reject H_0. The R^2 statistic in the simple linear regression setting measures the fit of the model; it explains what proportion of the variance from the variable Y is explained by X; if the value is closer to one, then it explains more about the variance of Y; and is defined as in Equation 3.22.

$$R^2 = \frac{TSS - RSS}{TSS} \ = \ 1 - \frac{RSS}{TSS}, \ 0 \leqslant R^2 \leqslant 1 \tag{3.22}$$

For the simple linear regression case, the R^2 statistic is equivalent to the correlation, because both describe the relation between Y and X, where the correlation is defined as in Equation 3.23.

$$Cor(X, Y) \ = \ \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{(x_i - \bar{x})^2(y_i - \bar{y})^2}} \tag{3.23}$$

When the problem is in a multiple regression setting, the R^2 statistic is equal to the square of the correlation between the response variable and the fitted linear model, $Cor(Y, \hat{Y})^2$.

A variation of the R^2 is the adjusted R^2 from Equation 3.24[94],

$$\text{Adjusted } \ R^2 = 1 - \frac{\frac{RSS}{n-d-1}}{\frac{TSS}{n-1}}, \tag{3.24}$$

where d is the number of variables and n the number of observations. When the number of variables increases, R^2 increases; in contrast, adjusted R^2 could decrease or increase because the number of variables d is included in the numerator. The reason is that including variables that do not explain the variance of the target variable will decrease the adjusted R^2 while adding variables helpful to explain the variance will increase the R^2. A suitable interpretation of R^2 and adjusted R^2 values is if $R^2 \gg adj.R^2$ the regression has predictor variables that do not contribute to explaining the variance of the model, whether if $R^2 \approx adj.R^2$ most of the predictor variables are meaningful.

3.4.3 Polynomial Regression

Polynomial regression[94] is an extension of linear regression in which the predictor variables and the target or response variable have a nonlinear relationship. It follows the structure from Equation 3.25,

$$Y_i \ = \ \beta_0 \ + \ \beta_1 X_i \ + \ \beta_2 X_i^2 \ + \ ... \ + \ \beta_i X_i^d + \epsilon_i, \tag{3.25}$$

where d is the degree of the polynomial and ϵ_i the error term. As with the multiple linear regression, the values of the coefficients from Equation 3.25 can be computed with least square linear regression.

4 Data

A review of the data used to generate the adversarial attacks is introduced in this chapter. Section 4.1 covers all the contents of the dataset. Section 4.2 describes the process of how the data was obtained. Section 4.3 presents which data was applied to pre-train the ACE model and which was selected to produce the adversarial attacks.

4.1 Dataset Overview

The 7-Scenes [95] dataset from Microsoft[1] was chosen to produce the adversarial attacks. It includes RGB-D frames from the seven indoor scenes: Chess, Fire, Heads, Office, Pumpkin, RedKitchen, and Stairs. As shown in Figure 4.1. Each scene has a different number of sequences used for training and testing. And each sequence has between 500 and 1000 frames. Three files represent a frame:

- Color: A 24-bit RGB image in .png format with a 640x480 resolution.
- Depth: A 16-bit black and white image in .png format representing the depth in millimeters, i.e., Figure 4.2 shows the depth image related to frame 0 from sequence 3 of the chess scene.
- Pose: A text file, .txt, containing the ground truth pose of the frame, represented in homogeneous coordinates in a 4x4 matrix, i.e., the matrix in Equation 4.1 is the homogeneous coordinates from sequence 3 and frame 0 from the chess scene.
- Calibration: A text file, .txt, containing the focal length, which is the same for all images. The center of the image is supposed to be the principal point. This file is not part of the original dataset that Microsoft provides; it is added as part of the ACE model replied in this thesis.

$$\begin{bmatrix} 0.9965 & 0.0250 & 0.0791 & -0.0260 \\ -0.0524 & 0.9292 & 0.3656 & -0.7870 \\ -0.0644 & -0.3685 & 0.9273 & -0.7046 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{4.1}$$

[1] https://www.microsoft.com/en-us/research/project/rgb-d-dataset-7-scenes/

Figure 4.1: Image samples from each scene of the 7-scenes dataset.

Figure 4.2: Depth Image sample from sequence 3, frame 0 of chess scene.

4.2 Data Extraction

The camera tracks and 3D model of the sequences were acquired through an implementation of the *KinectFusion* [96]. The Kinect system has video and infrared cameras in the middle, an infrared projector on the left side, and four microphones along the structure. The video camera sensor is used to obtain the RGB images. The Kinect uses the infrared projector as a light source to obtain the tracked sequence's depth; it emits infrared light that is projected on objects as dots, and the reflection of these is analyzed to determine the distance of the objects. The black-and-white images from the dataset represent distance values rather than color information.

4.3 Data Selection

This thesis uses the ACE model to produce the adversarial attacks and then performs these on the same model, but no training was performed. However, this section acknowledges what data was used for its training, trained by Niantic Labs footnote[2], to have a better context. The Table 4.1 from Figure 4.1 contains the sequences used for the training and test of each scene and the number of frames of each scenario.

The weights of the pre-trained ACE heads (Multi-Layer Perceptron) for each scene are available[2] in conjunction with the weights of the pre-trained encoder (Convolutional Neural Network).

[2]https://github.com/nianticlabs/ace

Scene	Train Sequences	Test Sequences	Train frames qty.	Test frames qty.
Chess	1, 2, 4, 6	3, 5	4000	2000
Fire	1, 2	3, 4	2000	2000
Heads	2	1	1000	1000
Office	1, 4, 3, 5, 8, 10	2, 6, 7, 9	6000	4000
Pumpkin	2, 3, 6, 8	1, 7	4000	2000
RedKitchen	1, 2, 5, 7, 8, 11, 13	3, 4, 6, 12, 14	7000	5000
Stairs	2, 3, 5, 6	1, 4	2000	1000

Table 4.1: Sequences used in training and test, and the number of frames of each scene.

4.3.1 Data applied

Twenty-two frames from the Chess scene were selected as the base images to produce the adversarial attacks, eleven from test sequence 03 and eleven from sequence 05. Frames 0 to 10 of each sequence, i.e., image samples from the frames shown in Figure 4.3. Using the ACE model in test mode requires the RGB, Pose, and Calibration files. Depth files are only needed if the ground truth pose needs to be computed, but they are already available. In addition, the weight of the pre-trained head of the Chess scene *7scenes_chess.pt*[3] was loaded.

Figure 4.3: Examples of images selected to produce the adversarial examples

[3]https://storage.googleapis.com/niantic-lon-static/research/ace/ace_models.tar.gz

5 Method

5.1 Workflow

To know where to compute the needed elements of the adversarial attacks, the workflow of the training phase from the ACE model is represented in Figure 3.1. The blue rectangles are the scene-agnostic Convolutional Backbone built from 3x3 convolutional layers that extract the descriptors from the query image. These descriptors are shuffled and input into the scene-specific regression MLP head built from 1x1 layers. This MLP predicts the 3D coordinates, which go into the solver **g**, conformed by a PnP and a RANSAC loop that estimates the pose. Finally, error and reprojection loss are computed from the estimated pose (**h**) and its ground truth pose.

To apply the chosen adversarial attacks, the 3D coordinates prediction from the MLP head is taken, and in conjunction with the ground truth coordinates of the corresponding input image, the loss functions $\mathcal{L}_1$ and $\mathcal{L}_2$ [97, 98] are computed (Figure 5.1) as shown in Equation 5.1.

$$\mathcal{L}_1 \; = \; ||y - f(x)||_1, \;\; \mathcal{L}_2 = ||y - f(x)||_2^2 \tag{5.1}$$

Where y is the 3D coordinates target and $f(x)$ is the 3D coordinates prediction from the MLP of ACE. Then, the error is backpropagated through the network without updating the weights (test mode) to obtain the gradient from the input image and perform FGSM and I-FGSM.

The attacks are a combination of white-box and black-box strategies. The computation of the perturbation is based on the white-box attacks $\{FGSM, I-FGSM\}$, but the *reprojection loss function* from the ACE model was not used as it would be too complex to backpropagate the error through the RANSAC+PnP loop and the neural network (CNN + MLP). Instead, $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions were computed, even if all the information about the ACE model is known; the election to use another loss function transformed the adversarial attack into a BPDA attack, where we approximate the gradient target model $\bigtriangledown_x g(x) \approx \bigtriangledown_x f(x)$, and then white-box strategies are implemented.

Twenty pictures (frames) from the chess test scene (Figure 4.3), ten from Sequence 03, and ten from Sequence 05 were selected to build the FGSM attacks. The gradients $\bigtriangledown_x g(x)$

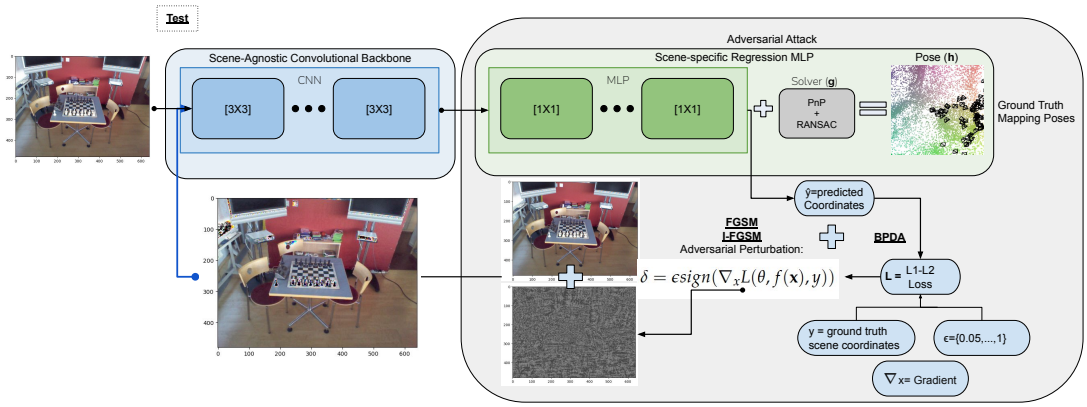


Figure 5.1: Workflow Adversarial Attacks

from the input images x were computed as described in Section 3.3. For each of the twenty original images, eleven epsilon values $\epsilon = 0, ..., 1$ were drawn incrementally to produce eleven *adversarial perturbations*, which then were added to the original images to produce the *adversarial attacks*. In total, there are 220 adversarial attacks. Finally, the adversarial attacks are taken as the input to compute and collect for each attack (image) $\mathcal{L}_1$ and $\mathcal{L}_2$ losses, the pose quaternion and translational vector, and the rotational and translational errors.

To produce the adversarial attacks with the *Iterative* FGSM, the same ten pictures from sequence 03 and ten from sequence 05 used in FGSM were used. The adversarial examples (images) were created following the formulas in Section 3.4 for each frame considering an $\alpha = 1$, eleven epsilon values $\epsilon^I = \{0, 0.1, ..., 1\}$ and twenty iteration $N = 20$. Which construct 4400 adversarial examples. Like with FGSM, the attacks were computed with $\mathcal{L}_1$ and $\mathcal{L}_2$ losses.

From attacks produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ losses, two datasets are created respectively, where the features X are the elements w, x, y, and z from the quaternion pose, the elements x, y, and z from the translational vector, and the epsilon value (Table 5.1). The target variable y is the unified magnitude of the errors defined by taking the average of both,

$$Average\ error = \frac{t_{err} + r_{err}}{2}, \tag{5.2}$$

where r_{err} is the rotational and t_{err} the translational error. For the I-FGSM attacks, Table 5.2 shows the result features of the dataset, which includes the iteration number of the attack.

epsilon ϵ	**rot**$_w$	**rot**$_x$	**rot**$_y$	**rot**$_z$	**tr**$_x$	**tr**$_y$	**tr**$_z$	**Avg_error**

Table 5.1: FGSM Dataset

epsilon ϵ	**iteration** N	**rot**$_w$	**rot**$_x$	**rot**$_y$	**rot**$_z$	**tr**$_x$	**tr**$_y$	**tr**$_z$	**Avg_error**

Table 5.2: I-FGSM Dataset

5.1.1 Code

We used the files and code from the GitHub[1] repository to attack the ACE model. This repository includes the weights of the pre-trained model for the chess scene, scripts to download

[1]https://github.com/nianticlabs/ace

the 7scenes dataset, and instructions about how to run the evaluation mode. To download the data, the following lines should be run:

```
    cd datasets
# Downloads the data to datasets/pgt_7scenes_{chess, fire, ...}
./setup_7scenes.py --poses pgt
```

5.1.1.1 FGSM

In order to generate adversarial attacks utilizing the Fast Gradient Sign Method (FGSM), we initially obtained the Python script "test_ace_fgsm.py" from the Master Thesis GitHub repository[2] and ran the script as illustrated below.

```
./test_ace_fgsm.py datasets/7scenes_chess /Downloads/7Scenes/7scenes_chess.pt
```

Here, we specify the weights of the pre-trained network of the chess scene. Inside the Python script, it can be specified which loss function ($\mathcal{L}_1$ or $\mathcal{L}_2$) to use by uncommenting lines 221-222 or 225-226. To choose the original images to produce the attacks, line 108 points to the images location. After running the script, this will produce the files that contain the gradients with respect to the input image "gradients_L1.pt" and "gradients_L2.pt." Then, the Jupyter Notebook "FGSM.ipynb"[2] should be run to produce the adversarial images.

