POLITECNICO DI TORINO

Collegio di Ingegneria Elettronica, delle Telecomunicazioni e Fisica (ETF)

Master degree course

COMMUNICATIONS AND COMPUTER NETWORKS ENGINEERING

Master Thesis

Neural Networks for image classification

An approach to adversarial perturbations robustness

Thesis Supervisor:
Prof. Enrico Magli

Candidate:
Luca Volpato
251586

December 2019

Abstract

The following document contains the analysis of a method for image classification problems for neural networks, developed with the goal of improving robustness to adversarial perturbations.

Through the use of an encoder, the system maps the input data to distributions with target mean and variance, inside of a latent space with dimensionality equal to the number of classes. The idea is that, by having well separated distributions, adversarial attacks will prove less effective. The system was already developed for one-vs-all class classification [1], consequently this thesis explores the results and methods for a multi-class extension.

We performed analyses on the MNIST and CIFAR-10 datasets, and the outcomes obtained are solid enough for an extension to other databases. Indeed, the results prove that a system as the one presented is consistently more resistant to adversarial perturbations compared to standard cross-entropy schemes, while providing comparable levels of accuracy when perturbations are absent.

Contents

Chapter 1

Context

This Introduction is meant for all those who've never heard about neural networks or don't know them well enough to understand a scientific paper, as well as those who just need some reminders about the specific topics faced in this thesis. It contains notions structured on various degrees of difficulty, ranging from history and application of neural networks to their technical implementation, and is therefore meant to be a meeting point for all possible public of the document.

Readers should feel free to skip, at their own discretion, sections they consider too technical or too obvious, even the whole Introduction, if they feel confident enough.

Overview

Neural networks are a type of artificial intelligence that in recent years has found many applications in the field of automatic image recognition, mostly due to their performances equal, if not superior, to those of humans.

However, together with impressive successes and achievements, they also attracted a fair amount of challenges; this thesis will focus in particular on obstacles posed by what is usually referred to as *adversarial perturbations*, endeavors that attempt to disrupt the intended mechanisms of neural networks, leading them to recognize images incorrectly, often with malicious intents.

Motivations

Colleagues at Politecnico di Torino in *Learning mappings onto regularized latent spaces for biometric authentication* [1] had developed a system meant to resist adversarial perturbation in the biometric image recognition field, where the input images were labeled as *Authorized* and *Non-Authorized*. When confronted with adversarial attacks aimed at the misclassification of inputs, the system proved significantly more robust to errors than many of its state-of-the-art equivalents.

This thesis focuses on implementing and studying a generalization of such a system,

expanding the number of classes above the binary dimensionality, up to an arbitrarily selected amount.

1.1 About neural networks

Neural networks belong to a type of computer algorithms commonly referred to as artificial intelligence, whose peculiarity is that they start functioning with absolutely no knowledge of their assigned task, and gradually learn the rules through many trials and errors. The programmer isn't required to specify all possible cases, solutions, and exceptions, that the machine could encounter, but instead he just needs to lay the correct foundations for the algorithm to improve and solve the presented task.

Neural networks, in particular, are modeled loosely after the human brain, and its ability to recognize patterns. The broad idea behind it is that by being confronted with a sufficient amount of specific cases, the system will eventually learn the general underlying rules: for example, in the field of image recognition, which is the topic of this thesis, neural networks might learn to identify cats by analyzing many examples that have been manually marked as *cat* or *no cat*, and use the results to identify such animal in other images. They do this without any prior knowledge of the features cats possess, for example, that they have fur, tails, four legs, etc. Instead, they automatically generate identifying characteristics from the examples that they process.

Despite the concept having been around since 1943, neural networks have encountered a major popularity surge only in the latest decade, when the technological improvements finally allowed the implementation of competitive models. These systems have proved to achieve human or even super-human performances in many fields: image recognition, face authentication, self-driving cars, generation of human-like features for cinematic special effects, speech recognition, automatic translation, medical diagnosis, game playing, and many others activities that have for a long time been considered exclusive to humans.

1.1.1 The MNIST database

Neural networks sometimes learn and memorize by hearth all the solutions of the assigned task, a phenomenon called *overfitting*, which causes poor performances in realistic scenarios; for this reason input data is usually divided into two sets, one for *training*, the phase when the network improves and learns, and one for *testing* the phase apt to verify the system's performance, and to avoid overfitting, the samples used for testing should never be the same used for training.

The amount of data necessary for the training phase is usually quite large, and it would be unpractical to generate it or collect it each time it's needed. Historically it was deemed convenient to have previously arranged data ready to be fed to the

algorithms, and thus several datasets were created, each with its own characterizing.

For the image processing field, the most known of such collections is the MNIST (Modified National Institute of Standards and Technology) dataset, composed of 70,000 handwritten digits between 0 and 9, separated in 60,000 for training and 10,000 for testing. Each image is composed of 28 x 28 pixels, with pixel values ranging from 0 to 1.

Commonly neural networks are employed to solve *classification problems*, where the input data has to be correctly separated into categories, called *classes*, each with its peculiar features; for the MNIST dataset each digit corresponds to a class, and the system's objective is to correctly identify them. Accordingly, each image is manually labeled with the digit it represents. This way the neural network algorithms can verify the correctness of their classification, and use it to improve. In Figure 1.1 some MNIST samples are displayed, separated by class.

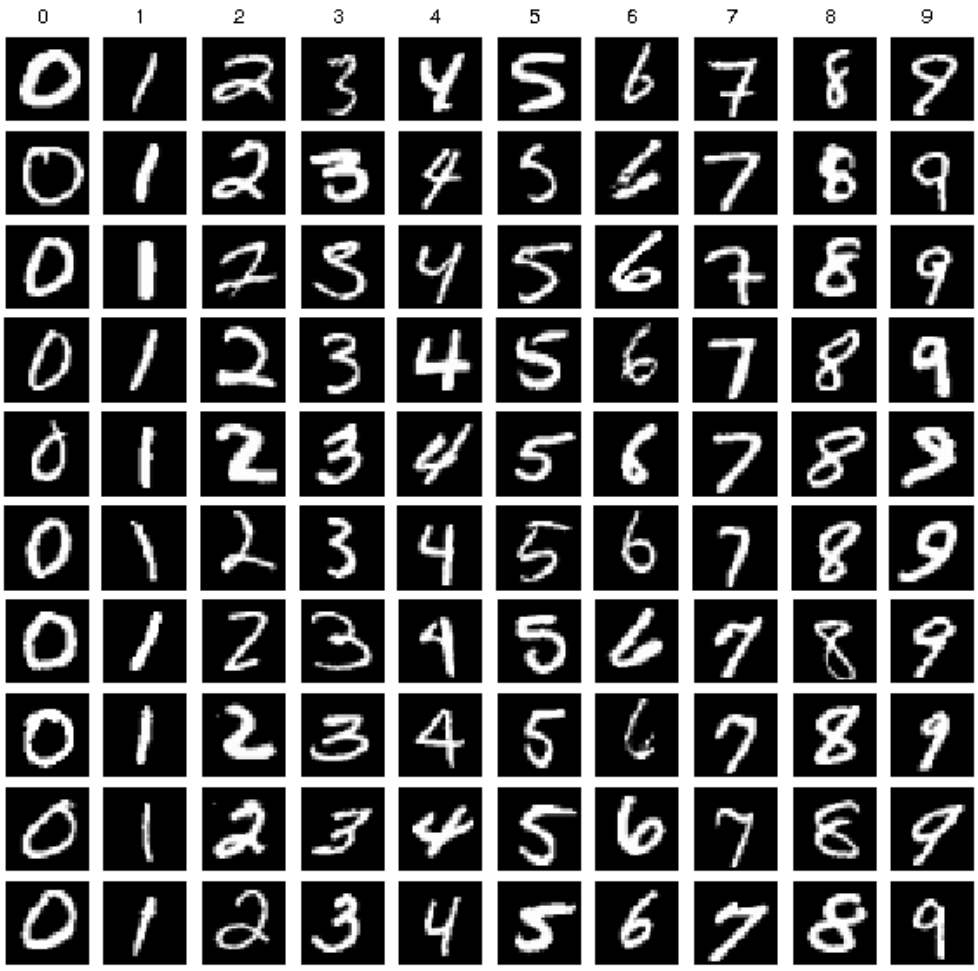


Figure 1.1: Ten MNIST images for each class, organized by columns

1.1.2 Neural Networks implementation

Neural networks are composed of many interconnected units, called artificial neurons or nodes, whose mechanisms are inspired by true neurons in human brains. In fact, the connections can transport a signal to other neurons, similarly to the synapses in a biological brain. Artificial neurons are able to receive such signal, process it, and re-transmit it to adjacent units.

The connections between neurons are commonly called edges, and they are characterized by a parameter *weight*, a measure of the importance that connection has in the decision process. Neurons too are associated with a parameter, named *bias*, a threshold value that is usually added to the input signal. The process of learning consists in tweaking the weights and biases until the neuron produces the intended output.

The output of each neuron is generated according to some mathematical function, of which the most basic example is the equation:

$$y = b + \sum_i w_i x_i \tag{1.1}$$

Where:

- y is the neuron's output
- b is the neuron's bias
- x_i is the input signal received from the i-th edge
- w_i is the weight of the i-th edge

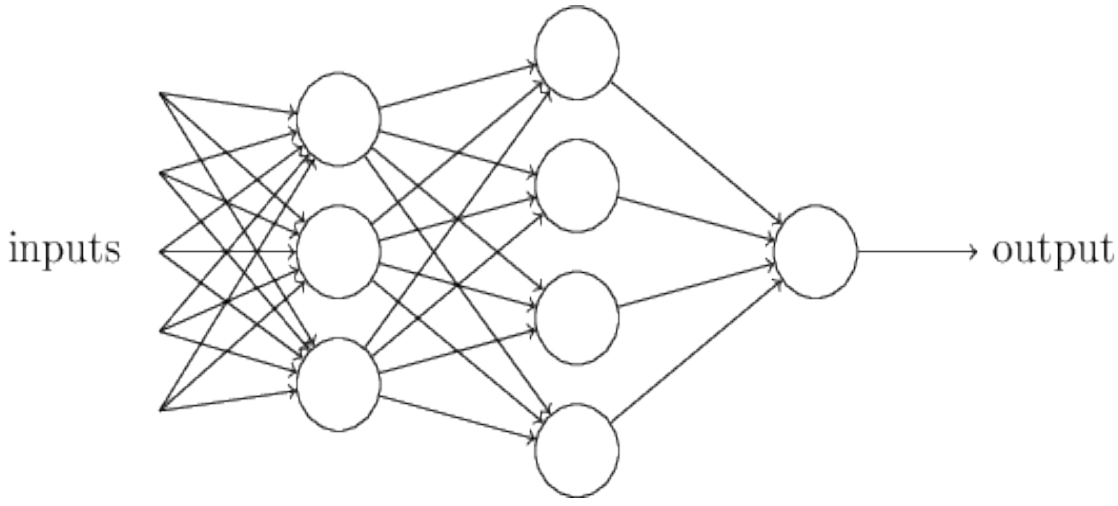


Figure 1.2: An example of neural network

Typically, neurons are aggregated into layers, which are positioned in sequence, and

the signal crosses this whole structure from one end to the other. There are many types of layers, and each of them performs on the inputs a different function, analogously to equation (1.1).

When organized this way, neurons form architectures similar to Figure 1.2, and as can be imagined, they are extremely hard to monitor in detail, but are also able to simulate very complex systems if set up correctly.

The loss function

We explained that weights and biases are the parameters that need to be changed for the network to learn, but we never mentioned if they should be increased or decreased, and by which amount. These issues are indirectly answered by the implementation of a loss function, a mathematical equation that connects all possible network variables into a single value representing the distance from the perfect performance, or how much the system is "losing" from its optimal state. When the network has poor accuracy, the loss function assumes a relatively high value, which gradually decreases as the system learns, ideally reaching zero at the end of training.

This decreasing process is entrusted to some algorithms characteristic of neural networks and the operational research field; there are several of them, but they are unnecessarily complex for this discussion, and therefore the reader is invited to appraise them of its own accord. Let it just be said that they are powerful tools that, given a function and the input data, are able to find the minimum point for the loss and enforce the corresponding values on bias and weights.

The choice of a loss function is one of the most critical decisions that fall on the programmer's shoulders, because an inappropriate one can have on the network effects extremely different from the ones intended. In fact, the loss function could be summarized as the assortment of aspects that the system will consider important during its learning process. This whole thesis is the study of the loss functions presented in Chapter 2, and their effects on the system.

1.1.3 About encoders

Encoders are a specific type of neural networks, the very one that we will employ for this thesis. Their aim is to learn a mapping (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal "noise". Usually, along with the reduction side there is a reconstructing side, which attempts to generate from the reduced encoding a representation as close as possible to the original input; this form is called *autoencoder*, but in this thesis, we'll only use the reduction side of encoders.

As an example, consider the MNIST images, whose dimensionality is 28 x 28 pixels, for a total of 784: this amount can be reduced with encoders, which map all possible

combinations of inputs with just ten outputs, one for each of the MNIST classes; each image could be represented with an ordered array of values, where the position of the highest value identifies the corresponding label. Given this encoding, each slot of the array could be imagined as a physical dimension, and thus the images could be represented into a *latent space*, as explained by Testa et al.[1].

The latent space representation is a crucial notion for this thesis, mostly because when the dimensionality is ≤ 3 the mappings can be depicted in a three-dimensional plot, whose interpretation is significantly simpler compared to the analysis of the numeric values for each dimension.

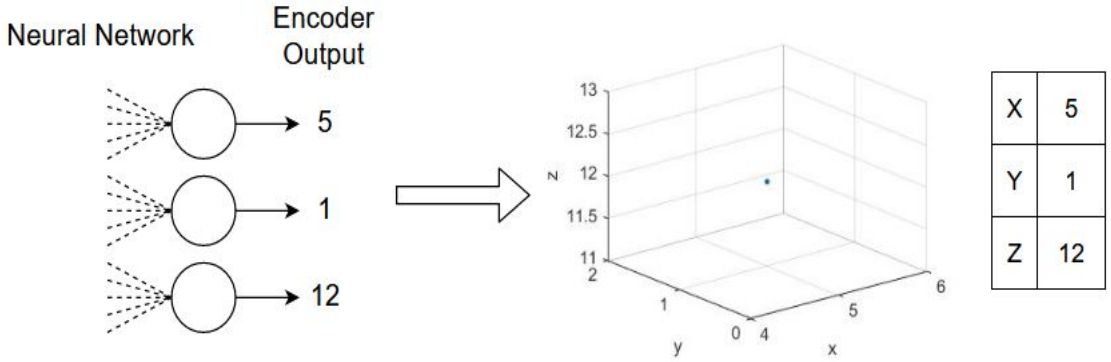


Figure 1.3: An example of three-dimensional mapping plot. The number of classes corresponds to the latent space dimensionality.