5.1.1.2 I-FGSM

To generate adversarial attacks utilizing the Iterative Fast Gradient Sign Method (IFGSM), we initially obtained the Python script "test_ace_ifgsm.py" from the Master Thesis GitHub repository[2] and ran it as illustrated below.

```
./test_ace_ifgsm.py datasets/7scenes_chess /Downloads/7Scenes/7scenes_chess.pt
```

In line 228, you can modify the argument L1_loss to specify the type of loss function to use. This script uses the function *ifgsm* available in the file "IFGSM.py"[2] to produce the attacks. Inside this file, it can be specified where to save the adversarial examples (images) in lines 84 and 85.

After producing the adversarial example of the corresponding methods and having the folder locations. To attack the ACE model, it has to be run in the evaluation mode by running the following lines:

```
./test_ace_copy.py datasets/7scenes_chess /Downloads/7Scenes/7scenes_chess.pt
```

The script will run the attacks and save a .csv dataset containing each computed pose's quaternion and translational vector elements, as well as the rotational and translational errors.

The code mentioned in section 5.1.1 was not produced in this thesis. The code mentioned in sections 5.1.1.1 and 5.1.1.2 was produced in this thesis.

5.1.2 Regression Models

With the collected data, **eight** regression models were fitted to analyze the effect of the adversarial attacks over the average error from the ACE computed pose and consequently investigate the robustness of the ACE model when adversarial attacks are applied to it. There are two simple linear models(Equation 5.3), two multiple linear models (Equation 5.5), and four polynomial models(Equations 5.4 and 5.6). These regression models are distributed as follows:

[2]https://github.com/AlanCT94/Ace-master-thesis

5.1.3 Regression models from FGSM generated attacks

Each model structure presented in the Equations from this subsection is repeated twice, one with the data from the produced attacks with $\mathcal{L}_1$ loss function, and another with data from $\mathcal{L}_2$ loss function produced attacks.

5.1.3.1 Regression models from results of $\mathcal{L}_1$ and $\mathcal{L}_2$ produced attacks

Equation 5.3 presents a simple linear regression,

$$y_{ae} = \beta_0 + \beta_1 x_\epsilon, \tag{5.3}$$

where y_{ae} is the target variable, the average error, and the predictor variable x_ϵ refers to the ϵ value used to produce the adversarial attacks.

Equation 5.4 presents a polynomial regression of degree d,

$$y_{ae} = \beta_0 + \beta_1 x_\epsilon + , ..., + \beta_i x_\epsilon^d \tag{5.4}$$

where y_{ae} is the target variable, the average error, and the predictor variable x_ϵ refers to the ϵ value used to produce the adversarial attacks. The selection of the degree d is explained in the next subsection.

5.1.4 Regression models from I-FGSM generated attacks

5.1.4.1 Regression models from results of $\mathcal{L}_1$ and $\mathcal{L}_2$ produced attacks

Equation 5.5 presents a simple linear regression,

$$y_{ae} = \beta_0 + \beta_1 x_{\epsilon^I} + \beta_2 x_N, \tag{5.5}$$

where y_{ae} is the target variable, the average error, the predictor variable x_{ϵ^I} refers to the ϵ^I value used to produce the adversarial attacks and x_N the iteration step number.

Equation 5.6 presents a polynomial regression of degree d,

$$y_{ae} = \beta_0 + \beta_i x_{\epsilon^I} + \beta_j x_N, ..., + \beta_i x_{\epsilon^I}^d + \beta_j x_N^{d_n} \tag{5.6}$$

where y_{ae} is the target variable, the average error, the predictor variable x_{ϵ^I} refers to the ϵ^I value used to produce the adversarial attacks and x_N is the iteration step number. The selection of the degree d and d_n is explained in the next subsection.

5.1.5 Polynomial degree selection

The dataset for adversarial attacks generated by FGSM was divided into 85% training and 15% testing. For adversarial attacks generated by I-FGSM, it was divided into 80% training and 20% testing. The proportion of data used for training is higher for the FGSM polynomial regression models because that dataset is smaller than the dataset from I-FGSM.

Then, 10 polynomial regression models were fitted for FGSM attacks, and the polynomial regression of degree d with the lowest MSE(Equation 5.7) was chosen where the predictor variable is $X = \{\epsilon\}$.

For I-FGSM, 100 polynomial regression models were fitted, where the predictor variables are

$X = \{\epsilon^I, N\}$ and the polynomial regression with the combination of degree d for ϵ^I and d_n for N with the lowest MSE was chosen.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\,, \tag{5.7}$$

where $\hat{y}_i$ is the predicted average error, y_i is the ithaverage error, and n the number of observations.

6 Results

This chapter presents the results of applying FGSM and I-FGSM adversarial attacks to the ACE model, produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ losses. 6.1 presents some examples of the adversarial attacks produced by both methods. 6.2 introduces the change in the value of the rotational and translational elements of the camera pose that the ACE model computed. 6.3 shows the change in percentage of the corresponding loss value, $\mathcal{L}_1$ or $\mathcal{L}_2$, and average error generated by the FGSM and I-FGSM attacks, and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$. 6.4 covers the polynomial, simple, and multiple linear regression analysis of the FGSM and I-FGSM, divided by the loss function from which it was produced, i.e., $\mathcal{L}_1$ or$\mathcal{L}_2$ losses.

In the adversarial attacks generated with the **FGSM**, the "No Attack" scenario is when the hyperparameter $\epsilon = 0$. This represents that the *sign function* of the gradient for the image obtained when backpropagating the $\mathcal{L}_1$ or $\mathcal{L}_2$ loss through the MLP of the ACE model is multiplied by 0. This causes the adversarial perturbation to be added to the original image to create the attack to be 0. Consequently, the ACE model input when $\epsilon = 0$ is the same as the original images from the chess scene, described in Chapter 4, Section 4.3. The "Attack" scenario is when the hyperparameter $\epsilon > 0$.

In the adversarial attacks generated by the **I-FGSM**, the "No Attack" scenario refers to when the hyperparameter $\epsilon^I = 0$, and the iteration step equals 1; with these values, the ACE model has as its inputs the original images from the chess scene, which are described in Chapter 4, Section 4.3. The "Attack" scenario is when the hyperparameter $\epsilon^I > 0$ and the iteration step N is greater than one.

6.1 Adversarial Examples Images

6.1.1 FGSM Adversarial Examples

The twelve images presented in Figure 6.1 are part of the 220 adversarial examples generated with the FGSM. All images situated in **column one** are the *"No Attack"* scenario. From images of **columns two** and **three**, it can be seen that increasing the epsilon value produced images with more noise, which are visually less similar to the original. The produced adversarial examples do not present a change in the form of the objects within these. However,

there appears to be a change in the color, its pattern, and the light conditions of the frame. No significant differences exist between the adversarial examples produced with $\mathcal{L}_1$ and $\mathcal{L}_2$. Nevertheless, there are changes in the color of some pixels around small areas, i.e., comparing the area between the two televisions from the image in row one and column three and the image in row three, column three, there is a change of color in the pixels. Some pixels that appear red in the first image are blue in the second one and vice versa. These types of differences are more present as the epsilon value increases.

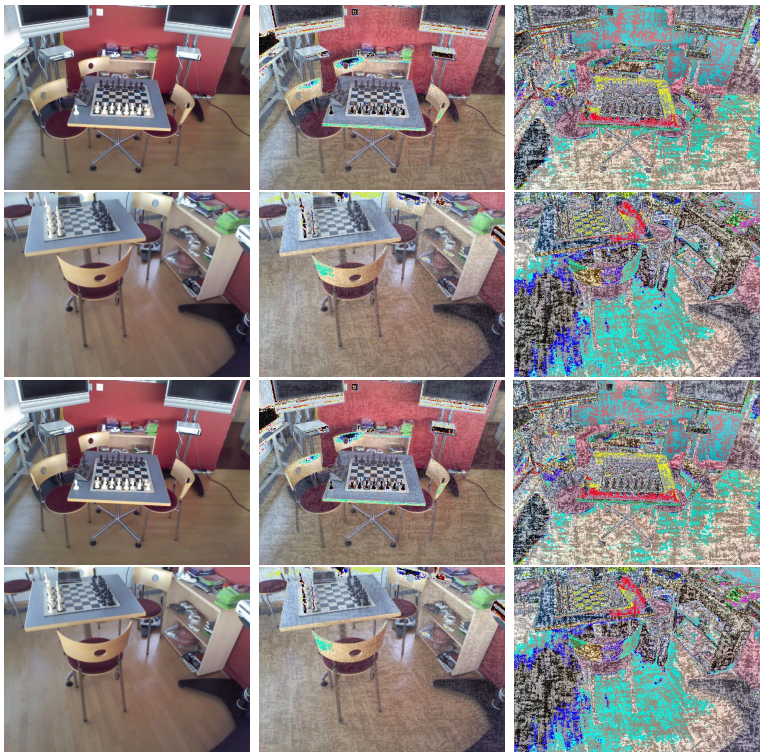


Figure 6.1: Images from adversarial examples used to make the adversarial attacks. Images from **rows one** and **two** were produced with loss function $\mathcal{L}_1$; from left to right, each image corresponds to the epsilon values $\epsilon \in \{0, 0.2, 0.8\}$. Images from **rows three** and **four** were produced with loss function $\mathcal{L}_2$; from left to right, each image corresponds to the epsilon values $\epsilon \in \{0, 0.2, 0.8\}$. Images from **rows one** and **three** are from frame zero of sequence 03. Images from **rows two** and **four** are from frame zero of sequence 05.

6.1.2 I-FGSM Adversarial Examples

Figure 6.2 and 6.3 images are part of the 4400 adversarial examples generated with the I-FGSM and correspondingly produced with the $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions. In both Figure 6.2 and 6.3, the image in **row and column one** is the *"No Attack"* scenario. The produced adversarial examples do not present a change in the form of the objects within these. Analyzing images by column, from columns two and three, it can be seen that increasing the iteration step produced images with minor modifications on the pixel's color and color pattern, almost imperceptible on a small scale. Analyzing images by row, it can be seen that increasing the epsilon value ϵ^I produced images with what could be described as more noise. No significant differences exist between the adversarial examples produced with $\mathcal{L}_1$ and $\mathcal{L}_2$. However, there are some changes in the color of some pixels around small areas, i.e., comparing the image from row three and column one from Figure 6.2 and the same image from Figure 6.3. The area to the left of the chair legs has some green and red pixels that change in pattern in

the image from Figure 6.3. Still, these types of differences are not perceptible in a small-scale image.

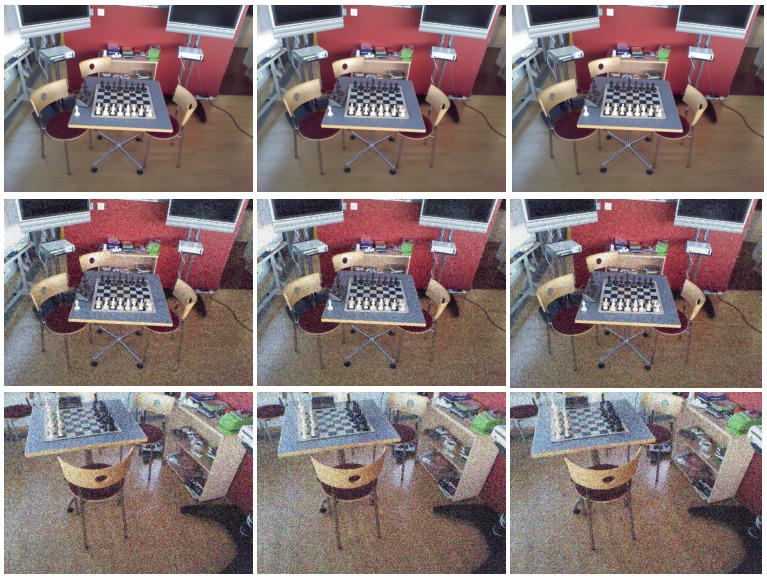


Figure 6.2: Images from adversarial examples generated with the I-FGSM and produced with the loss function $\mathcal{L}_1$ used to make the adversarial attacks. **All images** have an $\alpha = 1$. The **images** of **each row**, from left to right, correspond to the iteration step $\in \{1, 11, 20\}$. **Row one** images have an $\epsilon^I = 0$, **row two** images have an $\epsilon^I = 0.5$, and **row three** images have an $\epsilon^I = 1$. Images from **rows one** and **two** are from frame zero of sequence 03. Images from **row three** are from frame zero of sequence 05.

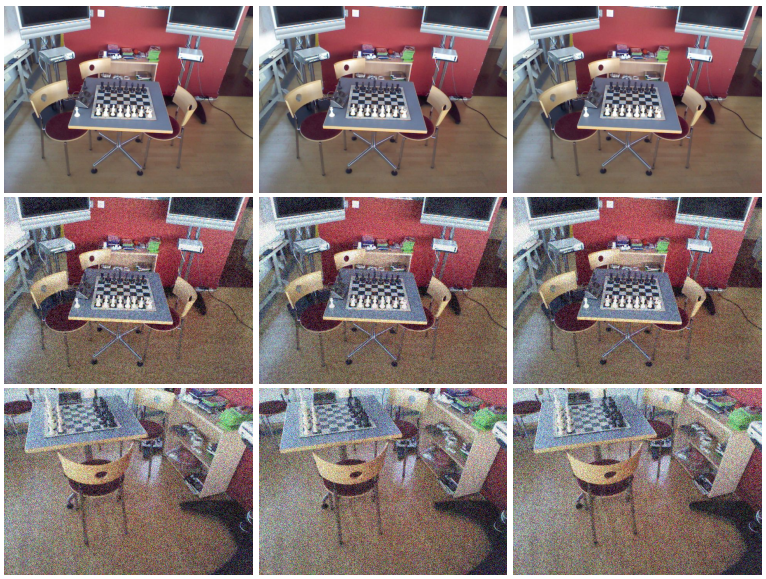


Figure 6.3: Images from adversarial examples generated with the I-FGSM and produced with the loss function $\mathcal{L}_2$ used to make the adversarial attacks. **All images** have an $\alpha = 1$. The **images** of **each row**, from left to right, correspond to the iteration step $\in \{1, 11, 20\}$. **Row one** images have an $\epsilon^I = 0$, **row two** images have an $\epsilon^I = 0.5$, and **row three** images have an $\epsilon^I = 1$. Images from **rows one** and **two** are from frame zero of sequence 03. Images from **row three** are from frame zero of sequence 05.

6.2 Rotational and translational elements of the camera pose

This section describes the change in the value of the quaternion's rotational elements and the translational vector elements that form the camera pose computed by the ACE model. First, the pose elements are presented when there is "No Attack"; this scenario is valid for both FGSM and IFGSM and loss functions. Within each section, the pose elements are analyzed by the loss function used to obtain the gradient of the "Attack" and compared to when there is "No Attack."

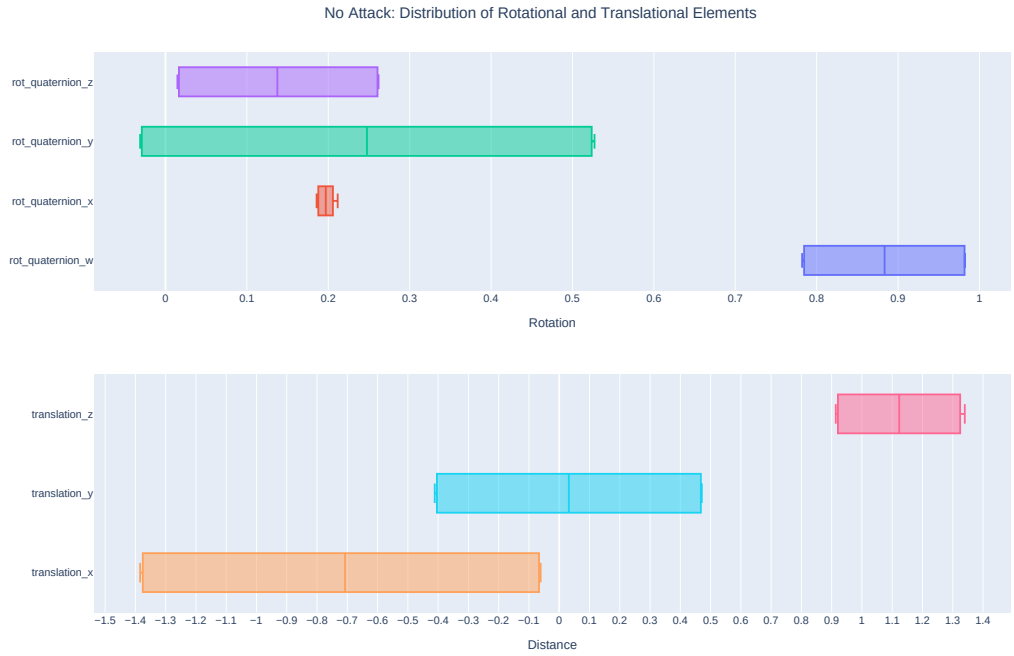


Figure 6.4: The top plot presents the values of the quaternion elements, and the bottom plot presents the value of the translational element of the camera pose computed by the ACE model when there is "No Attack."

Figure 6.4 represents that the element **z** of the **quaternion** has a median of 0.13, and 50 % of its values fall between 0.01(Q1) and 0.26(Q3). The element **y** of the **quaternion** has a median of 0.24, and 50 % of its values fall between -0.029(Q1) and 0.52(Q3). The element **x** of the **quaternion** has a median of 0.19, and 50 % of its values fall between 0.18(Q1) and 0.20(Q3). The element **w** of the **quaternion** has a median value of 0.88, and 50 % of its values fall between 0.78(Q1) and 0.98(Q3). The elements **z**, **y**, and **w** of the **quaternion** have whiskers close to the IQR box, and their median values are close to the center of the IQR box. This indicates the elements have a symmetrical distribution. Nevertheless, the right whisker from element **x** is larger than the left one, indicating a distribution skewed to the left.

Figure 6.4 shows that the element **z** of the **translational vector** has a median of 1.12, and 50 % of its values fall between 0.92(Q1) and 1.32(Q3). The element **y** of the **translational vector** has a median of 0.03, and 50 % of its values fall between -0.40(Q1) and 0.46(Q3). the element **x** of the **translational vector** has a median of -0.70, and 50 % of its values fall between -1.37(Q1) and -0.06(Q3). This indicates the **y** and **x** elements have a symmetrical distribution. However, the right whisker from the element **z** is larger than the left one, indicating a distribution skewed to the left. The whiskers' longitude and median value from all the pose elements indicate a low variability within its corresponding IQR.

6.2.1 Pose elements from FGSM attacks

6.2.2 $\mathcal{L}_1$ loss

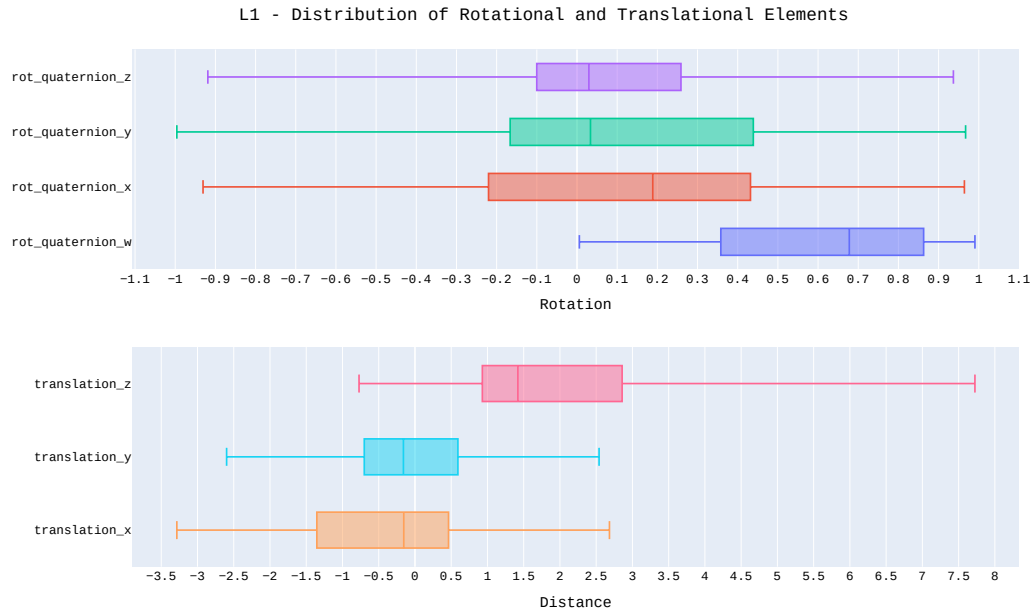


Figure 6.5: The top plot presents the values of the quaternion elements, and the bottom plot presents the value of the translational element of the camera pose computed by the ACE model when there is an "Attack" generated by the FGSM and produced with the $\mathcal{L}_1$ loss function.

Figure 6.5 represents that the element **z** of the **quaternion** has a median of 0.029, and 50 % of its values fall between -0.10(Q1) and 0.25(Q3). The element **y** of the **quaternion** has a median of 0.03, and 50 % of its values fall between -0.16(Q1) and 0.43(Q3). The element **x** of the **quaternion** has a median of 0.18, and 50 % of its values fall between -0.22(Q1) and 0.43(Q3). The element **w** of the **quaternion** has a median value of 0.67, and 50 % of its values fall between 0.35(Q1) and 0.86(Q3). The elements **z** and **y** have median values closer to the Q1 quantile, which indicates a skewed distribution to the left. The elements **x** and **w** have median values closer to the Q3 quantile, which indicates a skewed distribution to the right. **All elements** from the **quaternion** have long whiskers, indicating a high variability of values.

Figure 6.5 shows that the element **z** of the **translational vector** has a median of 1.41, and 50 % of its values fall between 0.92(Q1) and 2.85(Q3). The element **y** of the **translational vector** has a median of -0.15, and 50 % of its values fall between -0.69(Q1) and 0.59(Q3). the element **x** of the **translational vector** has a median of -0.15, and 50 % of its values fall between -1.35(Q1) and 0.46(Q3). Element **z** median value is closer to the Q1 quantile, which indicates a skewed distribution to the left. The element **y** median value is closer to the middle of the IQR box but towards the Q1 quantile, which indicates a slightly skewed distribution to the left. The element **x** median value is closer to the Q3 quantile, which indicates a slightly skewed distribution to the right. The whiskers from **all** the **translational elements** are long, indicating a high variability.

6.2.3 $\mathcal{L}_2$ loss

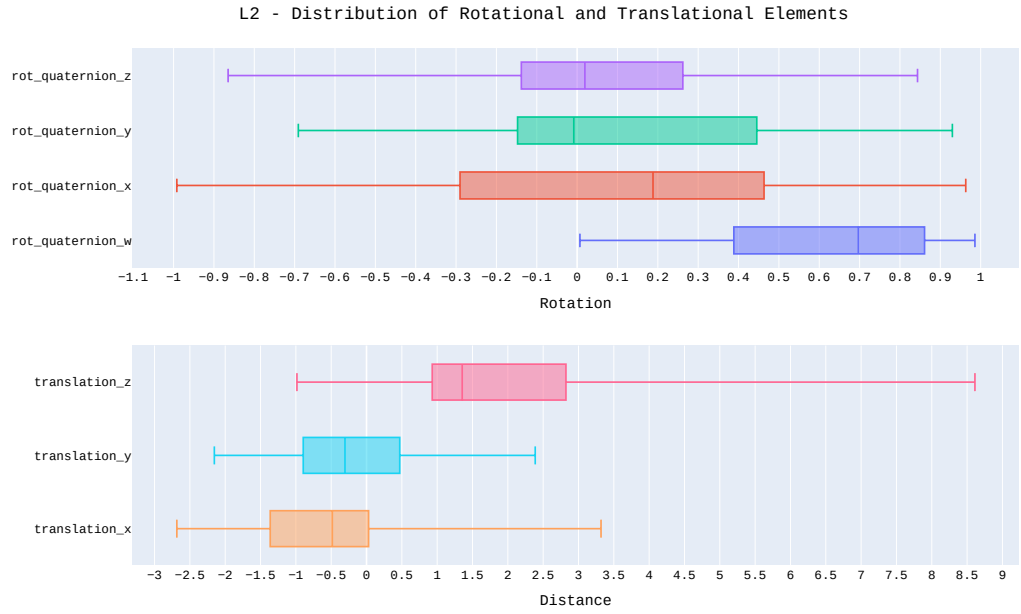


Figure 6.6: The top plot presents the values of the quaternion elements, and the bottom plot presents the value of the translational element of the camera pose computed by the ACE model when there is an "Attack" generated by the FGSM and produced with the $\mathcal{L}_2$ loss function.