1.2 About adversarial perturbations

Adversarial perturbations were first presented in 2014 in *Intriguing properties of neural networks* [6], where it was explained that neural networks can easily be tricked into incorrect classification by being fed altered inputs, with modifications invisible to the human eye, but critical for a digital system. These alterations, called *noise* because of their seemingly pseudo-random nature, are created with the same sign of the loss function's gradients with respect to the inputs. A very explanatory example, taken from *Explaining and harnessing adversarial examples* [2], can be found in Figure 1.4.

In 2017, the paper *Adversarial Examples in the Physical World* [3] demonstrated that adversarially generated images are misclassified even when printed out on paper, proving that the phenomenon is relevant in both the digital and the physical domains.

Accessorize to a Crime Real and Stealthy Attacks on State-of-the-Art Face Recognition [4] showed that facial recognition software can be fooled by wearing adversarial glasses, with the result of being mistaken for someone else or avoiding being recognized at all.

Yet another paper, *Adversarial Patch* [5], demonstrated how to generate a patch that can be placed anywhere within the field of view of the neural network and cause

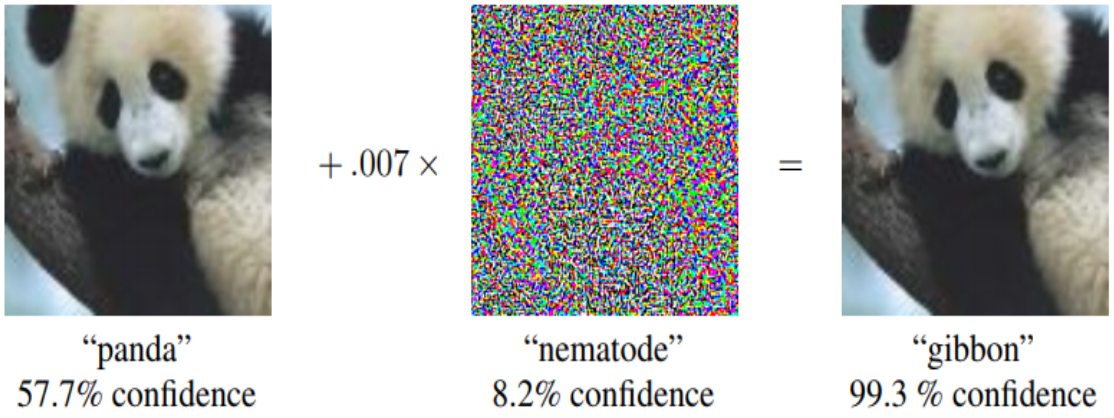


Figure 1.4: A demonstration of fast adversarial example generation. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, the system's classification of the image can be changed. Here noise power of .007 is invisible to the human eye.

it to output a targeted class. Quoting the authors, “this attack was significant because the attacker does not need to know what image they are attacking when constructing the attack. After generating an adversarial patch, this could be widely distributed across the Internet for other attackers to print out and use.”

One peculiarity of all these adversarial attacks is the high confidence level that often characterizes the erroneous classification. Furthermore, in many cases the attacks don't depend much on the specific deep neural network used for the task, meaning that multiple classifiers assign the same (incorrect) class to the same adversarial sample.

Adversarial perturbations can be classified according to the characteristics of their execution:

- in **white-box attacks** the attacker has access to the system parameters, while in **black-box attacks** it uses a different model, or no model at all, to generate adversarial images with the hope that these will affect the target model.
- **non-targeted attacks** only focus is to enforce the adversarial image's misclassification, while in **targeted attacks** the attacker attempts to get the image classified as a specific target class, different from the true class.
- in **one-shot attacks** the images are modified only once in the direction of the gradient, while in **iterative attacks** it happens recursively.

1.2.1 Defenses against adversarial perturbations

Adversarial perturbations have proven a formidable rival for neural networks, and no solid defense mechanism has yet been developed. Some partial successes only lead to a

further improvement of technologies and algorithms on the attacker's side. Of course, these are some of the reasons that encouraged the development of our system. At any rate, this section will report the most common defense methods and the reason why they are not considered effective.

The most brute-force approach to defend against adversarial perturbations is **adversarial training** [2]: it consists in simulating the attacks with a number of generated adversarial examples against the network, and then explicitly train the model to resist them. This improves the model's generalization but hasn't been able to provide a meaningful level of robustness, and in fact, it proved to be just a game where attackers and defenders constantly try to double guess each other.

Another approach is represented by **defensive distillation** [7], consisting in training a secondary model whose surface is smoothed in the directions an attacker will typically try to exploit, making it difficult for them to discover adversarial input tweaks that lead to incorrect categorization. The second model is trained on the primary model's soft probability outputs, rather than the hard true labels from the real training data. This technique was shown to have some success defending against initial variants of adversarial attacks but has been beaten by more recent ones.

1.3 Residual Networks

In the future chapters of this thesis we will often refer to the employed neural network as ResNet [8]; this stands for Residual layer Network, a specific type of neural networks. Their peculiarity is to use skip connections to jump over some layers. Typical ResNet models are implemented with double or triple layer skips that contain nonlinearities (ReLU) and batch normalization in between.

One motivation for skipping over layers is to avoid the problem of vanishing gradients, by reusing activations from a previous layer (the residual) until the adjacent layer learns its weights. During training, the weights adapt to mute the adjacent layer and amplify the previously-skipped layer.

Skipping effectively simplifies the network, using fewer layers in the initial training stages. This speeds learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through. The network then gradually restores the skipped layers as it learns the feature space. A neural network without residual parts explores more of the feature space. This makes it more vulnerable to perturbations that cause it to stray from the intended features, and it necessitates extra training data to recover.

1.4 About RegNet

As already mentioned, the choice of a loss function is a critical step in the implementation of neural networks, because it determines the system's behavior. This thesis expands on the RegNet system, a particular loss function that Testa et al., PhDs and professors at Politecnico di Torino, developed in *Learning mappings onto regularized latent spaces for biometries authentication* [1].

RegNet is a method that learns to map of the input biometries onto a distribution with target mean and variance; it creates a well-behaved space in which users can be separated by means of simple and tunable boundaries. More specifically, authorized and unauthorized users are mapped onto two different and well-behaved Gaussian distributions. While typical classification methods learn complex boundaries, RegNet learns mappings that can be easily shaped and modified, and are much simpler to analyze.

The center of mass of the two classes of users should be far enough from each other to allow identification based on a simple thresholding decision rule, therefore the target mean of the distributions is a critical parameter that heavily affects performances. To ensure that the distribution shape doesn't interfere with the classification of users mapped far from each other, the latent space in forced to be shaped in a simple and well-behaved manner (specifically, to follow Gaussian distributions).

The loss functions that make all of this possible will be addressed in Chapter 2, where we assume a more technical approach to this whole discussion. An example of RegNet's mapping can be observed in Figure 1.5.

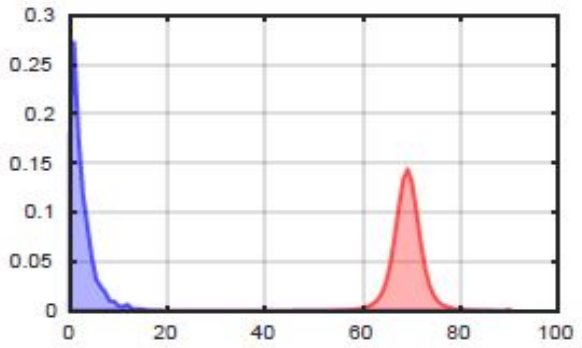


Figure 1.5: An example of RegNet mappings taken from [1]. The authorized users (blue) and the unauthorized ones (red) are mapped approximating Gaussian distributions and can be easily distinguished.

1.5 About this thesis

The purpose of this study is to further our understanding of RegNet and expand it to other uses. Testa et al. [1] implemented their system with a particular focus on the biometrics field, aiming to the correct identification of just the two classes *authorized* and *unauthorized*; our goal is instead to make RegNet usable on datasets containing all sort of subjects and any number of classes.

For the very nature of neural networks, capable of learning on their own the important features of the inputs, the use of a dataset different than the ones used by Testa e al. [1] is not an issue, and it will only impact the complexity or depth of the encoder. Instead, the study of results with increased class dimensionality is more meaningful; it could reveal inconsistencies in the theoretical part of RegNet's formulation and deviations from the two dimensional scenario.

With this in mind, at first we analyzed a simple increase of dimensionality, from two to three classes, modifying the latent space from a plane like in Figure 1.5 to a 3D environment as the one of Figure 1.3. We already mentioned that in this type of representation it's simple to compare the distributions with the ones obtained by Testa et al. [1], and to make sure that the spaces are well regularized as the theory suggests. Indeed, we found the theoretic system to be excessively optimistic in some sections, neglectful in others, probably because, at the moment of formulation, an N-dimensional environment wasn't being taken into consideration.

Secondly, we studied the effects of such inconsistencies in order to understand their impact on the performances, and we searched for solutions that could prove effective without completely altering the system's concept.

Finally, several papers addressing the adversarial field, such as [9], suggest employing a mapping like the one performed by RegNet to increase adversarial perturbation robustness. This was the original intention of this thesis, but we deemed appropriate to beforehand verify the intended functioning of the system in generalized predicaments, hence the formulation of Chapter 2. In Chapter 3 we return to the original purpose and take on the challenge posed by adversarial perturbations with considerable success.

Chapter 2

Study of three-dimensional latent space

RegNet had originally been developed for authentication purposes in the biometrics field with the classes *authorized* and *unauthorized*.

With this premise, in the process of a multi-class implementation, we deemed wise to advance gradually, and decided that the foremost thing to do was verifying the correct behavior of the system with three classes. In three dimensions the latent space can still be easily analyzed and represented, while for higher dimensions it becomes increasingly difficult to observe the distributions.

For this Chapter our objective is to obtain well behaved Gaussian distributions, correct mappings to the target mean, regularized latent spaces with simple boundaries, as well as an accuracy rating comparable to the standard cross-entropy method.

2.0.1 RegNet loss functions

The RegNet system maps the input images to a d-dimensional latent space. Each class is theoretically mapped to a portion of the space identified by a target mean value μ_T on the dimension corresponding to that class, and zero on all others; each class also has a target variance σ_T. The corresponding μ_O and σ_O are calculated from the output samples and are theoretically enforced to be equal to the target μ_T and σ_T by the RegNet loss function. The original RegNet paper [1] proposes for each class x the loss:

$$\mathcal{L}_{cov,\,x} = \frac{1}{x}\left[\log\frac{|\boldsymbol{\Sigma}_{Tx}|}{|\boldsymbol{\Sigma}_{Ox}|} - d + \mathrm{tr}(\boldsymbol{\Sigma}_{Tx}^{-1}\boldsymbol{\Sigma}_{Ox}) + (\boldsymbol{\mu}_{Tx} - \boldsymbol{\mu}_{Ox})^{\boldsymbol{T}}\boldsymbol{\Sigma}_{Tx}^{-1}(\boldsymbol{\mu}_{Tx} - \boldsymbol{\mu}_{Ox})\right] \tag{2.1}$$

Overall, the loss function for a d-dimensional space comes to be:

$$L_{TOT} = \sum_{i=0}^{d} \mathcal{L}_{cov,\,i} \tag{2.2}$$

The covariance matrices increase complexity and computation time; when the loss is minimized, they should be reduced to diagonal matrices, and with this assumption the equation can be approximated. Therefore the RegNet paper proposes an alternative, simplified loss:

$$\mathcal{L}_{simpl,\,x} = \frac{1}{x}\left[\log\frac{\sigma_{Tx}^{2d}}{\Pi_i \boldsymbol{\Sigma}_{Ox}^{(ii)}} - d + \frac{\sum_i^d \boldsymbol{\Sigma}_{Ox}^{ii}}{\sigma_{Tx}^2} + \frac{||\boldsymbol{\mu}_{Tx} - \boldsymbol{\mu}_{Ox}||^2}{\sigma_{Tx}^2}\right] \tag{2.3}$$

$$L_{TOT} = \sum_{i=0}^{d} \mathcal{L}_{simpl,\,i} \tag{2.4}$$

Equation (2.3) is an approximated version of (2.1), conceived to assume value equal to it only when the network is in an ideal state and both losses are close to their minimum. In the two-dimensional case, differences between their performances were hardly noticeable, therefore we executed our first set of experiments choosing the approximation formula (2.3) as the loss of our network, hoping it would keep proving as accurate as in the 2D case.

2.0.2 Architecture details

Having to process image data of various nature, we used a convolutional neural network as the encoder architecture. More specifically, we use a ResNet-ν architecture, similarly to the approach adopted by Testa et al.[1] [8]. ResNet is made of four blocks, each of them consisting of an increasing number of 3 x 3 filters. The last layer is a fully connected layer which maps the output of the last filter to z, the d-dimensional latent representation. Refer to Figure 2.1.

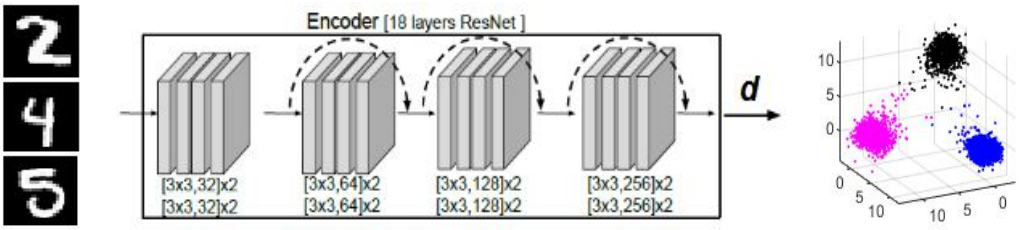


Figure 2.1: ResNet-18 architecture. The encoder, taking as input a batch of images, for each of them produces a d-dimensional output which can be considered as d separate dimensions. In this figure d=3 leads to each image being plottable in a 3D latent space.

The parameter ν represents the depth of each residual layer; we used ResNet-18, in the figure above, for most of our experiments; the only exception is Section 3.3.4, where we'll analyze the impact of residual layer depth on the system's robustness.