Figure 6.6 represents that the element **z** of the **quaternion** has a median of 0.019, and 50 % of its values fall between -0.13(Q1) and 0.26(Q3). The element **y** of the **quaternion** has a median of -0.08, and 50 % of its values fall between -0.14(Q1) and 0.44(Q3). The element **x** of the **quaternion** has a median of 0.18, and 50 % of its values fall between -0.29(Q1) and 0.46(Q3). The element **w** of the **quaternion** has a median value of 0.69, and 50 % of its values fall between 0.38(Q1) and 0.86(Q3). The elements **z** and **y** have median values closer to the Q1 quantile, which indicates a skewed distribution to the left. The elements **x** and **w** have median values closer to the Q3 quantile, which indicates a skewed distribution to the right. **All elements** from the **quaternion** have long whiskers, indicating a high variability of values.

Figure 6.6 shows that the element **z** of the **translational vector** has a median of 1.35, and 50 % of its values fall between 0.92(Q1) and 2.82(Q3). The element **y** of the **translational vector** has a median of -0.30, and 50 % of its values fall between -0.89(Q1) and 0.47(Q3). the element **x** of the **translational vector** has a median of -0.48, and 50 % of its values fall between -1.36(Q1) and 0.029(Q3). Element **z** median value is closer to the Q1 quantile, which indicates a skewed distribution to the left. The element **y** median value is closer to the middle of the IQR box but towards the Q1 quantile, which indicates a slightly skewed distribution to the left. The element **x** median value is closer to the Q3 quantile, which indicates a slightly skewed distribution to the right. The whiskers from **all** the **translational elements** are long, indicating a high variability.

6.2.4 Pose elements from I-FGSM attacks

6.2.4.1 $\mathcal{L}_1$ loss

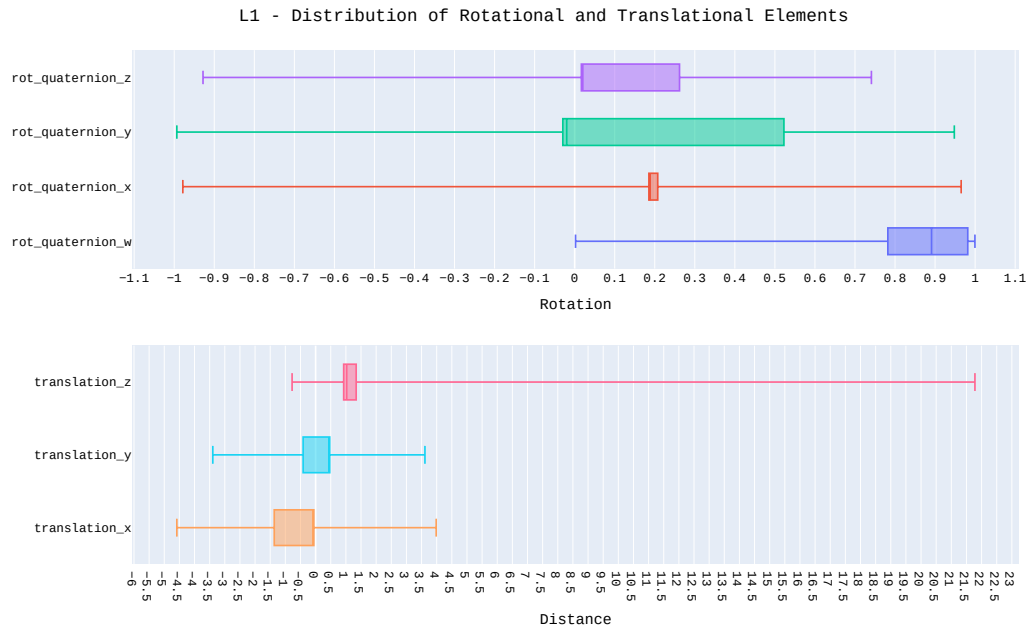


Figure 6.7: The top plot presents the values of the quaternion elements, and the bottom plot presents the value of the translational element of the camera pose computed by the ACE model when there is an "Attack" generated by the I-FGSM and produced with the $\mathcal{L}_1$ loss function.

Figure 6.7 represents that the element **z** of the **quaternion** has a median of 0.020, and 50 % of its values fall between -0.016(Q1) and 0.261(Q3). The element **y** of the **quaternion** has a median of -0.019, and 50 % of its values fall between -0.029(Q1) and 0.522(Q3). The element **x** of the **quaternion** has a median of 0.188, and 50 % of its values fall between 0.185(Q1) and 0.207(Q3). The element **w** of the **quaternion** has a median value of 0.891, and 50 % of its values fall between 0.782(Q1) and 0.981(Q3). The elements **z**, **y**, and **x** have median values too close to the Q1 quantile, which indicates a skewed distribution to the left. The element **w** has median values closer to the Q3 quantile, which indicates a skewed distribution to the right. **All elements** from the **quaternion** have long whiskers, indicating a high variability of values. Even when the right whisker of the element **w** is smaller, the left one indicates a high variability within values 0.022 and 0.782.

Figure 6.7 shows that the element **z** of the **translational vector** has a median of 1.026, and 50 % of its values fall between 0.926(Q1) and 1.34(Q3). The element **y** of the **translational vector** has a median of 0.447, and 50 % of its values fall between -0.412(Q1) and 0.466(Q3). The element **x** of the **translational vector** has a median of -0.076, and 50 % of its values fall between -1.366(Q1) and -0.059(Q3). Element **z** median value is closer to the Q1 quantile, which indicates a skewed distribution to the left. The elements **y** and **x** median values are too close to the Q3 quantile, which indicates a skewed distribution to the right. The whiskers from **all** the **translational elements** are long, indicating a high variability. Even when the left whisker of the element **z** is smaller, the right one indicates a high variability within values 1.34 and 21.780.

6.2.4.2 $\mathcal{L}_2$ loss

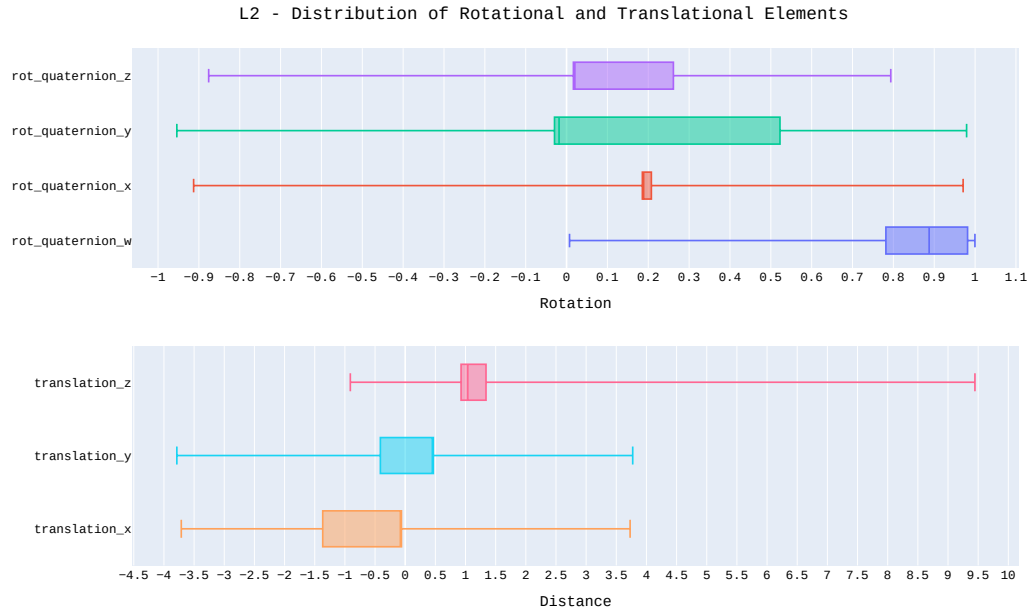


Figure 6.8: The top plot presents the values of the quaternion elements, and the bottom plot presents the value of the translational element of the camera pose computed by the ACE model when there is an "Attack" generated by the I-FGSM and produced with the $\mathcal{L}_2$ loss function.

Figure 6.8represents that the element **z** of the **quaternion** has a median of 0.020, and 50 % of its values fall between -0.016(Q1) and 0.261(Q3). The element **y** of the **quaternion** has a median of -0.018, and 50 % of its values fall between -0.029(Q1) and 0.522(Q3). The element **x** of the **quaternion** has a median of 0.188, and 50 % of its values fall between 0.185(Q1) and 0.207(Q3). The element **w** of the **quaternion** has a median value of 0.887, and 50 % of its values fall between 0.781(Q1) and 0.981(Q3). The elements **z**, **y**, and **x** have median values too close to the Q1 quantile, which indicates a skewed distribution to the left. The element **w** has median values closer to the Q3 quantile, which indicates a skewed distribution to the right. **All elements** from the **quaternion** have long whiskers, indicating a high variability of values. Even when the right whisker of the element **w** is smaller, the left one indicates a high variability within values 0.007 and 0.781.

Figure 6.8 shows that the element **z** of the **translational vector** has a median of 1.037, and 50 % of its values fall between 0.926(Q1) and 1.339(Q3). The element **y** of the **translational vector** has a median of 0.450, and 50 % of its values fall between -0.411(Q1) and 0.466(Q3). The element **x** of the **translational vector** has a median of -0.079, and 50 % of its values fall between -1.367(Q1) and -0.062(Q3). Element **z** median value is closer to the Q1 quantile, which indicates a skewed distribution to the left. The elements **y** and **x** median values are too close to the Q3 quantile, which indicates a skewed distribution to the right. The whiskers from **all** the **translational elements** are long, indicating a high variability. Even when the left whisker of the element **z** is smaller, the right one indicates a high variability within values 1.339 and 9.446.

6.3 Loss and Average error values

6.3.1 FGSM attacks

The plots in Figures 6.9 and 6.10 present the relation in the percentage of the Loss and Average error values and the epsilon values. The median of the loss and average error were taken and then applied to Equations6.1 and 6.2 to determine the change relative to the baseline "No Attack" scenario.

$$\text{Loss percentage change} = \frac{loss_{\epsilon=0,..,1} - loss_{\epsilon=0}}{loss_{\epsilon=0}} * 100, \tag{6.1}$$

where $loss_{\epsilon=0,..,1}$ is the loss value of the adversarial attacks with different ϵ values, and $loss_{\epsilon=0}$ is the loss value when there is "No Attack."

$$\text{Average error percentage change} = \frac{average\ \ error_{\epsilon=0,..,1} - average\ \ error_{\epsilon=0}}{average\ \ error_{\epsilon=0}} * 100 \tag{6.2}$$

where $average\ \ error_{\epsilon=0,..,1}$ is the average error produced by the adversarial attacks with different ϵ values, and $average\ \ error_{\epsilon=0}$ is the average error when there is "No Attack."

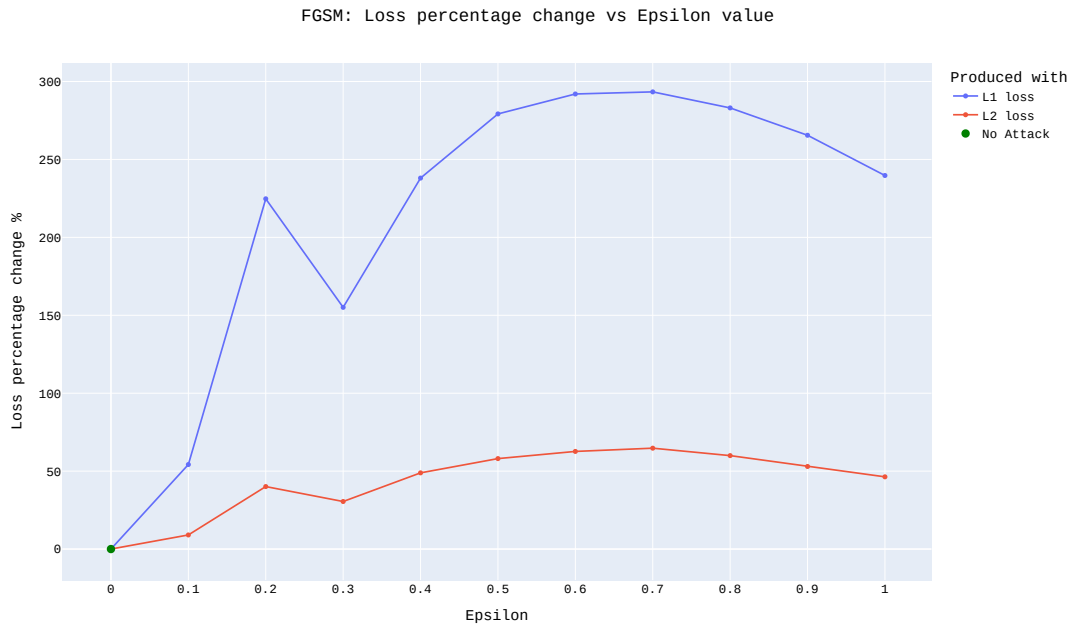


Figure 6.9: Change of the Loss value in percentage when the epsilon value increases from 0 to 1. When the adversarial attack is generated with the FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss.

Figure 6.9 presents the "No Attack" baseline scenario with a green point on epsilon equal to zero. The adversarial attacks produced with the $\mathcal{L}_1$ loss function have a Loss value increase as the epsilon value increases. However, after reaching the maximum Loss percentage change of 293% when the epsilon is equal to 0.7, the Loss decreases to 240% when the epsilon is 1. The Loss percentage change line follows a trajectory similar to an inverse parabolic curve, except when epsilon is 0.2, the Loss peaks 240%. The adversarial attacks produced with the $\mathcal{L}_2$ function have a Loss value increase as the epsilon value increases; the line has a trend and form similar to the $\mathcal{L}_1$ loss function line but with smaller magnitude. Nevertheless, like with adversarial attacks produced with $\mathcal{L}_1$, the $\mathcal{L}_2$ peaks when epsilon is 0.2 with a value of 40%.

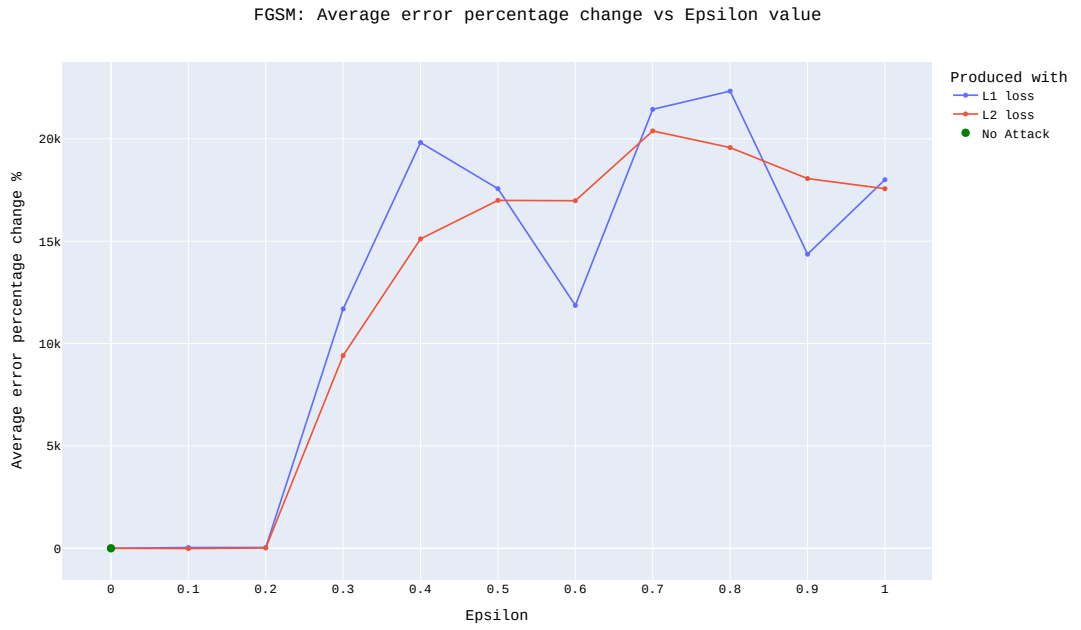


Figure 6.10: Change of the Average error in percentage when the epsilon value increases from 0 to 1. When the adversarial attack is generated with the FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss.

Figure 6.10 presents the "No Attack" baseline scenario with a green point on epsilon equal to zero. Attacks produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions when epsilon is 0.1 and 0.2 have an average error percentage around 30%, which seem like 0 when these are compared with the rest values of the plot. Both attacks plunge when epsilon is 0.6; $\mathcal{L}_1$ attack has a value of 11.82k % and $\mathcal{L}_2$ a value of 16.97k %. $\mathcal{L}_1$ attacks after the plunge has a maximum of 22.32K % when epsilon is 0.8 and then goes down to 14.36 % when epsilon is 0.9. $\mathcal{L}_2$ maximum of 20.38K % is when epsilon is 0.7.

6.3.2 I-FGSM attacks

The plots in Figures 6.11, 6.12, 6.13, and 6.15 present the relationship between the percentage changes in the Loss and Average error values with the epsilon values and the iteration step number. The loss and average error median were computed when the dataset was grouped by 'epsilon' and 'iteration step'; then applied to Equations 6.3 and 6.4 to determine the change relative to the baseline "No Attack" scenario, which is represented in the formulas when $\epsilon^I = 0$ and $N = 0$, where N is the iteration step number.

$$\text{Loss percentage change} = \frac{loss_{\epsilon^I=\{0,..,1\},N=\{0,...,20\}} - loss_{\epsilon^I=0,N=0}}{loss_{\epsilon^I=0,N=0}} * 100, \tag{6.3}$$

where $loss_{\epsilon^I=\{0,..,1\},N=\{0,...,20\}}$ is the loss value of the adversarial attacks with different ϵ^I and N values, and $loss_{\epsilon^I=0,N=0}$ is the loss value when there is "No Attack."

$$\begin{matrix}\text{Average error} \\ \text{percentage change}\end{matrix} = \frac{average\ \ error_{\epsilon^I=\{0,..,1\},N=\{0,...,20\}} - average\ \ error_{\epsilon^I=0,N=0}}{average\ \ error_{\epsilon^I=0,N=0}} * 100, \tag{6.4}$$

where $average\ \ error_{\epsilon^I=\{0,..,1\},N=\{0,...,20\}}$ is the average error produced by the adversarial attacks with different ϵ^I and N values, and $average\ \ error_{\epsilon^I=0,N=0}$ is the average error when there is "No Attack."

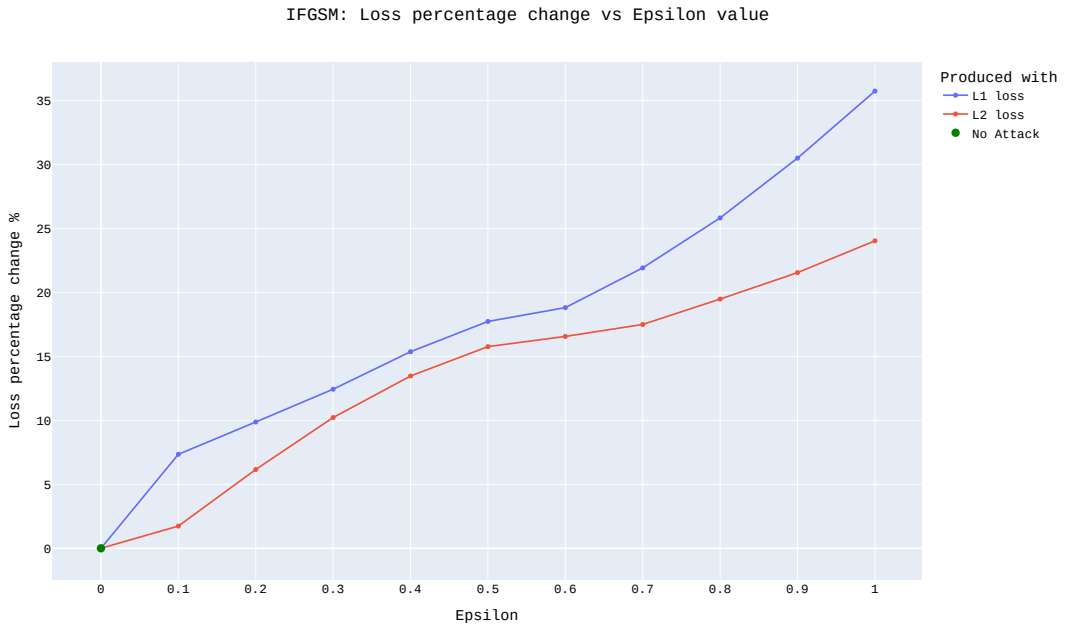


Figure 6.11: Change of the Loss value in percentage when the epsilon value increases from 0 to 1, from adversarial attacks generated with the I-FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions.

Figure 6.11 presents the "No attack" scenario with a green point at $\epsilon^I = 0$. Adversarial attacks produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ have a linear increasing change in Loss with a maximum of 35 % 24% both at $\epsilon^I = 1$ correspondingly. When $\epsilon^I = 0.1$, $\mathcal{L}_1$ produced attacks have an increment in slope compared with most epsilon values, while attacks produced with $\mathcal{L}_2$ have a decrease in slope. In contrast, when $\epsilon^I = 0.7$, both loss functions have a decrease in slope. Adversarial Attacks produced with $\mathcal{L}_2$ have a smaller Loss change compared with the $\mathcal{L}_1$. $\mathcal{L}_2$ produced attacks have a maximum of 24% when $\epsilon^I = 1$.

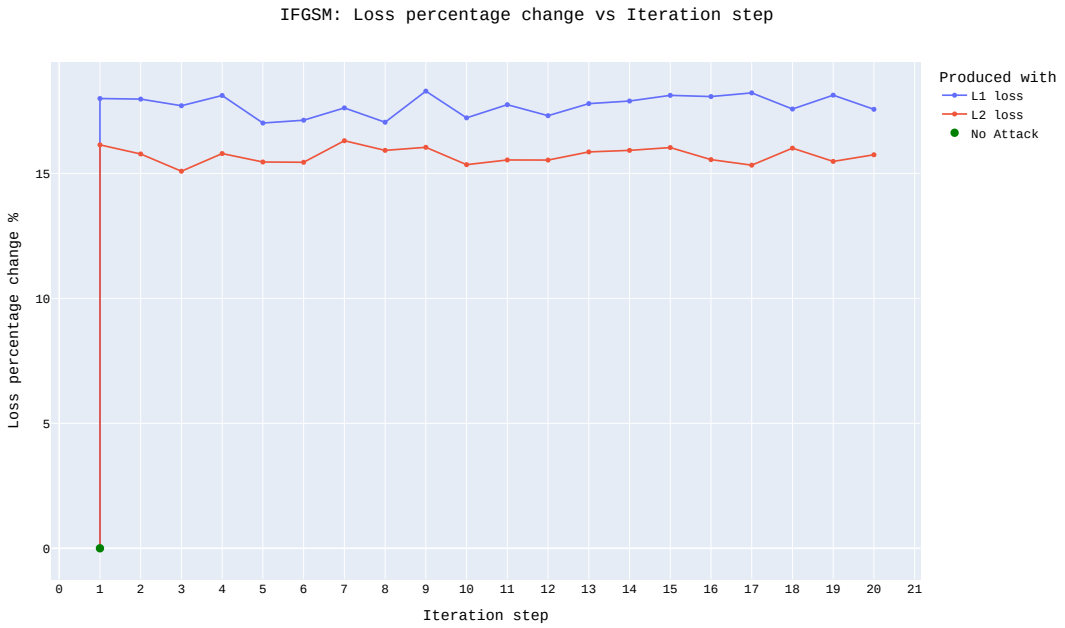


Figure 6.12: Change of the Loss value in percentage when the iteration step number increases from 1 to 19, from adversarial attacks generated with the I-FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions.