The parameter d, the dimensionality of the latent space, was set to 3 for all simula-

tions on three classes, to 10 when using the full MNIST/CIFAR-10 datasets. We also performed experiments with $3 < d < 10$, but they didn't produce notable results and aren't reported in this document.

2.1 Simplified loss

A first study was carried on the ten classes of the MNIST dataset by separating it in groups of three classes each (leaving the remaining digit out), similarly to what was done in the 2D case, where the classes were aggregated under the labels of *authorized* and *unauthorized*.

2.1.1 Experiment execution

For all results in Section 2.1 we executed the experiments with the following configuration:

- Dataset: MNIST, three classes, either [0,1,2] [3,4,5] [6,7,8] or [0] [1] [2].
- Encoder structure: ResNet-18.
- Optimizer: Adam optimizer and stochastic gradient descent.
- Iterations: 30000 batches.
- Batch size: 200.
- Learning rate: starts as $\lambda = 10^{-1}$, becomes $\lambda = 10^{-2}$ at $i = 1000$, $\lambda = 10^{-3}$ for $i = 5000$, $\lambda = 10^{-4}$ for $i = 10000$, $\lambda = 10^{-5}$ for $i = 20000$.
- Loss function: the one identified by (2.4).
- Latent space dimensionality: $d = 3$.
- Target mean: either $\mu_T = 10$ or $\mu_T = 70$.
- Target variance: $\sigma_T = 1$.
- For each class the encoder is supposed to produce an array of zeros containing a single μ_T value in the position identifying the class (e.g. for the second class the output should be [0,μ_T,0]).

2.1.2 Analysis of the latent space

Figure 2.2 shows the first results for this experiment: the distributions are mapped reasonably well to the target mean, but their spread is not Gaussian. The presence of gaussianity is particularly important because it would prove that the loss is able to regularize the latent space, as theorized, and also that there is a lack of interdependence among the classes; only if these assumptions are true, the simplified loss can be considered a good approximation of the true loss, and the result of Figure 2.2 show that these conditions are lacking.

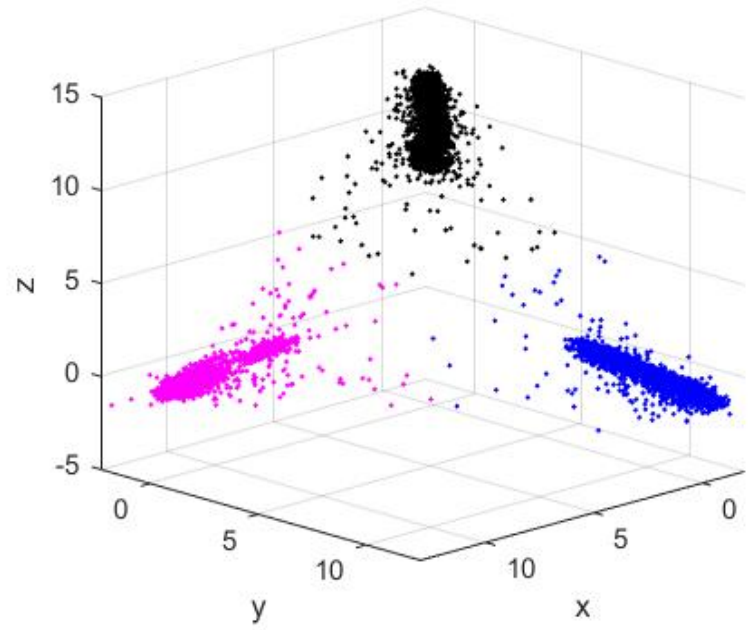


Figure 2.2: Latent space for MNIST classes [0,1,2] [3,4,5] [6,7,8] and μ_T=10

For good measure, the test was repeated with an increased target mean, μ_T=70, as shown in Figure 2.3. In this case the distributions appear more Gaussian. It seems that

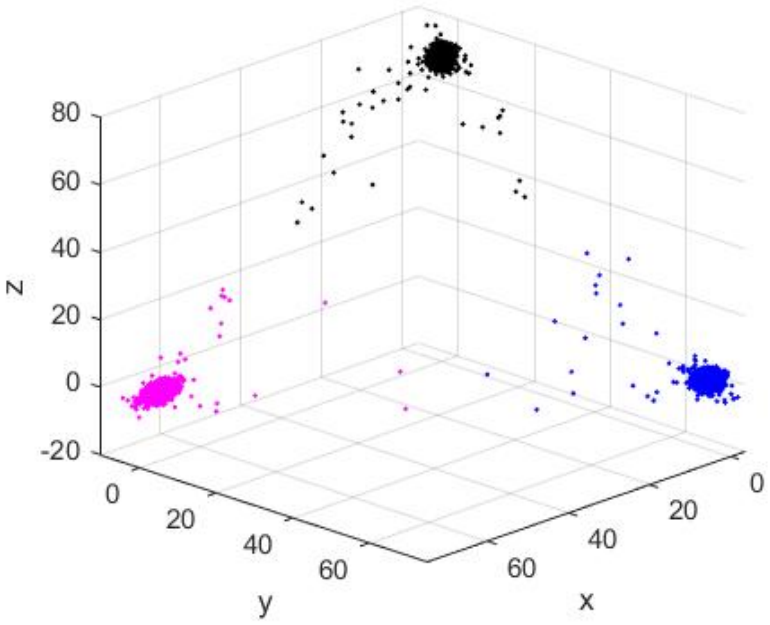
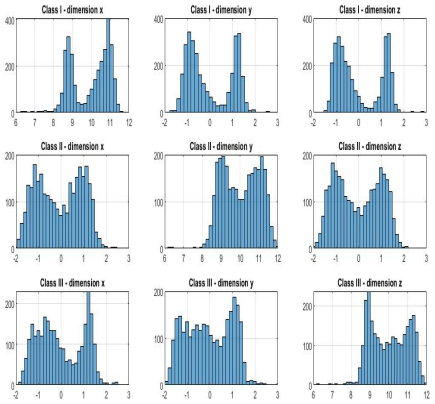


Figure 2.3: Latent space for MNIST classes [0,1,2] [3,4,5] [6,7,8] and μ_T=70

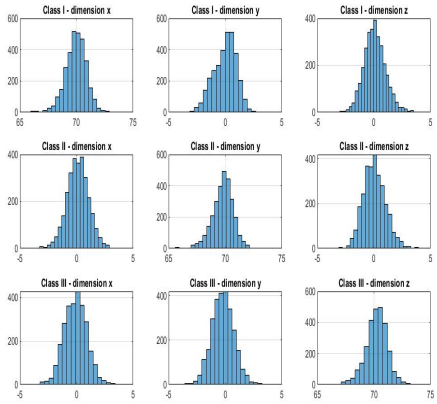
a greater mapping distance improves the overall situation, but further examination will be performed in the next sections.

2.1.3 Analysis of Gaussianity

As can be observed from Figures 2.4a and 2.4b, for a low μ_T such as the one standardly used in the 2D case, the distributions of Figures 2.2 and 2.3 are far from Gaussian, and furthermore seem to form two separate spikes on the sides; for higher means instead the histograms appear closer to an actual Gaussian curve; not having employed data augmentation techniques for these first sets of experiments, we considered such shapes almost acceptable, justifying the deviation from an actual normal curve (and therefore from a real random process) with the limitedness of the input data. However, as will become clear in the next sections, we took several steps to improve the distributions' Gaussianity.



(a) Spread of dimensions for μ_T=10



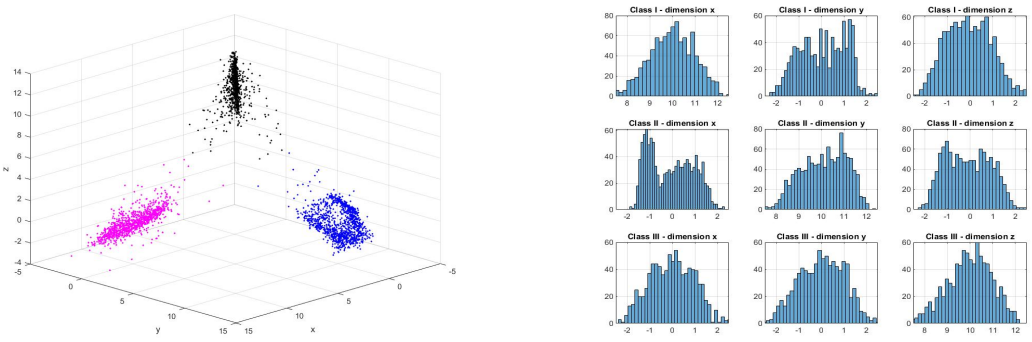
(b) Spread of dimensions for μ_T=70

Figure 2.4: Histograms of dimensions for classes [0,1,2] [3,4,5] [6,7,8]

2.1.4 Results with single label classes

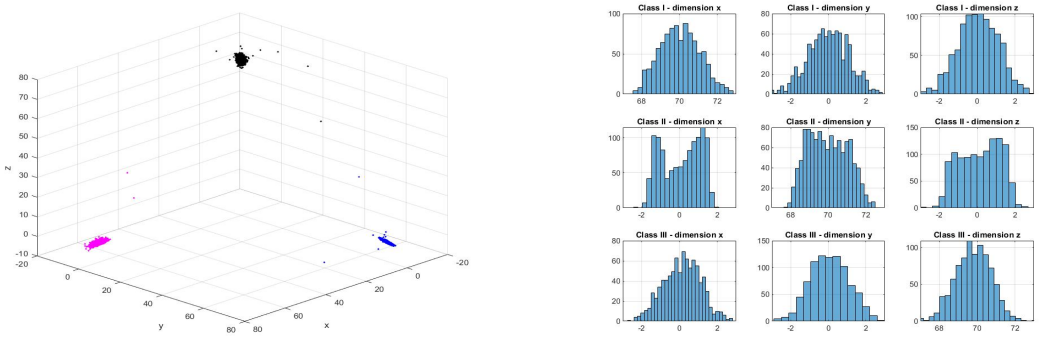
Our objective is to correctly identify each class of the dataset independently, and so we repeated the former experiments to study the latent space and the gaussianity of the distributions with classes corresponding to single MNIST digits. All other variables and parameters remain equal to the ones addressed at the beginning of 2.1. It should be mentioned that simulations with different combinations of classes do not yield significant differences among each other, and therefore, unless specified, the presented results can be assumed as a median sample of the whole category.

Regardless, from Figures 2.5 and 2.6 it can be evinced that training with single label classes doesn't hold much difference with what has been presented so far: the histograms of dimensions, in particular, for $\mu_T = 10$ are far from the Gaussian distributions we expected, and seem to improve slightly for higher μ_T.



(a) Latent space **(b)** Histograms of dimensions

Figure 2.5: Latent mapping for μ_T=10 and classes [0],[1],[2]



(a) Latent space **(b)** Histograms of dimensions

Figure 2.6: Latent mapping for μ_T=70 and single label classes [0],[1],[2]

2.1.5 Interpretation of abnormal distribution shapes

We deemed necessary to find an interpretation for the unusual distribution shapes, especially for such heavy tails in curves that are supposed to be Gaussian; the elongated shapes that are visible when μ_T=10 reflect in the histograms having heavy tails or flat peaks.

Figures 2.7 and 2.8 show some randomly selected MNIST images labeled as 1, and how they are mapped in the distribution of the respective class. A quick analysis reveals that the digits traced in a specific way, on the diagonal from top-left to bottom-right, are mapped at one extreme of the distribution, and the ones traced from top-right to bottom-left to the opposite one; therefore the network tends to separate the digits depending on the way they are traced, possibly also due to the excessive simplicity of the problem assigned. This anomaly can be easily identified for the digit 1, whose morphologies are limited; such conclusions can be more articulated in case of more complex digits, or when the classes are groups of labels, but still we believe this quick insight holds some validity even in those occurrences.

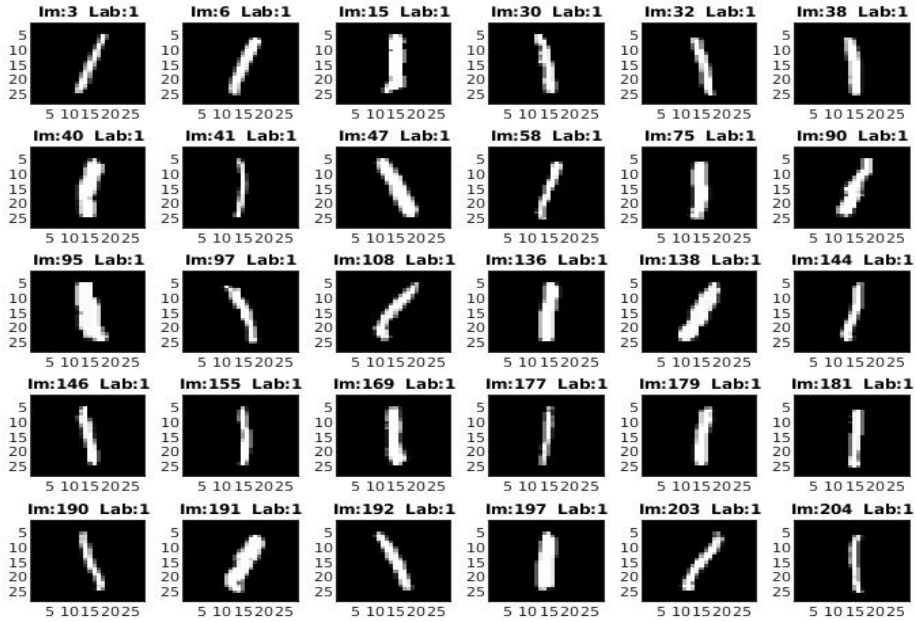


Figure 2.7: Some MNIST digits labeled as 1

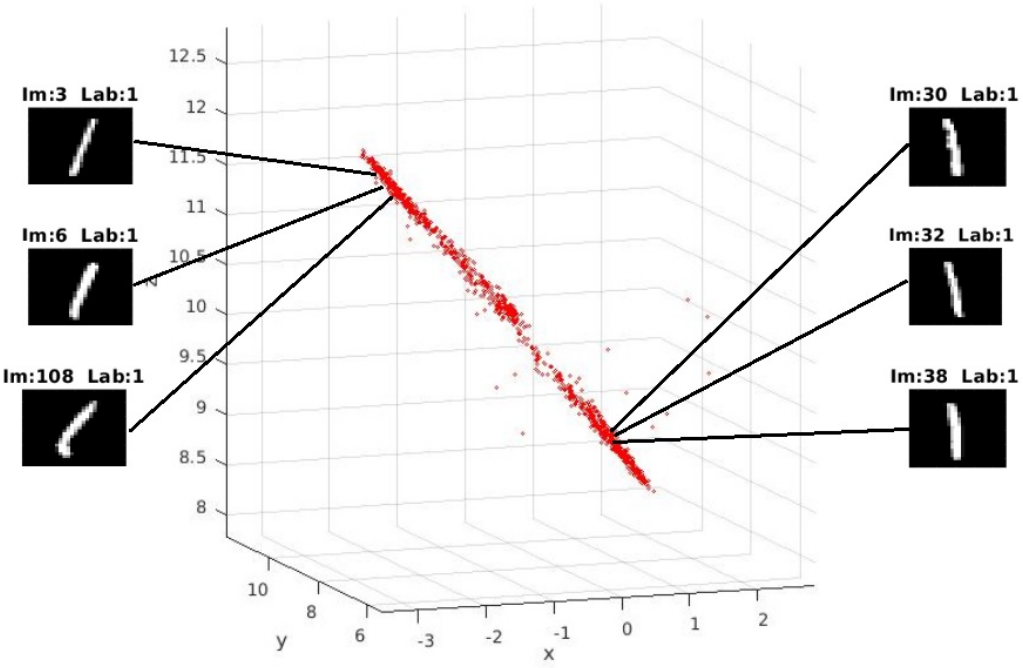


Figure 2.8: Latent distribution of a class containing all 1 MNIST digits

2.2 Covariance loss

Due to the partially unexpected results of experiments with the simplified loss function, we decided to also test the original formula (2.1) containing the full covariance matrices for each class; the minimization of the loss should force said matrices to be diagonal, pushing all cross-correlation terms to zero and thus ensuring independence among the classes.