Figure 6.12 presents the "No attack" scenario with a green point. There are two values at $N = 1$; the first one represents the "No Attack" scenario when $N = 1$ and $\epsilon^I = 0$, and the second one represents the Attack scenario when $\epsilon^I > 0$. Adversarial attacks generated with I-FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ do not present an increase in the Loss percentage change when the iteration step increases from 1 to 19. Attacks produced with $\mathcal{L}_1$ loss have a greater magnitude in Loss percentage change than those produced with $\mathcal{L}_2$ loss. For both losses, the change in Loss seems to be constant over the iteration steps.

Figure 6.13: Change of the Average error in percentage when the epsilon value increases from 0 to 1, from adversarial attacks generated with the I-FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions.

Figure 6.13 presents the "No attack" scenario with a green point at $\epsilon^I = 0$. Adversarial attacks generated with the I-FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions have an average error percentage change near 0 from $\epsilon^I = 0$ to $\epsilon^I = 0.4$. However, taking a closer look at Figure 6.14, it can be seen that the average error for both attacks decreases approximately to 0.6 for $\mathcal{L}_1$ attacks and 0.1 for $\mathcal{L}_2$ attacks when $\epsilon^I = 0.3$.

From 6.13, $\mathcal{L}_1$ attacks average error increase from $\epsilon^I = 0.4$ to $\epsilon^I = 0.7$, which curve has a convex form. In contrast, from the same interval of epsilon values, $\mathcal{L}_2$ attacks average error also increases but has a smaller magnitude than $\mathcal{L}_1$ attacks, reflected in the concave curve. In the interval $\epsilon^I = 0.7$ to $\epsilon^I = 0.9$ both $\mathcal{L}_1$ and $\mathcal{L}_2$ attacks increases at an smaller change than before; and $\mathcal{L}_2$ have a greater average error magnitude than $\mathcal{L}_1$ attacks.

Figure 6.14: Close up to epsilon values 0.0 to 0.4. Change of the Average error in percentage when the epsilon value increases from 0 to 1, from adversarial attacks generated with the I-FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions.

Figure 6.15: Change of the Average error in percentage when the iteration step number increases from 1 to 19, from adversarial attacks generated with the I-FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions.

Figure 6.15 presents the "No attack" scenario with a green point. There are two values at $N = 1$; the first one represents the "No Attack" scenario when $N = 1$ and $\epsilon^I = 0$, and the second one represents the Attack scenario when $\epsilon^I > 0$. Similar to the change in the Loss

plot from Figure 6.12, it seems that the iteration step does not have an effect of increasing the average error for both adversarial attacks generated with I-FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$. Nonetheless, $\mathcal{L}_1$ attacks have greater average error than $\mathcal{L}_2$ attacks.

6.4 Analysis

6.4.1 Analysis of Adversarial Attacks generated by FGSM

The polynomial degree with the lowest MSE was 7 for attacks produced with $\mathcal{L}_1$ loss function and 3 for attacks produced with $\mathcal{L}_2$ loss function.

6.4.1.1 Attacks produced with $\mathcal{L}_1$ loss function

	Intercept	**epsilon** ϵ
Coefficient	8.13	70.52
P>|t|	0.006	< 2e-16
R-squared		0.478
Adj. R-squared		0.476

Table 6.1: Value of R^2, adjusted R^2, coefficients and p-values from the intercept and the predictor variable ϵ used to fit the simpler linear regression, where ϵ is the hyperparameter used to create the adversarial attacks generated by FGSM and produced with $\mathcal{L}_1$ loss function.

From Table 6.1, the p-value of ϵ is smaller than 0.05, indicating a statistically significant variable for the model to predict the average error. Its coefficient value reflects a positive increment of the average error of 70.52 times when it increases by 1 unit. The adjusted R^2 is closer to 0.5, suggesting that the model explains less than half of the variance in the average error.

	Intercept	**epsilon** ϵ	ϵ^2	ϵ^3	ϵ^4	ϵ^5	ϵ^6	ϵ^7
Coefficient	43.398	330.796	-160.877	-16.680	75.405	-77.927	32.262	131.998
P>|t|	< 2e-16	< 2e-16	< 2e-16	0.333	1.83e-05	9.80e-06	0.062	5.97e-13
R-squared								0.725
Adj. R-squared								0.716

Table 6.2: Value of R^2, adjusted R^2, coefficients and p-values from the intercept and the predictor variables $X = \{\epsilon, \epsilon^2, ..., \epsilon^7\}$ used to fit the polynomial regression of degree 7, where ϵ is the hyperparameter used to create the adversarial attacks generated by FGSM and produced with $\mathcal{L}_1$ loss function.

From Table 6.2, the predictor variables $X = \{\epsilon, \epsilon^2, \epsilon^4, \epsilon^5, \epsilon^7\}$ have a p-value smaller than 0.05, indicating that these variables are statistically significant to explain the behavior of the target variable, the average error. ϵ has the biggest positive coefficient value, which reflects a positive increment of the average error of 330.7 times when it increases by one unit. The adjusted R^2 of 0.71 is closer to 1, suggesting that the model explains a substantial portion of the variance in the average error.

Both models have an adjusted R^2 slightly smaller than the R^2, indicating that some predictor variables may not explain the target variable and add complexity to that model.

6.4.1.2 Attacks produced with $\mathcal{L}_2$ loss function

	Intercept	**epsilon** ϵ
Coefficient	7.464	71.72
P>|t|	0.0070	< 2e-16
R-squared		0.523
Adj. R-squared		0.520

Table 6.3: Value of R^2, adjusted R^2, coefficients and p-values from the intercept and the predictor variable ϵ used to fit the simpler linear regression, where ϵ is the hyperparameter used to create the adversarial attacks generated by FGSM and produced with $\mathcal{L}_2$ loss function.

From Table 6.3, the p-value of ϵ is smaller than 0.05, which indicates it is variable statistically significant for the model to predict the average error. Its coefficient value reflects a positive increment of the average error of 71.72 times when it increases by 1 unit. The adjusted R^2 of 0.52 suggests that the model explains slightly more than half of the variance in the average error.

	Intercept	**epsilon** ϵ	ϵ^2	ϵ^3
Coefficient	43.32	336.39	-172.11	-50.06
P>|t|	< 2e-16	< 2e-16	< 2e-16	0.006
R-squared				0.671
Adj. R-squared				0.667

Table 6.4: Value of R^2, adjusted R^2, coefficients and p-values from the intercept and the predictor variables $X = \{\epsilon, \epsilon^2, \epsilon^3\}$ used to fit the polynomial regression of degree 3, where ϵ is the hyperparameter used to create the adversarial attacks generated by FGSM and produced with $\mathcal{L}_2$ loss function.

From Table 6.4, the predictor variables $X = \{\epsilon, \epsilon^2, \epsilon^3\}$ have a p-value smaller than 0.05, indicating that these variables are statistically significant to explain the behavior of the target variable, the average error. ϵ is the only predictor variable with a positive coefficient bigger than the rest. It reflects an increase of 336.39 times the average error when it increases by 1 unit. The adjusted R^2 of 0.66 suggests that the model explains slightly above half of the variance in the average error.

Both models have an adjusted R^2 slightly smaller than the R^2, indicating that some predictor variables may not explain the target variable and add complexity to that model.

6.4.2 Analysis of Adversarial attacks generated by I-FGSM

The polynomial degree combination with the lowest MSE was 7 for the predictor variable ϵ^I and 2 for N, for both attacks produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions.

6.4.2.1 Attacks produced with $\mathcal{L}_1$ loss function

	Intercept	**epsilon** ϵ^I	**iteration** N
Coefficient	14.04	27.73	-0.05
P>|t|	< 2e-16	< 2e-16	0.327
R-squared			0.151
Adj. R-squared			0.150

Table 6.5: Value of R^2, adjusted R^2, coefficients and p-values from the intercept, the predictor variables ϵ^I and the iteration step number N of the multiple linear regression fitted using the adversarial attacks generated by I-FGSM and produced with $\mathcal{L}_1$.

From Table 6.5, the p-value of ϵ^I is smaller than 0.05, which indicates it is variable statistically significant for the model to predict the average error. As opposed to ϵ^I, the p-value of 0.327 from N indicates it is a variable that does not contribute to explaining the target variable. The ϵ^I coefficient is positive, reflecting an increase in the average error of 28.07 times when ϵ^I increases by one unit. Meanwhile, the N coefficient of -0.05 is negative and small, which indicates that the average error may decrease slightly. The adjusted R^2 is closer to 0.1, suggesting that the model does not explain the majority of the variance in the average error.

	Intercept	**epsilon** ϵ^I	ϵ^{I^2}	ϵ^{I^3}	ϵ^{I^4}	ϵ^{I^5}	ϵ^{I^6}	ϵ^{I^7}	**iteration** N	N^2
Coefficient	27.3540	581.820	246.709	-19.054	-38.763	37.811	50.742	14.815	-20.421	-9.366
P>|t|	<2e-16	<2e-16	<2e-16	0.3524	0.0585	0.0650	0.0133	0.4696	0.3189	0.6475
R-squared										0.180
Adj. R-squared										0.178

Table 6.6: Value of R^2, adjusted R^2, coefficients and p-values from the intercept and the predictor variables $X = \{\epsilon, \epsilon^{I^2}, \epsilon^{I^3}, \epsilon^{I^4}, \epsilon^{I^5}, \epsilon^{I^6}, \epsilon^{I^7}, N, N^2\}$ used to fit the polynomial regression of degree 7, where ϵ^I and N are the hyperparameters used to create the adversarial attacks generated by I-FGSM and produced with $\mathcal{L}_1$ loss function.

From Table 6.6, the predictor variables $X = \{\epsilon^I, \epsilon^{I^2}, \epsilon^{I^6}\}$ have a p-value smaller than 0.05, indicating that these variables are statistically significant to explain the behavior of the target variable, the average error. ϵ^I has a positive coefficient, which reflects a positive increment of the average error of 581.82 times when it increases by one unit. The adjusted R^2 of 0.17 is closer to 0, suggesting that the model does not explain the majority of the variance in the average error.

6.4.2.2 Attacks produced with $\mathcal{L}_2$ loss function

	Intercept	**epsilon** ϵ^I	**iteration** N
Coefficient	13.23	28.07	0.006
P>|t|	< 2e-16	< 2e-16	0.899
R-squared			0.154
Adj. R-squared			0.153

Table 6.7: Value of R^2, adjusted R^2, coefficients and p-values from the intercept, the predictor variables ϵ^I, iteration step number N used to fit the multiple linear regression, where ϵ^I and N are the hyperparameters used to create the adversarial attacks generated by I-FGSM and produced with the $\mathcal{L}_2$.

From Table **??**, the p-value of ϵ^I is smaller than 0.05, which indicates it is variable statistically significant for the model to predict the average error. Contrary to ϵ^I, the p-value of 0.89 from N indicates it is a variable that does not contribute to explaining the target variable. The ϵ^I coefficient is positive, reflecting an increase in the average error of 28.07 times when ϵ^I increases by one unit. The adjusted R^2 is closer to 0.1, suggesting that the model does not explain the majority of the variance in the average error.

	Intercept	**epsilon** ϵ^I	ϵ^{I^2}	ϵ^{I^3}	ϵ^{I^4}	ϵ^{I^5}	ϵ^{I^6}	ϵ^{I^7}	**iteration** N	N^2
Coefficient	27.346	588.956	259.509	-15.716	-49.072	25.020	41.183	4.876	2.644	16.333
P>|t|	<2e-16	<2e-16	<2e-16	0.441	0.016	0.220	0.043	0.811	0.896	0.423
R-squared										0.186
Adj. R-squared										0.184

Table 6.8: Value of R^2, adjusted R^2, coefficients and p-values from the intercept and the predictor variables $X = \{\epsilon, \epsilon^{I^2}, \epsilon^{I^3}, \epsilon^{I^4}, \epsilon^{I^5}, \epsilon^{I^6}, \epsilon^{I^7}, N, N^2\}$ used to fit the polynomial regression of degree 7, where ϵ^I and N are the hyperparameters used to create the adversarial attacks generated by I-FGSM and produced with $\mathcal{L}_2$ loss function.

From Table 6.8, the predictor variables $X = \{\epsilon^I, \epsilon^{I^2}, \epsilon^{I^4}, \epsilon^{I^6}\}$ have a p-value smaller than 0.05, indicating that these variables are statistically significant to explain the behavior of the target variable, the average error. ϵ^I has a positive coefficient, which reflects a positive increment of the average error of 588.956 times when it increases by one unit. The adjusted R^2 of 0.18 is closer to 0, suggesting that the model does not explain the majority of the variance in the average error.