It should be mentioned that for all experiments from this point onward, only single label classes were used, since the grouped classes were just a preliminary approach to smoothly transition from two-dimensional environments to the three-dimensional ones.

Experiment execution

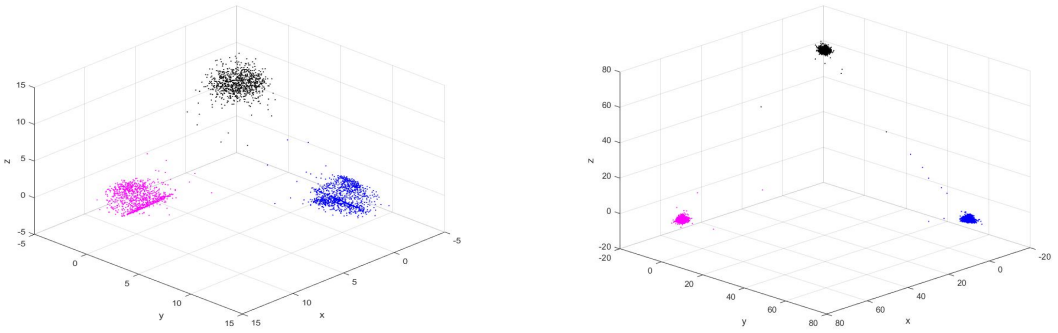
For all results in Section 2.2 we executed the experiments with the following configuration:

- Dataset: MNIST, classes [0] [1] [2].
- Encoder structure: ResNet-18.
- Optimizer: Adam optimizer and stochastic gradient descent.
- Iterations: 30000 batches.
- Batch size: 200.
- Learning rate: starts as $\lambda = 10^{-1}$, becomes $\lambda = 10^{-2}$ at $i = 1000$, $\lambda = 10^{-3}$ for $i = 5000$, $\lambda = 10^{-4}$ for $i = 10000$, $\lambda = 10^{-5}$ for $i = 20000$.
- Loss function: the one identified by (2.2).
- Latent space dimensionality: $d = 3$.
- Target mean: either $\mu_T = 10$ or $\mu_T = 70$.
- Target variance: $\sigma_T = 1$.
- For each class the encoder is supposed to produce an array of zeros containing a single μ_T value in the position identifying the class (e.g. for the second class the output should be [0,μ_T,0]).

2.2.1 Distributions in the latent space

The distributions in Figure 2.9 are shaped as spheroids and are equally distributed in all dimensions. However, the close-up of Figure 2.10 shows that, while considerably improved, the distributions are still not Gaussian.

We repeated the experiment several times to evaluate the frequency of such inconsistency and concluded that it is quite a common occurrence. Therefore even the covariance loss doesn't ensure perfectly Gaussian distributions.



(a) Latent space for μ_T=10 **(b)** Latent space for μ_T=70

Figure 2.9: Latent spaces after training with loss function containing covariance matrices - single label classes [0],[1],[2]

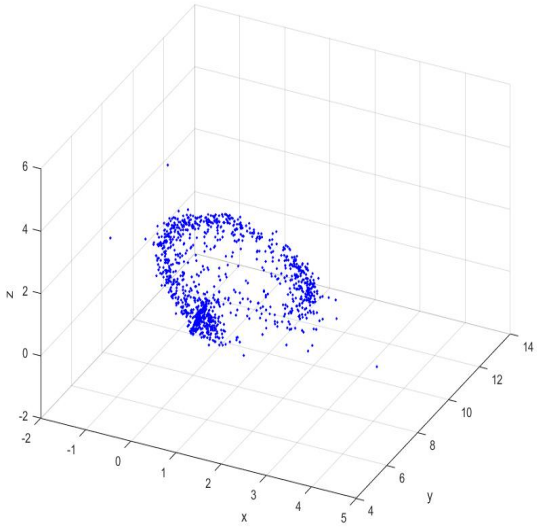
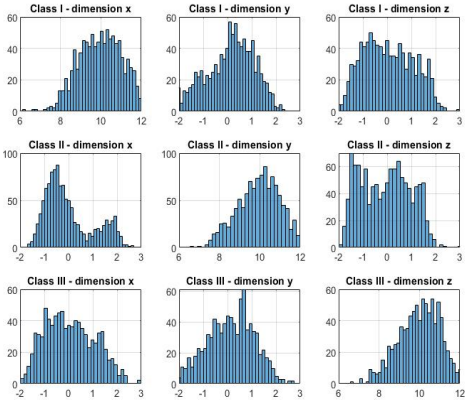


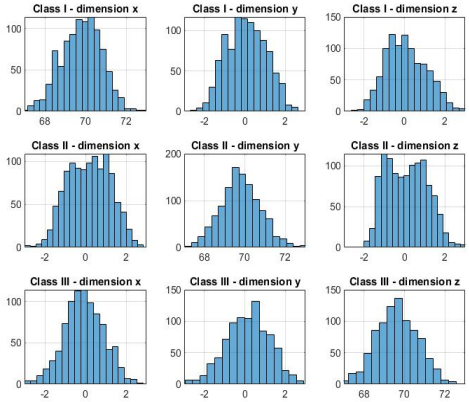
Figure 2.10: Close-up of class 1 from Figure 2.9a

2.2.2 Gaussianity

Histograms of the latent space scores obtained with the covariance loss for three classes can be seen in Figures 2.11a and 2.11b. For a high target mean value the results are more Gaussian compared to those obtained with simplified loss function for the same μ_T. Instead, for $\mu_T = 10$ it's possible to observe just a slight improvement from the previous case; some of the histograms appear more Gaussian, while others are still far from it, displaying two spikes in place of the Gaussian tails.



(a) Histograms of dimensions for μ_T=10



(b) Histograms of dimensions for μ_T=70

Figure 2.11: Histograms of dimensions corresponding to Figure 2.9

2.3 Kurtosis improvement

The results presented up to this Section demonstrate a certain degree of divergence from the ones obtained by Testa et al. [1] in the 2D case. It seems that not only the loss referenced in (2.3), which was an approximation, but also the one in (2.1) do not guarantee normal distributions along the dimensions.

In this regard we modified the loss functions with the addition of a term containing the kurtosis of the class mappings at the encoder's output, as follows.
With the definition of (2.5), the nominal loss function (2.1) becomes as in (2.6):

$$\mathcal{L}_{cov,\,x} = \frac{1}{x}\left[\log\frac{|\boldsymbol{\Sigma}_{Tx}|}{|\boldsymbol{\Sigma}_{Ox}|} - d + \mathrm{tr}(\boldsymbol{\Sigma}_{Tx}^{-1}\boldsymbol{\Sigma}_{Ox}) + (\boldsymbol{\mu}_{Tx} - \boldsymbol{\mu}_{Ox})^{\boldsymbol{T}}\boldsymbol{\Sigma}_{Tx}^{-1}(\boldsymbol{\mu}_{Tx} - \boldsymbol{\mu}_{Ox})\right] \tag{2.1}$$

$$\mathcal{K}_x = \frac{1}{d}\sum\left(\frac{\boldsymbol{x} - \boldsymbol{\mu}_{Ox}}{\boldsymbol{\sigma}_{Ox}}\right)^4 \tag{2.5}$$

$$L_{TOT} = \sum_{i=0}^{d} \mathcal{L}_{cov,\,i} + \mathcal{F}_k(\mathcal{K}_i - 3) \tag{2.6}$$

Equation (2.5) is the standard formula for the kurtosis of a distribution; moreover, the kurtosis of any univariate normal distribution is 3, which explains why in (2.6) the absolute minimum of $\mathcal{L}_x$ can only be reached when $\mathcal{K}_i = 3 \;\; \forall i$. The parameter $\mathcal{F}_k$ is a scaling factor used to tune the kurtosis term to the same order of magnitude of $\mathcal{L}_x$, and will be discussed further in following sections.
Similarly, the same procedure can be applied to the simplified loss:

$$\mathcal{L}_{simpl,\,x} = \frac{1}{x}\left[\log\frac{\sigma_{Tx}^{2d}}{\Pi_i\boldsymbol{\Sigma}_{Ox}^{(ii)}} - d + \frac{\sum_i^d \boldsymbol{\Sigma}_{Ox}^{ii}}{\sigma_{Tx}^2} + \frac{||\boldsymbol{\mu}_{Tx} - \boldsymbol{\mu}_{Ox}||^2}{\sigma_{Tx}^2}\right] \tag{2.3}$$

$$L_{TOT} = \sum_{i=0}^{d} \mathcal{L}_{simpl,\,i} + \mathcal{F}_k(\mathcal{K}_i - 3) \tag{2.7}$$

In synthesis we hoped that adding a kurtosis term to the existing losses could lead the network to map more regular distributions during the training phase. Experiments executed with these functions yield significantly better behaved latent spaces, as can be observed in the following discussion.

Experiment execution

For all results in Section 2.3 we executed the experiments with the following configuration:

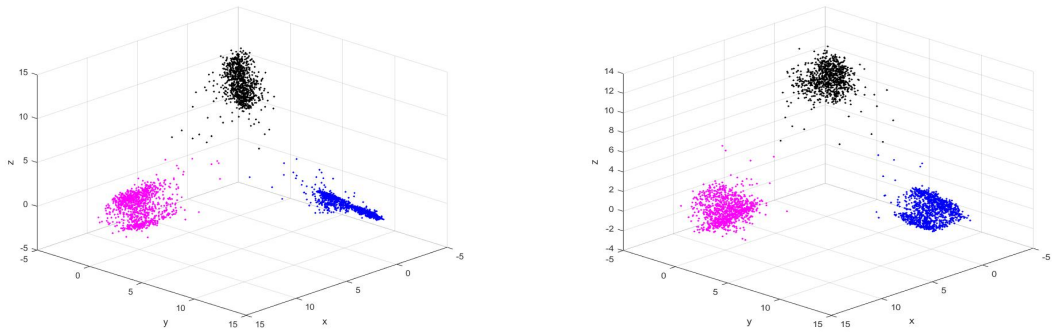
- Dataset: MNIST, classes [0] [1] [2].
- Encoder structure: ResNet-18.
- Optimizer: Adam optimizer and stochastic gradient descent.
- Iterations: 30000 batches.
- Batch size: 200.
- Learning rate: starts as $\lambda = 10^{-1}$, becomes $\lambda = 10^{-2}$ at $i = 1000$, $\lambda = 10^{-3}$ for $i = 5000$, $\lambda = 10^{-4}$ for $i = 10000$, $\lambda = 10^{-5}$ for $i = 20000$.
- Loss function: either (2.6) or (2.7), for the covariance loss and the simplified loss respectively.
- Latent space dimensionality: $d = 3$.
- Target mean: $\mu_T = 10 \vee \mu_T = 70$ or $\mu_T \in [5, 100]$. Specified for each result.
- Target variance: $\sigma_T = 1$.
- Kurtosis scaling factor: varies, specified for each result.
- For each class the encoder is supposed to produce an array of zeros containing a single μ_T value in the position identifying the class (e.g. for the second class the output should be [0,μ_T,0]).

2.3.1 Effects of kurtosis magnitude on latent spaces

The correct tuning of the scaling factor $\mathcal{F}_k$ is crucial for the correct outcome of training. The two terms $\mathcal{L}_x$ and $\mathcal{F}_k(\mathcal{K}_x - 3)$ must be around the same order of magnitude in order to minimize them simultaneously; it was observed that if one is consistently smaller than the other, it ends up being considered as just a random noise on top of the greater one, and the optimizer is unable to distinguish it from the small natural fluctuations of the loss function.

Figure 2.12 compares the results of loss function with and without kurtosis term with a small scaling factor; it can be evinced that the behaviors are similar to the ones

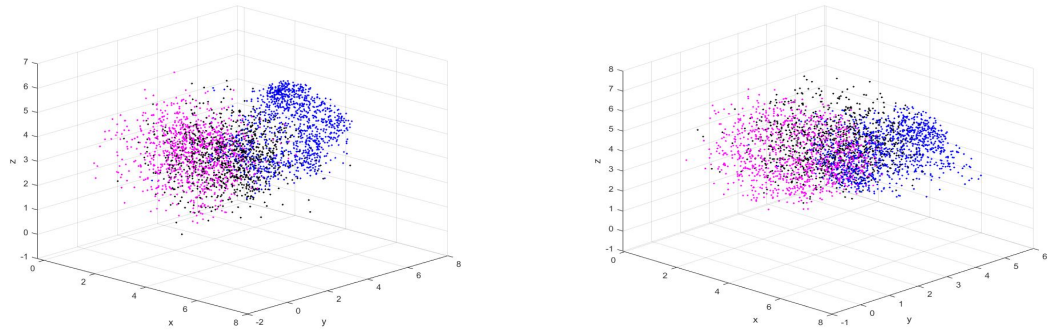
where the loss functions contained no kurtosis correction (refer to 2.1.4 and 2.2.1), since the effect of this term becomes too small to influence the distributions.



(a) Latent space simplified loss **(b)** Latent space for covariance loss

Figure 2.12: Latent spaces trained with small scaling factor ($\mathcal{F}_k = 0.05$, μ_T=10)

On the other hand, when the scaling factor is too large in comparison to the rest of the loss, there is a collapse of all classes into a single cluster, as shown in Figure 2.13, which leads to low classification accuracy.

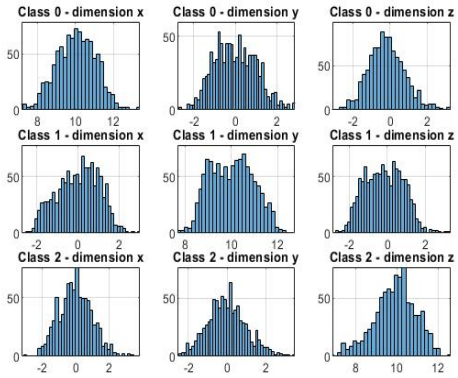


(a) Latent space simplified loss **(b)** Latent space covariance loss

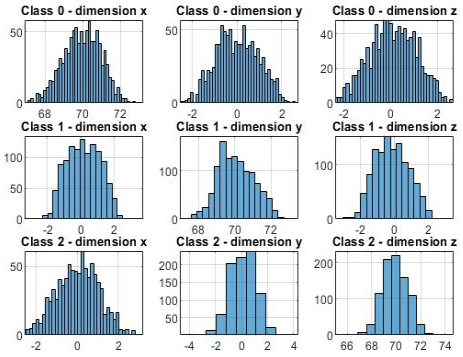
Figure 2.13: Latent spaces trained with large scaling factor ($\mathcal{F}_k = 3000$, μ_T=10)

Unsurprisingly, a full spectrum of intermediate states between the presented excesses exists and is not interesting enough to be reported.

Figures 2.14 and 2.15 show the results of training with a properly tuned scaling factor $\mathcal{F}_k$: these histograms have undergone a significant shape improvement, and appear more Gaussian than their unmodified counterparts (refer to Figures 2.5b, 2.6b, and 2.9). The most decisive upgrades are noticeable in the cases of low mean, which were critical with no kurtosis correction applied, as well as in the case of simplified loss function with high μ_T value.

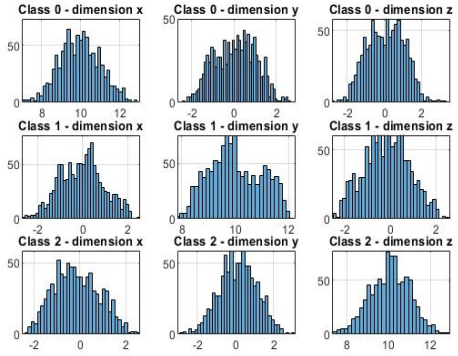


(a) μ_T=10

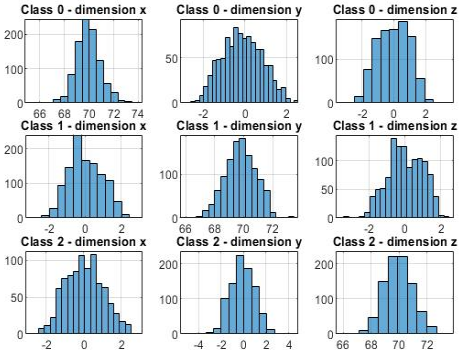


(b) μ_T=70

Figure 2.14: Dimensions for simplified loss and tuned scaling factor ($\mathcal{F}_k = 0.2$)



(a) μ_T=10



(b) μ_T=70

Figure 2.15: Dimensions for covariance loss and tuned scaling factor ($\mathcal{F}_k = 0.2$)

2.3.2 Kurtosis and skewness of dimensions

In an attempt to find a valid measure of Gaussianity, we evaluated the kurtosis and skewness of the histograms, and came across an interesting phenomenon, depicted in Figures 2.16 and 2.17. Apparently, the values of kurtosis and skewness (averaged across all classes and dimensions) suffer from a sizeable increase proportional to the target mean value μ_T, even when the histograms appear to be reasonably Gaussian; this is troublesome, because having used the kurtosis to enforce the Gaussianity, we would expect a kurtosis value close to 3 when the distributions approach Gaussianity. Eventually, we discovered that a single point mapped far from the target location increases enormously kurtosis values across some dimensions; this phenomenon is accentuated for greater target means because they allow for greater absolute error in the mapping, which causes greater kurtosis distortion.

We concluded that, even if the Gaussianity is enforced through the introduction of a kurtosis term, kurtosis itself is not an appropriate measure of Gaussianity (for our

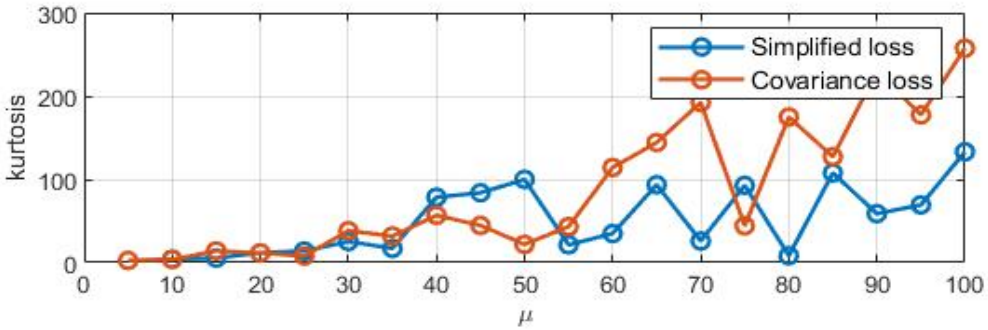


Figure 2.16: Kurtosis value for varying μ_T

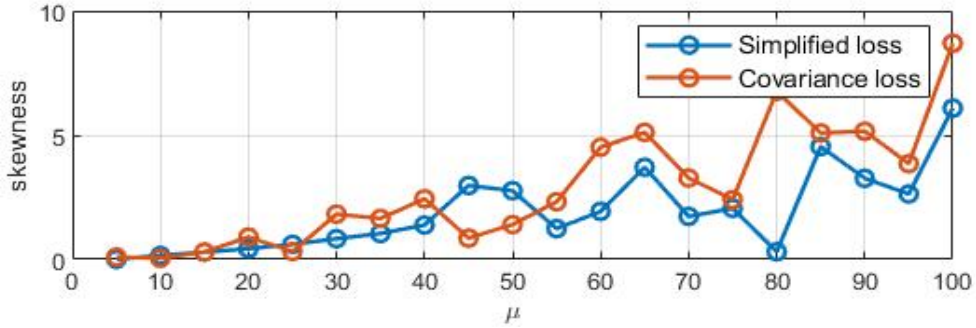


Figure 2.17: Skewness value for varying μ_T

case), and neither is the skewness; instead, we suggest to use a calculation of root mean square error between the histograms of the output scores and a standard normal curve with mean μ_T and variance σ_T.

2.4 Cross-Entropy loss function

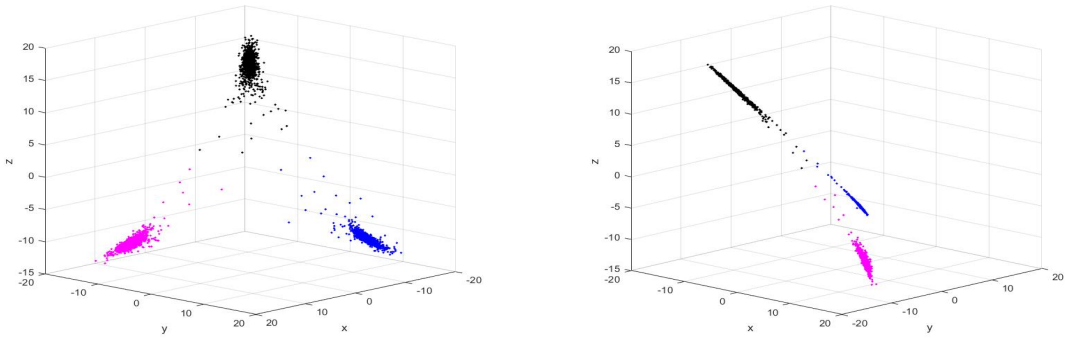
This section exhibits briefly the results of training the encoder with a cross-entropy loss function, since one of the main focuses of this whole thesis is to compare the RegNet loss with standard methods.

Experiment execution

- Dataset: MNIST, classes [0] [1] [2].
- Encoder structure: ResNet-18.
- Optimizer: Adam optimizer and stochastic gradient descent.
- Iterations: 30000 batches.
- Batch size: 200.
- Learning rate: starts as $\lambda = 10^{-1}$, becomes $\lambda = 10^{-2}$ at $i = 1000$, $\lambda = 10^{-3}$ for $i = 5000$, $\lambda = 10^{-4}$ for $i = 10000$, $\lambda = 10^{-5}$ for $i = 20000$.

- Loss function: the standard one provided by Tensorflow, (tf.nn.softmax_cross_entropy_with_logits( )).
- Latent space dimensionality: $d = 3$.
- Target mean: not relevant for this loss function.
- Target variance: not relevant for this loss function.

Figures 2.18 shows the latent distributions obtained with the cross-entropy method: it's notable the fact that they are not Gaussian, but rather placed on a plane in a triangular formation.



(a) Front view **(b)** Side view

Figure 2.18: Latent space for cross-entropy loss

2.5 Accuracy

All the speculations and details about alternative classification methods, Gaussianity, and latent space distributions, would be meaningless if the overall results couldn't match the state of the art accuracy, which is usually the main goal for any classification problem.

Therefore, in this section we'll examine the achieved accuracy levels for the four previously introduced loss functions:

1. Covariance loss
2. Simplified loss
3. Covariance loss with kurtosis
4. Simplified loss with kurtosis

In particular, we'll compare our results with the accuracy obtained for standard cross-entropy loss training sessions with the same parameters.

Although until now we only discussed three-dimensional cases, the accuracy results of this section are pertinent to the whole MNIST dataset, employing all ten classes; this choice is due to the fact the MNIST is inherently a classification problem on ten classes, and accuracy levels calculated on an arbitrarily chosen set of them are not relevant; also, no significant differences are present between the 3D and 10D cases, for what concerns accuracy.

Experiment execution

- Dataset: MNIST, all 10 classes.
- Encoder structure: ResNet-18.
- Optimizer: Adam optimizer and stochastic gradient descent.
- Iterations: 30000 batches.
- Batch size: 200.
- Learning rate: starts as $\lambda = 10^{-1}$, becomes $\lambda = 10^{-2}$ at $i = 1000$, $\lambda = 10^{-3}$ for $i = 5000$, $\lambda = 10^{-4}$ for $i = 10000$, $\lambda = 10^{-5}$ for $i = 20000$.
- Loss function: (2.2), (2.4), (2.6), (2.7) are used and compared.
- Latent space dimensionality: $d = 10$.
- Target mean: $\mu_T \in [5, 300]$.
- Target variance: $\sigma_T = 1$.
- Kurtosis scaling factor: $\mathcal{F}_k = 0.2$, unless otherwise specified.
- For each class the encoder is supposed to produce an array of zeros containing a single μ_T value in the position identifying the class.

2.5.1 Dependency of accuracy on target mean

The distance among distributions can affect the classification accuracy, as could be predicted from some of the previous discussions. In particular, when μ_T is excessively small the classes are too close to each other to allow a correct evaluation, while for high values of μ_T it's easier for points of the distribution to be loosely spread across the latent space.

At any rate, Figures 2.19 and 2.20 shows the accuracy for a range of target means for all four loss functions. For low μ_T values, usually for $\mu_T < 10$, all of them display a relatively low accuracy, while for greater means the performances settle around a steady-state value. The simplified loss function displays a low accuracy in its steady-state, both with and without the kurtosis term. This shows that perhaps for ten dimensions it stops being an accurate approximation of the covariance loss, which instead maintains levels comparable to the cross-entropy function.

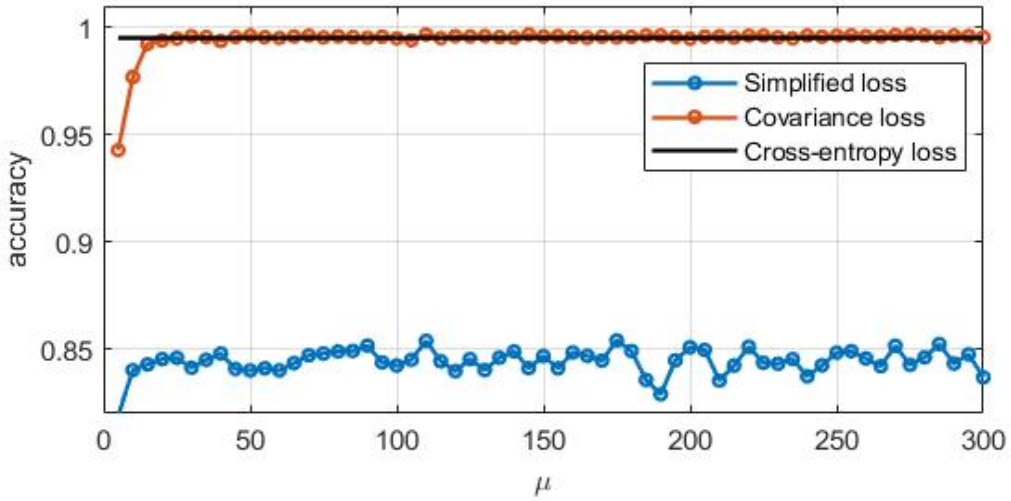


Figure 2.19: Accuracy levels for loss functions with no kurtosis correction - 10 classes

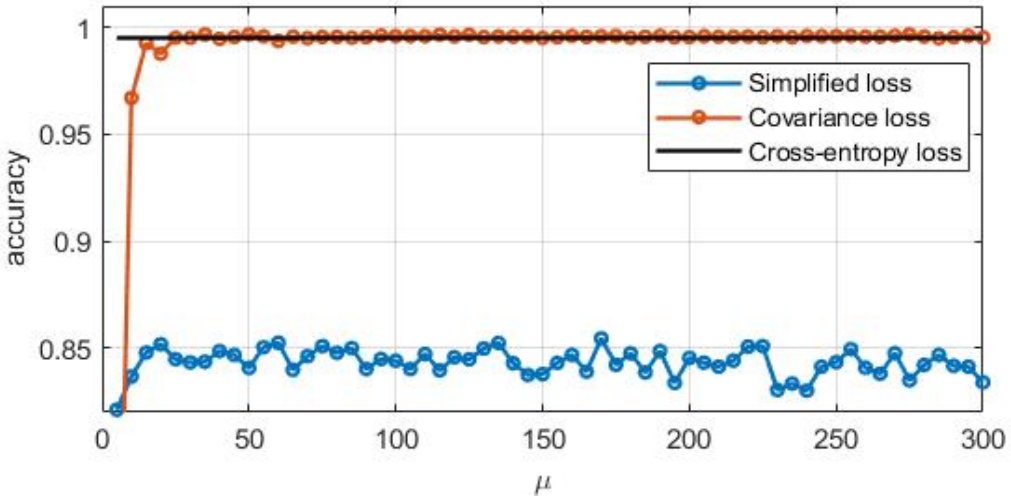


Figure 2.20: Accuracy levels for loss functions with kurtosis correction - 10 classes

2.5.2 Dependency of accuracy on scaling factor

Similarly to what's been analyzed in 2.5.1, also the parameter $\mathcal{F}_k$ can affect the classification accuracy, as was also mentioned in 2.3.1. Notably, a large kurtosis scaling factor causes all classes to collapse in the center of the latent space. An undersized scaling factor, instead, doesn't affect the accuracy levels, but rather the distributions' Gaussianity. The effects of scaling factor variation can be observed in Figure 2.21.