7 Discussion

Visually, there is no substantial difference between adversarial example images produced with $\mathcal{L}_1$ and $\mathcal{L}_2$, whether these were generated by FGSM or I-FGSM. Adversarial example images generated by FGSM are less similar to the original images and have more color changes than those generated by I-FGSM when the corresponding hyperparameter, ϵ **or** ϵ^I, increases.

By analyzing the boxplots in Section 6.2, the symmetrical distribution with short whiskers that are too close to the Q1 and Q3 quantiles suggests a low variability of values. At the same time, the pose elements from all the "Attack" scenarios do not have median values in the center of the IQR box, in addition to larger whiskers that are far from the Q1 and Q3 quantiles. This suggests that adversarial attacks generated by FGSM and I-FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions, when applied to the ACE model, provoke a higher variability of the pose elements computed by the model.

The attacks generated by the FGSM compared with the I-FGSM have pose elements with larger IQR boxes, indicating a higher variability of values within the quantiles Q1 and Q3. The whisker's longitude between these methods is almost alike, which indicates similar variability. Within each method, the attacks produced with $\mathcal{L}_1$ loss function compared with the $\mathcal{L}_2$ loss function present larger IQR boxes. This indicates that $\mathcal{L}_1$ loss function produced attacks causing a higher variability of the pose elements values than those produced with $\mathcal{L}_2$ loss function. Nonetheless, the attacks produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ have relatively similar median values conserving the same skewness.

The adversarial attacks generated by FGSM produce the same change of loss measured in percentage regardless of the loss function used to produce the attacks. Nevertheless, the attacks produced with $\mathcal{L}_1$ loss function produce a higher Loss change than attacks produced with $\mathcal{L}_2$ loss function. But, when comparing the change in percentage of the average error, the difference in magnitude decreases, but $\mathcal{L}_1$ loss attacks still produce a higher change in percentage of the average error.

The adversarial attacks generated with the I-FGSM and produced with the $\mathcal{L}_1$ loss function increase the average error and loss when the hyperparameter ϵ^I increases. Generally, $\mathcal{L}_1$

attacks produced a bigger average error and loss increase than $\mathcal{L}_2$ attacks. The iteration step number N does not present an apparent effect over the average error and loss for attacks of both loss functions. A reason why adversarial attacks produced with $\mathcal{L}_2$, regardless of being generated by FGSM or I-FGSM, produced the ACE model to compute poses with the highest average error for most of the range of ϵ and ϵ^I values could be that $\mathcal{L}_2$ loss function penalize larger values, which results in smaller loss values. This is due to the nature of its formula, which takes the square root of the sum of squares.

Taking a closer look at the average error change in percentage vs the corresponding epsilon value, ϵ and ϵ^I, compared with the "No attack" scenario, the adversarial attacks generated by FGSM and produced with $\mathcal{L}_1$ loss function, and the attacks generated by I-FGSM with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions reduced the average error when the corresponding epsilon is 0.1. Furthermore, the attacks generated with I-FGSM keep reducing the change in the average error when ϵ^I is 0.2, 0.3, and 0.4.

The decrease in the average error could suggest that the ACE model has been trained with images including some noise. The ACE model used in this thesis was trained with the frames of the chess sequences from the Microsoft 7scenes dataset. These images were obtained through a Kinect system; it is known that depth images captured with it introduced some type of noises, such as temporal and spatial noise[99]. Moreover, the frames were taken from the sequence recordings, adding some motion to the images.

The change in the loss value vs. the ϵ^I in Figure 6.11 from Section 6.3 is the only relationship that suggests being linear; the rest of the Figures from this section have polynomial relationships. In particular, plots in Figures 6.10 and 6.13 from Section 6.3, which includes the established target variable "average error," show this behaves polynomially when the epsilon value increases. This was the motivation to include the polynomial regression to analyze the average error resulting from the different values from the hyperparameters ϵ, ϵ^I and N, in addition to the linear regression.

Results in Section 6.4.1 suggest that the adversarial attacks generated by FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ do not follow a linear relationship between the average error and the epsilon value because the linear regressions have an adjusted R^2 that roughly explain half of the variance of the target variable. However, both linear regressions reflect a positive increment of the average error when ϵ increases. Results from polynomial regressions of $\mathcal{L}_1$ and $\mathcal{L}_2$ attacks have adjusted R^2's, explaining around 70% of the average error variance. In addition to the positive coefficient of ϵ, the results show that the adversarial attacks generated by FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions increase the average error of the pose computed by the ACE model in a polynomial trend.

Results in Section 6.4.2 suggest that the adversarial attacks generated by I-FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}_2$ do not follow a linear relationship or a polynomial relationship because the two linear regressions and two polynomial regression have a close to 0.1 adjusted R^2. Nonetheless, polynomial adjusted R^2's are slightly higher. The coefficient value of ϵ^I in all regression shows an increase of the average error when it increases. The $\mathcal{L}_1$ produced attacks appear to be of higher magnitude when compared to $\mathcal{L}_2$ attacks, which is in line with a higher value in the ϵ^I coefficients from the results of the regressions. The predictor variables with small negative coefficients compared to the ϵ^I, for $X = \{\epsilon^{I^3}, \epsilon^{I^4}\}$ $\mathcal{L}_1$ produced attacks and $X = \{\epsilon^{I^3}, \epsilon^{I^4}, N, N^2\}$ for $\mathcal{L}_2$ are not statically significant, however could explain the small decrease in the average error pointed out in Figure 6.14 from Section 6.3.

One possible reason the polynomial regression does not sufficiently describe the I-FGSM-

generated attacks compared to generated FGSM attacks is that it uses a global approach, where the model explains the target variable over the whole range of X. However, after the obtained results, it has to be considered that the target variable y_{ae} behaves differently over different intervals of $X = \{\epsilon^I, N\}$, making a piecewise polynomial a more suitable model for understanding the nature of the adversarial attacks applied to the ACE model.

An important insight is that the iteration step number N does not appear to affect the increase in the average error value. This is confirmed by the imperceptible changes between images of different iteration numbers, the plots in Section 6.3 showing what could be described as arbitrary values regardless of the N number, and the larger p-values in all regression results.

The plots in Section 6.3 show average error curves that follow a polynomial trend, which aligns with the higher adjusted R^2 values on the polynomial regressions. Attacks generated with FGSM produced a higher increment of the average error than attacks produced with IFGSM, regardless of the loss function applied. This could be because each attack's hyperparameter epsilon serves a different purpose. In FGSM Equation 3.7, ϵ is used as a scale constant that amplifies or reduces the adversarial perturbation. While in the I-FGSM Equation 3.8, the scale constant is α, with a fixed value of one for all the adversarial examples produced. Furthermore, clip Equation 3.9 applied the ϵ^I to constrain the value of the adversarial example by subtracting or adding the ϵ^I value to the value of channel z from the last image at coordinates (x,y).

The results obtained in this thesis should be repeatable if the values of the hyperparameters are the same for the adversarial examples and the same target variable and predictors are used for the regression models. Nonetheless, it is known that further research on this topic could find that there is another metric that could replace the "average error" presented here. This metric could better represent the nature of the pose computed error, perhaps changing how much relevance the rotational and translational components have.

7.1 Limitations and further work

The implementations consider only a learning-based method with a Neural Network architecture and two adversarial attacks that blend white-box and black-box types. It represents a small fraction of the entire ecosystem of visual localization methods and adversarial attacks. Additionally, the data utilized is only from indoor images. To know more about the robustness of visual localization methods, outdoor data needs to be explored, which will represent a more demanding scenario for the model because it will carry more abrupt changes in the environmental conditions such as light, season of the year when the images were taken, weather and more.

Furthermore, a wider range of hyperparameter values could be explored to review if the adversarial attacks generated with FGSM and produced with $\mathcal{L}_1$ and $\mathcal{L}2$ loss functions keep the trend of reducing the average error once the maximum is reached. It could be the case that a new optimization method is found if the average error keeps decreasing or that, perhaps, the average error increases in a polynomial and cyclic trend, where in some values of ϵ, it reduces. Still, it increases after some increment of ϵ. And review if the attacks generated by I-FGSM keep increasing in a polynomial trend as the ϵ^I increments.

7.2 Ethical Considerations

Visual localization methods are used in real life to estimate the positions of robotics and autonomous vehicles in different environments. The addition of adversarial attacks, whether in image classification, object detection, or visual localization methods, among other computer vision systems, raises ethical considerations about what kind of disruption these attacks could create if they were targeted for malicious use outside academic research, i.e., application in modern warfare. This concern alone has brought big technological corporations such as "Lucid AI" and "Google" to the conversation in addition to 30 countries, through the Lethal Autonomous Weapons Pledge[1]. Simultaneously, research[100] made by RAND, a nonprofit organization, reveals that the speed and scale of operation from autonomous weapons systems instigate conflict escalation. This thesis focuses on the robustness of one visual localization method model, but it is important to be aware of possible consequences around the topic.

7.3 Use of AI tools

The AI tools used only as a support element in this thesis were ChatGPT and Grammarly. ChatGPT was used strictly to debug some errors in the adversarial attack code, and Grammarly was used to review the spelling and grammar.

[1]https://futureoflife.org/open-letter/lethal-autonomous-weapons-pledge/

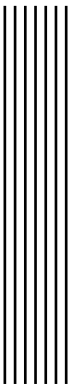
8 Conclusion

The FGSM and I-FGSM applied to the ACE model were a good first approach. The characteristics considered were that the VL method needed a sequential workflow for a more straightforward computation of the attacks and access to all the model information, such as the gradient, architecture, and training dataset.

Access to the loss function and the gradient was crucial to adapt the adversarial attacks to the ACE model. To overcome backpropagating the projection error through the differentiable RANSAC, $\mathcal{L}_1$, and $\mathcal{L}_2$ losses were chosen to produce the attacks. The 3D scene-predicted coordinates of the MLP were used to compute the losses and backpropagate through the MLP network.

The FGSM-generated attacks produced by $\mathcal{L}_1$, and $\mathcal{L}_2$ losses incremented the loss value, which increased with the ϵ value, resulting in a higher average pose error and a decline in the performance of the ACE model. The I-FGSM-generated attacks produced by $\mathcal{L}_1$, and $\mathcal{L}_2$ losses also incremented the loss value when ϵ^I increased, but the additional hyperparameter the iteration step number N do not have an evident effect on the average error. However, there is an exception when $\epsilon = 0.1$ in the FGSM $\mathcal{L}_2$ attacks and when $\epsilon^I = 0.1, .., 0.4$ in the I-FGSM $\mathcal{L}_1$ and $\mathcal{L}_2$ attacks reduce the average error.

The ϵ and ϵ^I values control the magnitude of the attacks, and the coefficients of the different regression analyses show it creates a positive or negative increase in the average error. The first hypothesis was to see a linear relationship between the average error and the epsilon values. However, the results confirmed that the increment is polynomial for FGSM-generated and I-FGSM attacks. Nevertheless, the results indicate that part of the relation could not be explained by only adding the epsilon values to the model due to the small R^2 and p-values.



Bibliography

[1] Junyi Chai, Hao Zeng, Anming Li, and Eric W.T. Ngai. "Deep learning in computer vision: A critical review of emerging techniques and application scenarios". In: *Machine Learning with Applications* (2021).

[2] Samuel Henrique Silva and Peyman Najafirad. *Opportunities and Challenges in Deep Learning Adversarial Robustness: A Survey*. 2020. arXiv: 2007.00753.

[3] Ahmed Aldahdooh, Wassim Hamidouche, and Olivier Deforges. *Reveal of Vision Transformers Robustness against Adversarial Attacks*. 2021. arXiv: 2106.03734 [cs.CV].

[4] Aniruddha Saha, Akshayvarun Subramanya, Koninika Patil, and Hamed Pirsiavash. "Role of Spatial Context in Adversarial Robustness for Object Detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. eprint: 1910.00068.

[5] Lintong Zhang, Yifu Tao, Jiarong Lin, Fu Zhang, and Maurice Fallon. *Visual Localization in 3D Maps: Comparing Point Cloud, Mesh, and NeRF Representations*. 2024. arXiv: 2408.11966.

[6] Lucas Brytne. "Learning and Optimizing Camera Pose". PhD thesis. Chalmers University of Technology, 2024.

[7] Computer Vision Machine Learning Team. *An On-device Deep Neural Network for Face Detection*. https://machinelearning.apple.com/research/face-detection. [Accessed 11-09-2024]. 2017.