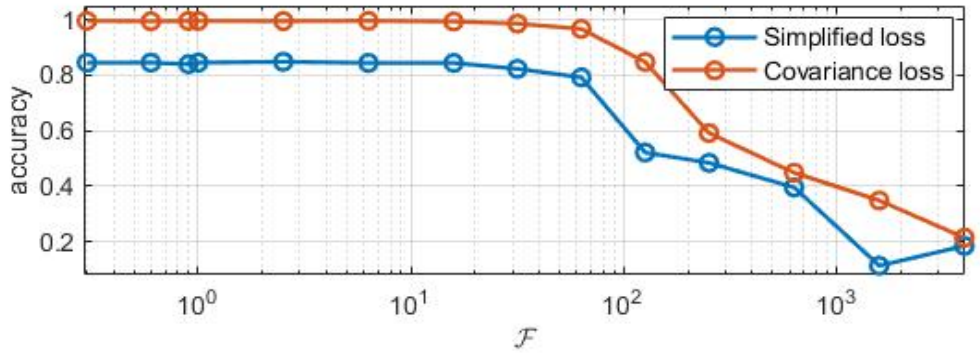


Figure 2.21: Accuracy for varying $\mathcal{F}_k$

2.6 Chapter conclusions

Section 2.5.1 made clear that, given an appropriate target mean, the ResNet system can achieve competitive results. In the next chapter these performances will be tested against the CIFAR dataset, which is considerably more complex than the MNIST, and against adversarial perturbations.

The simplified loss functions described in (2.3) and its kurtosis variation are to be discarded for the multi-dimensional implementation of ResNet. They were meant to be approximations of the real function, but the unpredictability of the latent distributions' shapes and the poor accuracy on ten classes training showed that this hypothesis was only valid in the 2D setting, and is now to be abandoned.

Moreover, the original covariance loss function containing no kurtosis correction described in (2.1), when compared to its two-dimensional analogous displayed a surprising behavior concerning the Gaussianity, and its usage could produce unforeseen results too. Therefore, the only loss function that will be employed in the next chapter for adversarial perturbations and the CIFAR dataset is the one containing both the covariance matrices and the kurtosis correction, as defined by (2.6).

At any rate, all of the originally proposed functions displayed unexpected behaviors in their results, but also during the training phase. They were designed to reach zero as an ideal minimum so that in realistic scenarios they would assume values at least close to it; this manifested correctly in the two-dimensional case, but it's no longer the case for the experiments we performed. The expansion to many dimensions revealed a lack of consideration for the numerical conditioning of the losses, which settled in their steady-state to values considerably greater than zero; in the 10D case their minimum values were several orders of magnitude greater than zero, when the loss was minimized and the training complete. Although this didn't impact the training process, it's an immediate indicator of the possible presence of other defects, such as the one that caused the unforeseen distribution shapes.

An example of this phenomenon, but there are surely others that we didn't examine

in the same detail, is the calculation of the covariance matrix's determinant $|\Sigma_{Ox}|$ in (2.1): ideally the covariance matrix should be reduced to a diagonal matrix, and the determinant should assume value 1, being product of the elements on the diagonal. Yet, in realistic cases said matrix would assume values not exactly equal to 1 on the diagonal, and not exactly equal to 0 elsewhere, causing the determinant to be a linear combination of all these terms. In a two-dimensional scenario, these effects are barely noticeable, being the covariance a 2 by 2 matrix, but with ten classes the determinant is calculated from a 10 by 10 matrix, and all non-idealities cause it to assume a value many times smaller than the intended one. Considering that the $|\Sigma_{Ox}|$ is a denominator and that the total loss is calculated as the sum of ten individual class losses, the overall steady-state loss function explodes to levels far above 1, often assuming values of several thousands even at the completion of training.

While some of the presented loss functions, especially (2.6), will prove effective against adversarial perturbation, they displayed several unpredicted behaviors, and so the authors advise that they should be re-formulated; in a corrected version there should be more focus on their numerical conditioning, and more attention to the effects of approximations, which are negligible for 2D scenarios but become gradually more impactful as the dimensionality increases.

Chapter 3

Adversarial perturbations

Chapter 2 examined the issues and similarities of ResNet expansion from one-vs-all classification to multi-class classification, with some resulting hindrances that lead us to modify the choice of loss function.

This chapter will focus on adversarial perturbations and the adversarial robustness of RegNet. The MNIST dataset is quite simple, and so we further analyze RegNet's robustness on a more challenging dataset like CIFAR-10. Furthermore, we'll also study the application of the RegNet loss function to a shake-shake encoder.

3.1 Fast Gradient Sign Method

For our experiments we perform the adversarial attacks with FGSM, or Fast Gradient Sign Method. It is an attack with the following characteristics:

- **One-shot**: each image is altered exactly once
- **Non-targeted**: the attack doesn't try to obtain the incorrect classification in favor of one specific class, instead focuses on having the lowest overall confidence on the correct class
- **White box**: adversary has access to the system parameters, and used them to generate the worst possible adversarial sample for each input

FGSM perturbations are generated according to the following equation:

$$x_{adv} = x + \varepsilon \operatorname{sign}(\nabla_x \mathcal{L}(x, y_{true})) \tag{3.1}$$

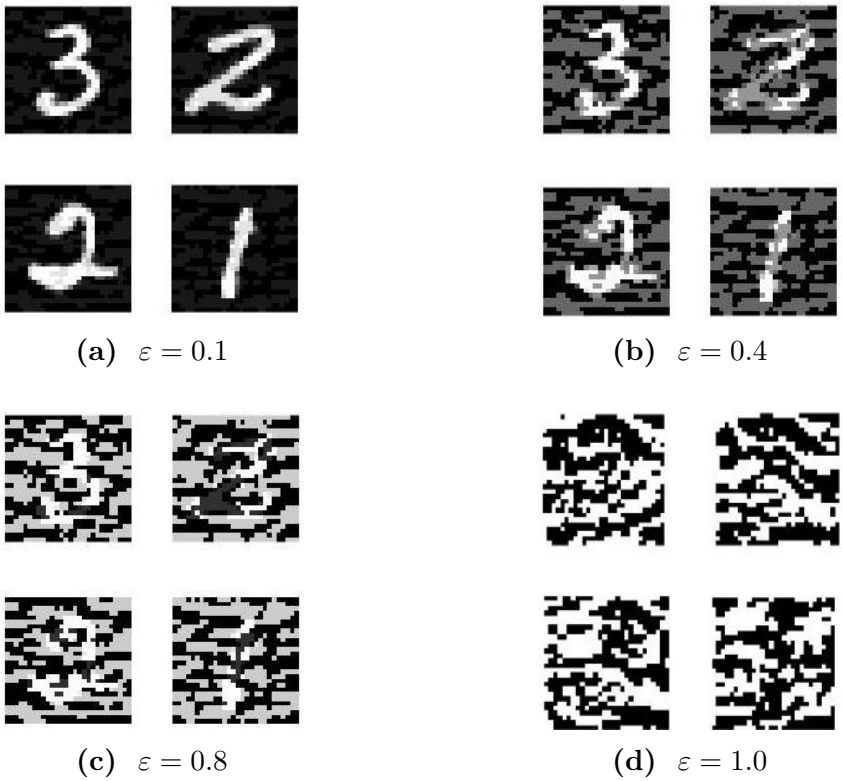
Where:

- x and x^{adv} are the clean and the perturbed inputs respectively.

- ε is the noise power.
- $\mathcal{L}(x, y_{true})$ is the loss function with respect to the true labels.
- sign() and ∇_x are the standard mathematical symbols for sign function and gradient, respectively.

When addressing the noise power, we will refer to it as a number between 0 and 1: clearly, this is to be interpreted as the ratio between the power itself and the maximum possible power of the input.

Example of perturbed MNIST images



(a) $\varepsilon = 0.1$ **(b)** $\varepsilon = 0.4$

(c) $\varepsilon = 0.8$ **(d)** $\varepsilon = 1.0$

Figure 3.1: MNIST images with different noise power

Figure 3.1 demonstrates the effects of adversarial perturbations on a sample of MNIST images. In FGSM attack model the pixels that get changed the most are the ones representing the most peculiar features of the digit according to the neural network. For $\varepsilon = 0.8$ the silhouettes can still be faintly distinguished, but with $\varepsilon = 1$ the noise power is equal to the maximum input power, and the images appear like random noise; in this state, any attempt of classification is completely random, even for humans.
Further on in our discussion we'll measure the robustness of a system by the slope that characterizes the performance's decline.

3.2 Effects of adversarial attacks on the latent space

In Chapter 2 the three-dimensional latent space was analyzed in great detail from several standpoints. In this section we analyze the effects of adversarial attacks on the latent space. We expect the attack to cause a lower detection accuracy, but the way this could be mirrored in the latent dimensions may not be intuitive and is worth to be examined.

Experiment execution

For all results in Section 3.2 we executed the experiments with the following configuration:

- Dataset: MNIST, various triplets of classes, specified for each result.
- Encoder structure: ResNet-18.
- Optimizer: Adam optimizer and stochastic gradient descent.
- Iterations: 30000 batches.
- Batch size: 200.
- Learning rate: starts as $\lambda = 10^{-1}$, becomes $\lambda = 10^{-2}$ at $i = 1000$, $\lambda = 10^{-3}$ for $i = 5000$, $\lambda = 10^{-4}$ for $i = 10000$, $\lambda = 10^{-5}$ for $i = 20000$.
- Loss function: the one identified by (2.6).
- Latent space dimensionality: $d = 3$.
- Target mean: $\mu_T = 70$.
- Target variance: $\sigma_T = 1$.
- Kurtosis scaling factor: $\mathcal{F}_k = 0.2$.

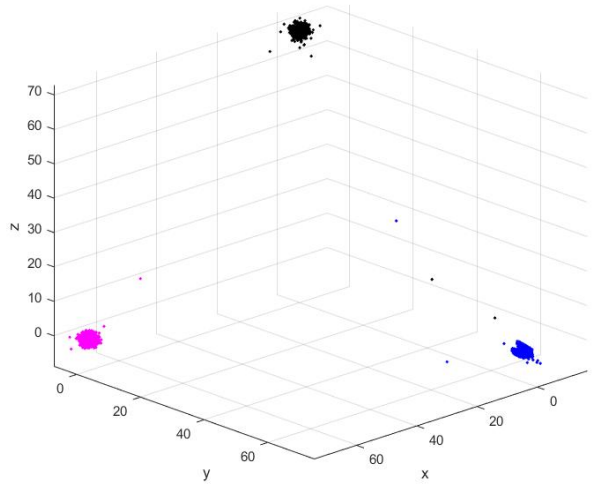


Figure 3.2: Latent space for $\varepsilon = 0.0$

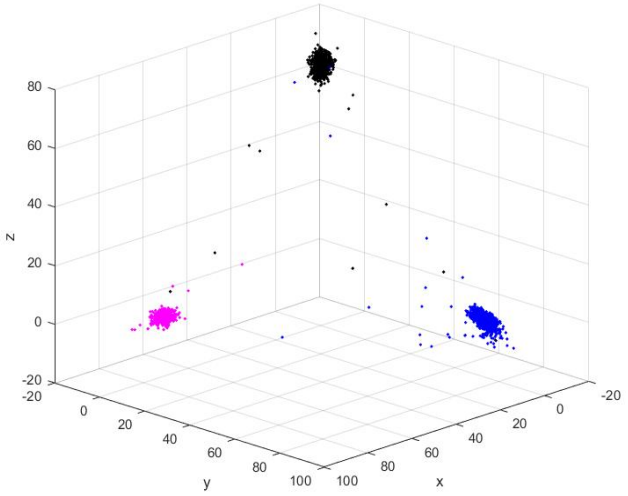
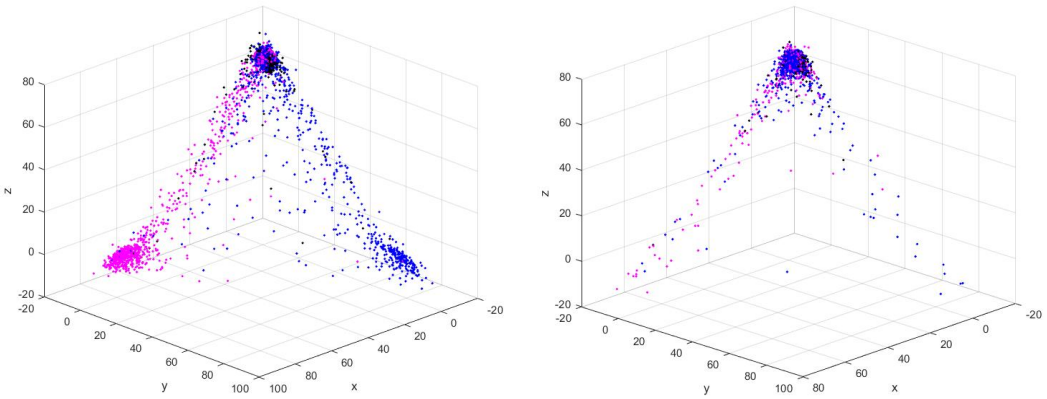


Figure 3.3: Latent space for $\varepsilon = 0.1$

Figures 3.2 and 3.3 show 10% of the maximum noise power already forcing some inputs to be mapped far from their target means. Figure 3.4 contains the latent mappings for greater noise powers, and it's interesting to observe that all the classes are gradually mapped on the same target mean. In other words, in this case the adversarial attack causes the encoder to identify all inputs as the same MNIST digit.



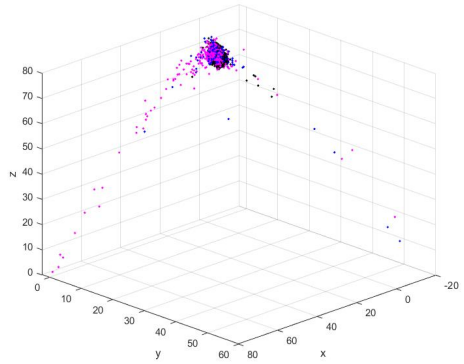
(a) $\varepsilon = 0.4$ **(b)** $\varepsilon = 0.8$

Figure 3.4: Latent spaces with higher noise power

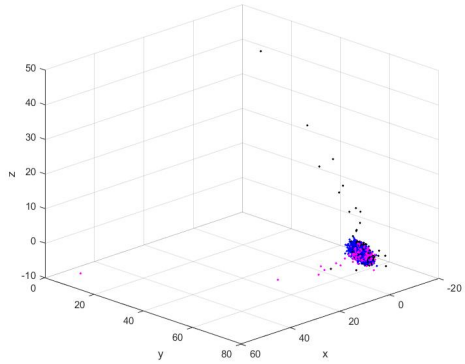
3.2.1 In depth examination of perturbed latent space

Figures 3.2, 3.3, and 3.4 show the effect of an adversarial attack on three MNIST classes [0,1,2], and as a result they are all mapped as the 2 digit. An interesting aspect of this phenomenon is whether they are always mapped to the same class, if it is random, or if there is some kind of bias.

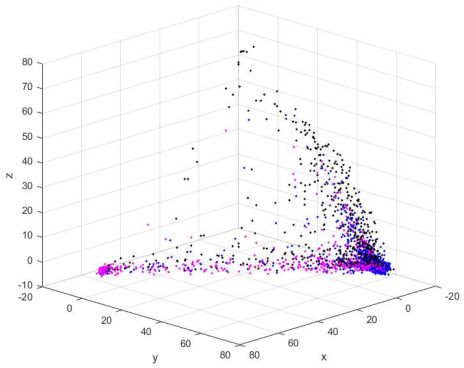
Sets of repeated experiments revealed that, every time the system is trained with the same triplet of labels and then perturbed, the mappings collapse always on the same digit; for example, when training with the classes [0,1,2], the attacks always cause the inputs to be mapped as 2. Such results suggest that the position where the mappings collapse is determined by the features and complexity of the input digits.
However, when trained with different triplets, the classes collapse different digits (Figure 3.5); for example, adversarial attacks on the classes [0,1,2] causes them to always collapse on 2, while attacks on [2,3,4] cause them to collapse on 4, no longer on 2. This result is significant because it shows that there is no bias towards any class in the implementation of the system.



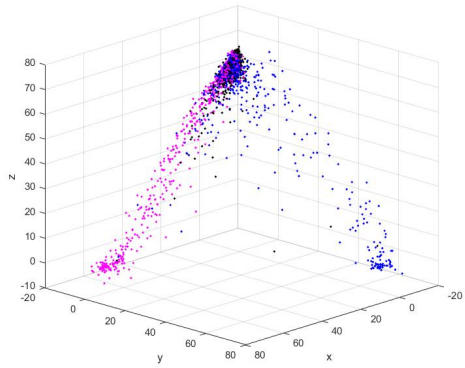
(a) Classes [2,3,4], converges to 4



(b) Classes [4,5,6], converges to 5



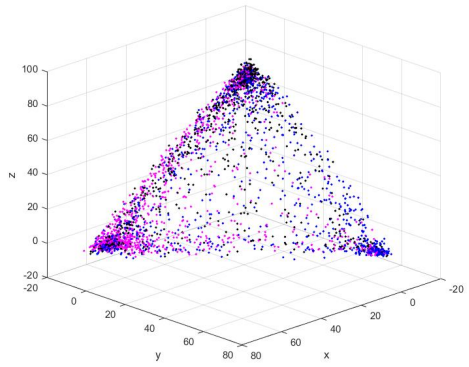
(c) Classes [5,6,7], converges to 6



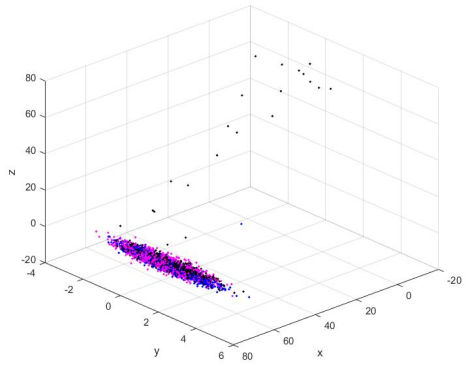
(d) Classes [6,7,8], converges to 8

Figure 3.5: Latent spaces with $\varepsilon = 0.8$ for different class sets

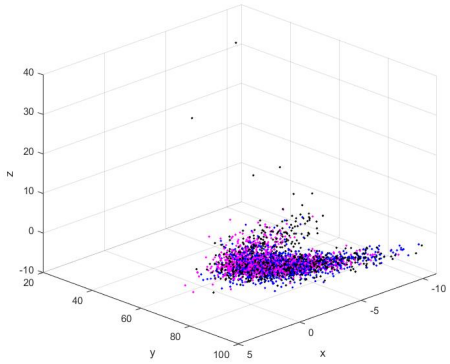
Conversely, some triplets of classes do not converge on a particular one, but instead respond to the adversarial attack as in Figure 3.6a: in some cases the classes become mapped in a uniform distribution across the whole latent space (Figure 3.6a), in others they converge in unconventionally shaped clusters, as in Figures 3.6b and 3.6c.



(a) Classes [1,2,3], doesn't converge



(b) Classes [3,4,5], atypical shape



(c) Classes [7,8,9], atypical shape

Figure 3.6: Latent spaces with peculiar features, $\varepsilon = 0.8$

Such a variety in the effects of adversarial perturbations may be due to the non-targeting nature of FGSM, and it would be interesting to examine if a targeting attack always obtains the same type of latent space. If so, knowing the latent space, it could be possible to devise adversarial defenses specific for that attack type. At any rate, none of such possibilities has been explored in thesis.

3.3 RegNet's robustness

As the final and most important section of this thesis, we'll compare the robustness of our RegNet loss function and a classic cross-entropy one. As explained before, a discussion concerning accuracy requires the experiments to be performed on the whole array of labels, not just a portion as was the case when examining the behavior of

latent spaces. At any rate, the tests we performed proved the results to be equivalent when considering three or ten dimensions.

We performed analyses on two datasets, the MNIST and the CIFAR-10, whose traits we'll briefly introduce in Section 3.3.2. This double examination is necessary to ensure that the obtained results are not limited by the morphology or simpleness of a specific database.

We performed yet another test by replacing the ResNet encoder with a shake-shake one, to verify that the merits of the results are traceable to the loss function and not to the specific encoder employed.

3.3.1 RegNet robustness on MNIST

Experiment execution

- Dataset: MNIST, all 10 classes.
- Encoder structure: ResNet-18.
- Optimizer: Adam optimizer and stochastic gradient descent.
- Iterations: 60000 batches.
- Batch size: 200.
- Learning rate: starts as $\lambda = 10^{-1}$, becomes $\lambda = 10^{-2}$ at $i = 1000$, $\lambda = 10^{-3}$ for $i = 5000$, $\lambda = 10^{-4}$ for $i = 10000$, $\lambda = 10^{-5}$ for $i = 30000$.
- Loss function: the one identified by (2.6).
- Latent space dimensionality: $d = 10$.
- Target mean: varies.
- Target variance: $\sigma_T = 1$.
- Kurtosis scaling factor: $\mathcal{F}_k = 0.2$.
- For each class the encoder is supposed to produce an array of zeros containing a single μ_T value in the position identifying the class.

Figure 3.7 displays the accuracy performance for all $\varepsilon \in [0, 1]$, for different target means μ_T. It can be observed that when noise is not present, the cross-entropy and the RegNet model have comparable accuracy. However, as noise power increases, our system proves consistently better than its cross-entropy counterpart, independently on the chosen μ_T. The only exception happens for exceedingly low target mean values, accordingly with the accuracy results produced in the previous chapter.

The means reported in the plot are chosen arbitrarily, an array that we deemed well representative of the range that different means could achieve (more on this in 3.4.1 and 3.4.2). At any rate, this type of performance is exactly what we expected and what we wanted to verify when we began this whole study.

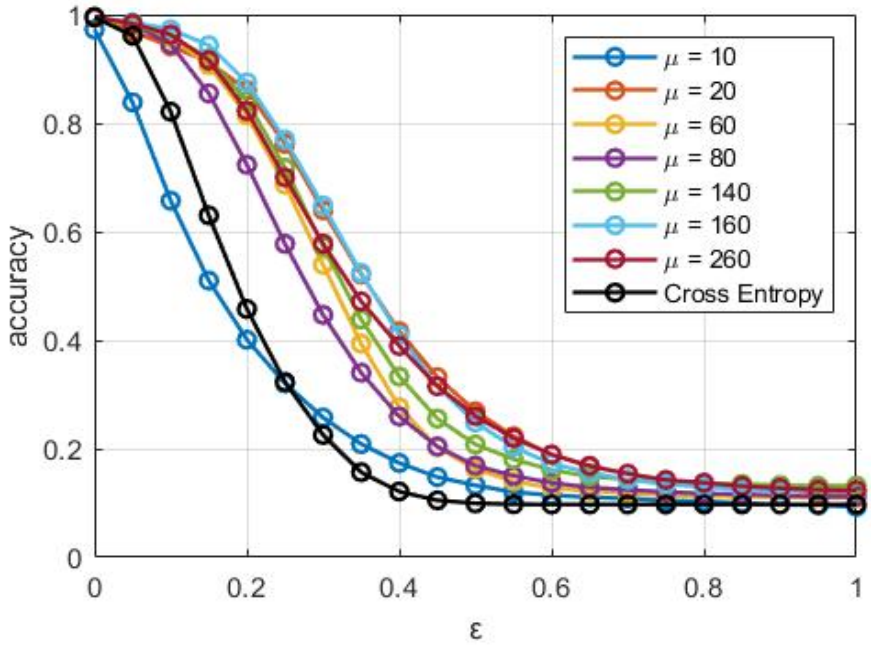


Figure 3.7: Accuracy for all $\varepsilon \in [0, 1]$ and some selected μ_T

3.3.2 CIFAR-10 structure

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. It is divided into training and test images, with a count of 50000 and 10000 respectively.

The ten labels in the dataset, with their respective name are:

0. airplane
1. automobile
2. bird
3. cat
4. deer
5. dog
6. frog
7. horse
8. ship
9. truck

The classes are completely mutually exclusive. There is no overlap between ambiguous classes like automobiles and trucks: for example, "automobile" includes sedans, SUVs, and similar, while "truck" includes only big trucks.

A sample of ten images per class is provided in Figure 3.8, in real size.

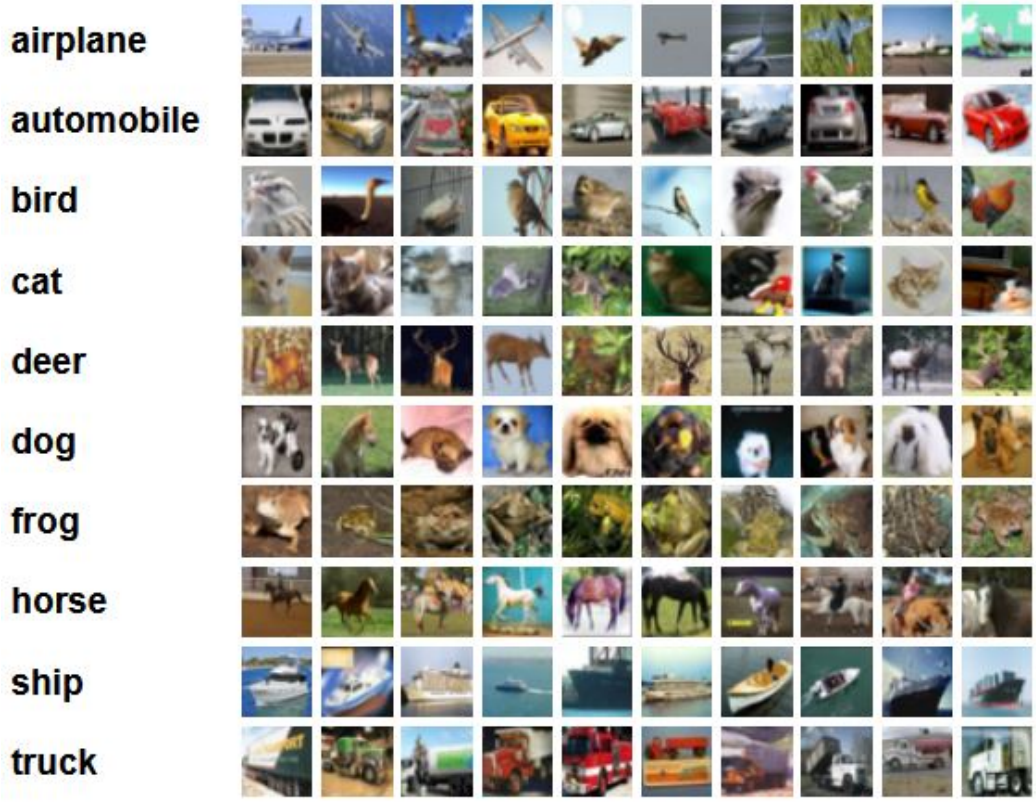


Figure 3.8: An example of CIFAR-10 images

3.3.3 CIFAR-10 performance

Experiment execution

- Dataset: MNIST, all 10 classes.
- Encoder structure: ResNet-18.
- Optimizer: Adam optimizer and stochastic gradient descent.
- Iterations: 60000 batches.
- Batch size: 200.
- Learning rate: starts as $\lambda = 10^{-1}$, becomes $\lambda = 10^{-2}$ at $i = 1000$, $\lambda = 10^{-3}$ for $i = 5000$, $\lambda = 10^{-4}$ for $i = 10000$, $\lambda = 10^{-5}$ for $i = 30000$.
- Loss function: the one identified by (2.6).
- Latent space dimensionality: $d = 10$.
- Target mean: varies.
- Target variance: $\sigma_T = 1$.
- Kurtosis scaling factor: $\mathcal{F}_k = 0.2$.
- For each class the encoder is supposed to produce an array of zeros containing a single μ_T value in the position identifying the class.