[8] Niantic. *Visual Positioning System (VPS) | Niantic Lightship — lightship.dev*. https://lightship.dev/docs/ardk/features/lightship_vps/. [Accessed 11-09-2024].

[9] Google. *Google Maps help*. Available at: https://support.google.com/maps. [Accessed 04-01-2025]. 2025.

[10] Amazon Staff. *10 years of Amazon robotics: how robots help sort packages, move product, and improve safety*. Available at: https://www.aboutamazon.com/news/operations/10-years-of-amazon-robotics-how-robots-help-sort-packages-move-product-and-improve-safety. [Accessed 04-01-2025]. 2022.

[11] The Waymo Team. *Utilizing key point and pose estimation for the task of autonomous driving*. Available at: https://waymo.com/blog/2022/02/utilizing-key-point-and-pose-estimation. [Accessed 04-01-2025]. 2022.

[12] UAV Navigation. *Visual Navigation System*. Available at: https://www.uavnavigation.com/company/blog/visual-navigation-system. [Accessed 04-01-2025]. 2019.

[13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. *Intriguing properties of neural networks*. 2014. arXiv: 1312.6199.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. 2012.

[15] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. *Robust Physical-World Attacks on Deep Learning Models*. 2018. eprint: 1707.08945.

[16] Anurag Ranjan, Joel Janai, Andreas Geiger, and Michael J. Black. *Attacking Optical Flow*. 2019. arXiv: 1910.10053.

[17] Jenny Schmalfuss, Lukas Mehl, and Andrés Bruhn. "Attacking motion estimation with adversarial snow". In: (2022). eprint: 2210.11242.

[18] Carl Toft, Will Maddern, Akihiko Torii, Lars Hammarstrand, Erik Stenborg, Daniel Safari, Masatoshi Okutomi, Marc Pollefeys, Josef Sivic, Tomas Pajdla, Fredrik Kahl, and Torsten Sattler. "Long-Term Visual Localization Revisited". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

[19] Xing Xin, Jie Jiang, and Yin Zou. "A review of Visual-Based Localization". In: *Proceedings of the 2019 International Conference on Robotics, Intelligent Control and Artificial Intelligence*. 2019.

[20] Google Cloud. *What is Deep Learning? Applications & Examples | Google Cloud — cloud.google.com*. https://cloud.google.com/discover/what-is-deep-learning. [Accessed 17-09-2024].

[21] Mark Scapicchio Jim Holdsworth. *What Is Deep Learning? | IBM — ibm.com*. https://www.ibm.com/topics/deep-learning. [Accessed 17-09-2024]. 2024.

[22] Microsoft. *What Is Deep Learning? | Microsoft Azure — azure.microsoft.com*. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-deep-learning. [Accessed 17-09-2024].

[23] Amazon. *What is Deep Learning? - Deep Learning Explained - AWS — aws.amazon.com*. https://aws.amazon.com/what-is/deep-learning/. [Accessed 17-09-2024]. 2024.

[24] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation* (2006).

[25] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* (2012).

[26] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2019.

[27] Anu Aggarwal and B. Hamilton. "Training artificial neural networks with memristive synapses: HSPICE-matlab co-simulation". In: 2012.

[28] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* (1943).

[29] F. Rosenblatt. "The Perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* (1958).

[30] Magnus Borga. *Lecture 2 Supervised learning – Linear classifiers*. 2023.

[31] Paul Johns. “Chapter 7 - Synaptic transmission”. In: *Clinical Neuroscience*. Churchill Livingstone, 2014.

[32] Hongyu An, M. Amimul Ehsan, Zhen Zhou, and Yang Yi. “Electrical modeling and analysis of 3D synaptic array using vertical RRAM structure”. In: *2017 18th International Symposium on Quality Electronic Design (ISQED)*. 2017.

[33] M. Minsky and S. Papert. *Perceptrons*. MIT Press, 1969.

[34] F Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. 1961.

[35] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1998.

[36] Stephen Grossberg. “Recurrent neural networks”. In: *Scholarpedia* (2013).

[37] Kashu Yamazaki, Viet-Khoa Vo-Ho, Darshan Bulsara, and Ngan Le. “Spiking neural networks and their applications: A review”. In: *Brain Sciences* (2022).

[38] Bidyadhar Subudhi and Debashisha Jena. “A differential evolution based neural network approach to nonlinear system identification”. In: *Applied Soft Computing* (2011).

[39] Ahmad Jobran Al-Mahasneh, Sreenatha G Anavatti, and Matthew A Garratt. “The development of neural networks applications from perceptron to deep learning”. In: *2017 International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation (ICAMIMIA)*. IEEE. 2017, pp. 1–6.

[40] David E. Rumelhart and James L. McClelland. “Learning Internal Representations by Error Propagation”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. MIT Press, 1987.

[41] M.A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[42] D. H. Hubel. “Single unit activity in striate cortex of unrestrained cats”. In: *The Journal of Physiology* (1959).

[43] D. H. Hubel and T. N. Wiesel. “Receptive fields of single neurones in the cat’s striate cortex”. In: *The Journal of Physiology* (1959).

[44] D. H. Hubel and T. N. Wiesel. “Receptive fields and functional architecture of monkey striate cortex”. In: *The Journal of Physiology* (1968).

[45] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological Cybernetics* (1980).

[46] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. “Handwritten Digit Recognition with a Back-Propagation Network”. In: *Advances in Neural Information Processing Systems*. Morgan-Kaufmann, 1989.

[47] Yann LeCun, Yoshua Bengio, et al. “Convolutional networks for images, speech, and time series”. In: ().

[48] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* (1989).

[49] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* (1998).

[50] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[51] Erik G Learned-Miller. “Introduction to computer vision”. In: *University of Massachusetts, Amherst* (2011).

[52] Apple. *Working with Quaternions*. https://developer.apple.com/documentation/accelerate/simd/working_with_quaternions. [Accessed 22-10-2024]. 2024.

[53] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. *From Coarse to Fine: Robust Hierarchical Localization at Large Scale*. 2019. arXiv: 1812.03506.

[54] Shuai Chen, Yash Bhalgat, Xinghui Li, Jiawang Bian, Kejie Li, Zirui Wang, and Victor Adrian Prisacariu. *Neural Refinement for Absolute Pose Regression with Feature Synthesis*. 2024. arXiv: 2303.10087.

[55] Alex Kendall, Matthew Grimes, and Roberto Cipolla. *PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization*. 2016. arXiv: 1505.07427.

[56] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2015.

[57] Eric Brachmann and Carsten Rother. *Visual Camera Re-Localization from RGB and RGB-D Images Using DSAC*. 2020.

[58] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. "Scene coordinate regression forests for camera relocalization in RGB-D images". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013.

[59] Torsten Sattler, Akihiko Torii, Josef Sivic, Marc Pollefeys, Hajime Taira, Masatoshi Okutomi, and Tomas Pajdla. "Are large-scale 3d models really necessary for accurate visual localization?" In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

[60] Liu Liu, Hongdong Li, and Yuchao Dai. "Efficient global 2d-3d matching for camera localization in a large-scale 3d map". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017.

[61] Yunpeng Li, Noah Snavely, Dan Huttenlocher, and Pascal Fua. "Worldwide pose estimation using 3d point clouds". In: *European conference on computer vision*. Springer. 2012.

[62] Shimon Ullman. "The interpretation of structure from motion". In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 203.1153 (1979), pp. 405–426.

[63] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861.

[64] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. *SuperPoint: Self-Supervised Interest Point Detection and Description*. 2018. arXiv: 1712.07629.

[65] Paul-Edouard Sarlin, Ajaykumar Unagar, Måns Larsson, Hugo Germain, Carl Toft, Viktor Larsson, Marc Pollefeys, Vincent Lepetit, Lars Hammarstrand, Fredrik Kahl, and Torsten Sattler. *Back to the Feature: Learning Robust Camera Localization from Pixels to Pose*. 2021. arXiv: 2103.09213.

[66] Relja Arandjelovic and Andrew Zisserman. "All About VLAD". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013.

[67] Relja Arandjelović, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. *NetVLAD: CNN architecture for weakly supervised place recognition*. 2016. arXiv: 1511.07247.

[68] A. Torii, R. Arandjelović, J. Sivic, M. Okutomi, and T. Pajdla. "24/7 place recognition by view synthesis". In: *CVPR*. 2015.

[69] Hyo Jin Kim, Enrique Dunn, and Jan-Michael Frahm. "Predicting good features for image geo-localization using per-bundle vlad". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015.

[70] Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. "Deep supervised hashing for fast image retrieval". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[71] Leigh Metcalf and William Casey. "Chapter 2 - Metrics, similarity, and sets". In: *Cybersecurity and Applied Mathematics*. 2016.

[72] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762.

[73] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929.

[74] Alaaeldin El-Nouby, Natalia Neverova, Ivan Laptev, and Hervé Jégou. *Training Vision Transformers for Image Retrieval*. 2021. arXiv: 2102.05644.

[75] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015.

[76] S Hochreiter. "Long Short-term Memory". In: *Neural Computation MIT-Press* (1997).

[77] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. "What is the best multi-stage architecture for object recognition?" In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009.

[78] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. "Maxout networks". In: *International conference on machine learning*. PMLR. 2013.

[79] Hongshuo Liang, Erlu He, Yangyang Zhao, Zhe Jia, and Hao Li. "Adversarial attack and defense: A survey". In: (2022).

[80] Naveed Akhtar and Ajmal Mian. *Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey*. 2018. arXiv: 1801.00553.

[81] Siddhant Bhambri, Sumanyu Muku, Avinash Tulasi, and Arun Balaji Buduru. *A Survey of Black-Box Adversarial Attacks on Computer Vision Models*. 2020. arXiv: 1912.01667.

[82] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. *Adversarial Attacks and Defences: A Survey*. 2018. arXiv: 1810.00069.

[83] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. *Universal adversarial perturbations*. 2017. arXiv: 1610.08401.

[84] Dong C Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical programming* (1989).

[85] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. *DeepFool: a simple and accurate method to fool deep neural networks*. 2016. arXiv: 1511.04599.

[86] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. *Adversarial examples in the physical world*. 2017. arXiv: 1607.02533.

[87] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. *Generating Adversarial Examples with Adversarial Networks*. 2019. arXiv: 1801.02610.

[88] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. "One Pixel Attack for Fooling Deep Neural Networks". In: *IEEE Transactions on Evolutionary Computation* (2019).

[89] Eric Brachmann, Tommaso Cavallari, and Victor Adrian Prisacariu. *Accelerated Coordinate Encoding: Learning to Relocalize in Minutes using RGB and Poses*. 2023.

[90] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. "Efficient Effective Prioritized Matching for Large-Scale Image-Based Localization". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017).

[91] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* (1981).

[92] Bert M Haralick, Chung-Nan Lee, Karsten Ottenberg, and Michael Nölle. "Review and analysis of solutions of the three point perspective pose estimation problem". In: *International journal of computer vision* 13 (1994), pp. 331–356.

[93] F.M. Dekking. *A Modern Introduction to Probability and Statistics: Understanding Why and How*. 2005.

[94] Gareth James. *An introduction to statistical learning*. 2013.

[95] Ben Glocker, Shahram Izadi, Jamie Shotton, and Antonio Criminisi. "Real-time RGB-D camera relocalization". In: *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE. 2013, pp. 173–179.

[96] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. "Kinectfusion: Real-time dense surface mapping and tracking". In: *2011 10th IEEE international symposium on mixed and augmented reality*. Ieee. 2011, pp. 127–136.

[97] Katarzyna Janocha and Wojciech Marian Czarnecki. *On Loss Functions for Deep Neural Networks in Classification*. 2017. arXiv: 1702.05659.

[98] Juan Terven, Diana M. Cordova-Esparza, Alfonso Ramirez-Pedraza, Edgar A. Chavez-Urbiola, and Julio A. Romero-Gonzalez. *Loss Functions and Metrics in Deep Learning*. 2024. eprint: 2307.02694.

[99] Tanwi Mallick, Partha Pratim Das, and Arun Kumar Majumdar. "Characterizations of noise in Kinect depth images: A review". In: *IEEE Sensors journal* (2014).

[100] Yuna Huh Wong, John Yurchak, Robert W. Button, Aaron B. Frank, Burgess Laird, Osonde A. Osoba, Randall Steeb, Benjamin N. Harris, and Sebastian Joon Bae. *Deterrence in the Age of Thinking Machines*. Santa Monica, CA: RAND Corporation, 2020.