In terms of accuracy and robustness observed on the CIFAR-10 dataset, the results are

similar to the MNIST: the RegNet loss is more robust than its cross-entropy counterpart for any possible noise power.

The results of Figure 3.9, however, call for some necessary comments. First, it's easy to detect that even the curves with the best performances have a shamefully low accuracy; with $\varepsilon = 0$ the best accuracy reached in the chart is around 86%. This is explained with the fact that the residual layer encoder employed for these experiments, and originally implemented for the two-dimensional version of ResNet, was never meant for a complex problem like a ten-dimensional classification of the CIFAR-10 dataset. In the next sections (3.3.4 and 3.3.5) we'll address this issue, but at any rate, the focus of this discussion is the superior robustness of RegNet's loss, independently on the employed encoder.

Moreover, as a continuation of a trend already observed, exceedingly low target mean values produce unforeseen results. The network trained with $\mu_T = 10$, manifests particularly poor accuracy in unperturbed conditions ($\varepsilon = 0$), just like previous experiments of Chapter 2 showed that a relatively high μ was necessary to obtain competitive accuracy (Section 2.5.1 and Figure 2.20); however it's interesting to notice that for all the range $\varepsilon \in [0.1, 1]$ the curve for $\mu_T = 10$ proves to actually be the more robust.

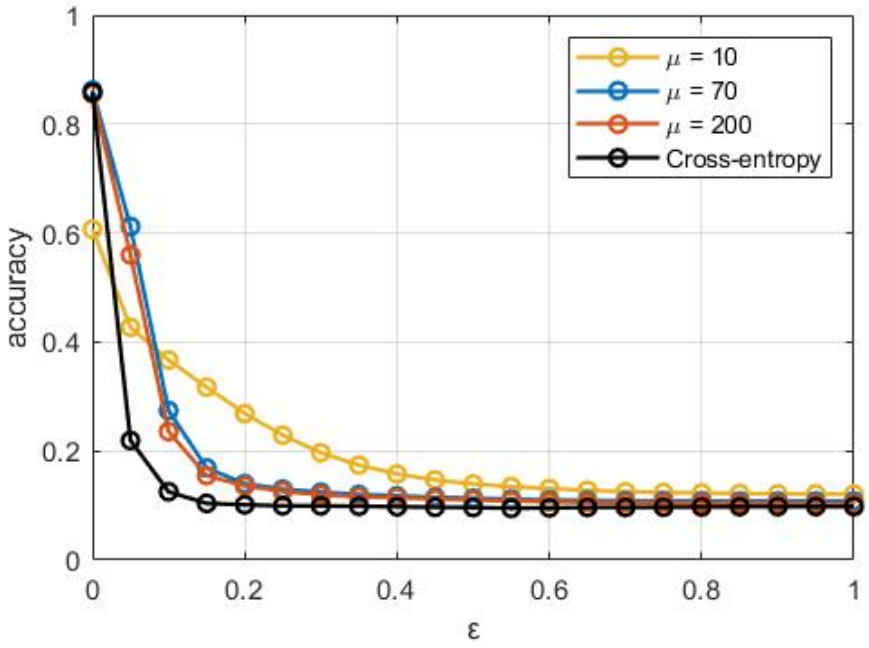


Figure 3.9: Accuracy for all $\varepsilon \in [0, 1]$ for some selected μ_T

3.3.4 Effects of residual layer depth on robustness

The previous experiment on the CIFAR-10 dataset resulted in a low overall accuracy; for $\varepsilon = 0$, the maximum was around 86%. We believed that a deeper residual network like ResNet-34 or ResNet-50 could help raise the accuracy to levels comparable with the MNIST experiments.

Experiment execution

- Dataset: MNIST, all 10 classes.
- Encoder structure: varies. The details of each residual layer for versions of ResNet other than ResNet-18 are reported in Figure 3.10, as described in [8].
- Optimizer: Adam optimizer and stochastic gradient descent.
- Iterations: 60000 batches.
- Batch size: 200.
- Learning rate: starts as $\lambda = 10^{-1}$, becomes $\lambda = 10^{-2}$ at $i = 1000$, $\lambda = 10^{-3}$ for $i = 5000$, $\lambda = 10^{-4}$ for $i = 10000$, $\lambda = 10^{-5}$ for $i = 30000$.
- Loss function: the one identified by (2.6).
- Latent space dimensionality: $d = 10$.
- Target mean: $\mu_T = 70$.
- Target variance: $\sigma_T = 1$.
- Kurtosis scaling factor: $\mathcal{F}_k = 0.2$.
- For each class the encoder is supposed to produce an array of zeros containing a single μ_T value in the position identifying the class.

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

Figure 3.10: ResNet residual layer details.

Figure 3.11 reports the results of tests with several depths of ResNet; each curve is averaged over six simulations, in order to have a definite estimate of performances.

It can be observed that the curves overlap almost entirely, implying that the residual layer depth doesn't affect the robustness at all.

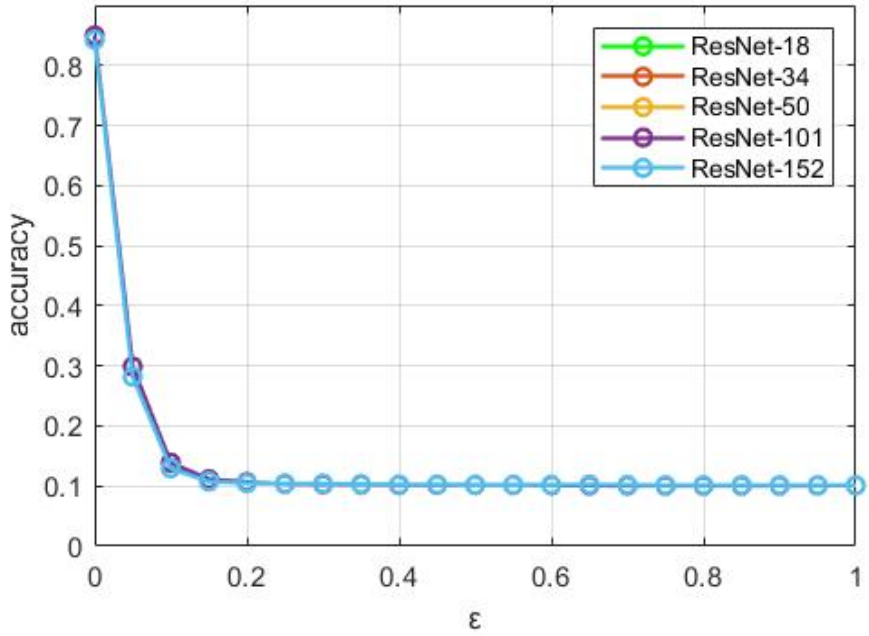


Figure 3.11: Accuracy for all $\varepsilon \in [0, 1]$ for different depths of ResNet. They all display the same performances, and the curves overlap.

3.3.5 Shake-Shake encoder

Shake-shake regularization, first introduced in *Shake-Shake regularization* [10] is a technique to improve the performances of deep residual neural networks.

When confronted with a dataset too small or too complex, residual nets will either not converge to a univocal solution or, if trained for a sufficient number of epochs, overfit. This is, in fact, what happens to ResNet: the CIFAR-10 dataset causes the encoder to classify the inputs with high error rate, but if training would protract for a great amount of time the network would eventually reach high accuracy on all training samples, maintaining a low accuracy on the test samples.

The idea of shake-shake regularization is to introduce a sort of data augmentation inside the network, between its layers. With this modification the network is hopefully able to train for a greater number of epochs without overfitting, therefore reducing the classification error.

As described by Gastaldi [10], shake-shake regularization is a way of "blending" two parallel tensors according to the following equation:

$$x_{i+1} = x_i + \alpha_i \mathcal{F}_k(x_i, \mathcal{W}_i^{(1)}) + (\alpha_i - 1)\mathcal{F}_k(x_i, \mathcal{W}_i^{(2)}) \tag{3.2}$$

Where:

- x_i represents the tensor of inputs in residual block i
- α_i is a random variable with with uniform distribution between 0 and 1
- $\mathcal{F}_k$ denotes the residual function
- x_{i+1} is the tensor of outputs from residual block i
- $\mathcal{W}_i^{(2)}$ and $\mathcal{W}_i^{(2)}$ are the sets of weights associated with residual blocks 1 and 2 respectively

The above formula (3.2) is similar to the basic equation of Residual Layer networks, the only exceptions being the presence of α_i and $(1 - \alpha_i)$. It is in fact the random variable α_i that introduces a degree of randomness in the otherwise deterministic generation of the residual layers, de facto adding the equivalent of random noise to the network's internal representation of the inputs, applying data augmentation on them, and reducing the probabilities of overfitting.

Experiment execution

- Dataset: MNIST, all 10 classes.
- Encoder structure: shake-shake-96, with depth 26 and widen factor 2.
- Optimizer: Momentum optimizer and stochastic gradient descent.
- Iterations: 1800 epochs.
- Batch size: 200.
- Loss function: the one identified by (2.6).
- Latent space dimensionality: $d = 10$.
- Target mean: $\mu_T = 70$.
- Target variance: $\sigma_T = 1$.
- Kurtosis scaling factor: $\mathcal{F}_k = 0.2$.
- For each class the encoder is supposed to produce an array of zeros containing a single μ_T value in the position identifying the class.

Shake-Shake encoder's performances

Figure 3.12 shows the robustness results of training with the shake-shake encoder. It can be observed that, when compared with Figures 3.7 and 3.9, the curves representing RegNet's loss exhibit performances similar to the cross-entropy loss. This would suggest that, in the previous experiments, the increased robustness was due to the ResNet encoder and not to the RegNet loss function, or at least that the effectiveness of RegNet is limited to cases where an appropriate encoder can be employed.

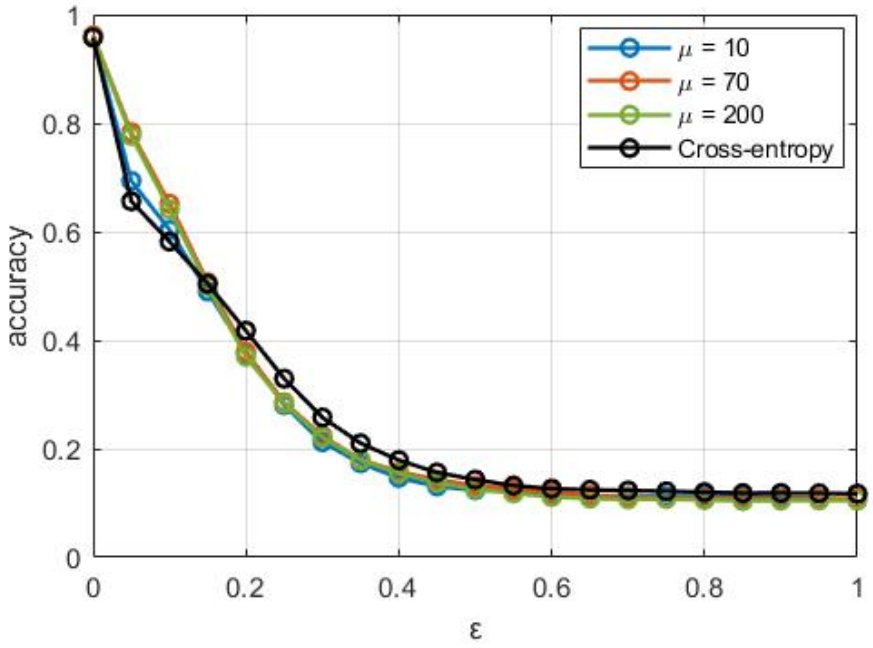


Figure 3.12: Accuracy for all ε between 0 and 1 for some selected μ_T

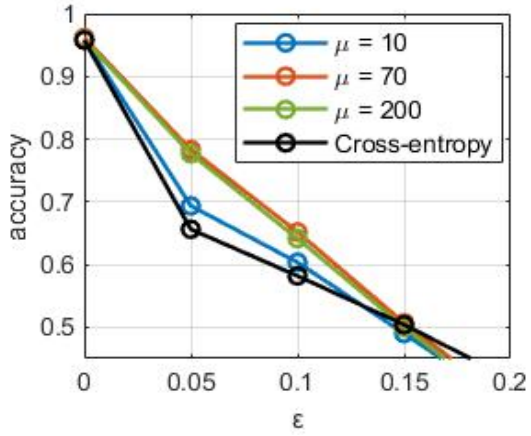


Figure 3.13: Close-up of Figure 3.12. The section where $0 \leq \varepsilon \leq 0.2$ is the most relevant because the adversarial attack can impact the system's accuracy while being undetectable by human observers.

However, the most significant section of the plot is the one where ε is small. A low noise power can disrupt the correct functioning of image processing systems without being noticed by the human eye. In the plot, when $\varepsilon = 0.5$, and the accuracy is around 15% for all systems, the fact that RegNet underperforms the cross-entropy loss by 1% is insignificant, because both their accuracies are too low. Regarding this, the zoom reported in Figure 3.13 shows that, for the most relevant section of the graph, for $\varepsilon \in [0, 0.1]$, RegNet performs moderately better than its cross-entropy counterpart.

3.4 Analysis of robustness variance

In most of the experiments of Section 3.4.1, the curves displayed ranging levels of robustness for RegNet, leaving a certain degree of variance in any possible conclusion. RegNet performs better than its cross-entropy counterpart, but it's not clear by which amount. In this section we'll analyze some possibilities to narrow down the results.

3.4.1 Dependency of accuracy on μ_T

In previous sections, especially 3.3.1, experiments with different μ_T produced different robustness performances, suggesting that a correlation between the two might exist. However, as it can be evinced from the following experiment (Figure 3.14), the variance in the plots is not due to a connection between robustness and target mean value.

Experiment execution

- Dataset: MNIST, all 10 classes.
- Encoder structure: ResNet-18.
- Optimizer: Adam optimizer and stochastic gradient descent.
- Iterations: 60000 batches.
- Batch size: 200.
- Learning rate: starts as $\lambda = 10^{-1}$, becomes $\lambda = 10^{-2}$ at $i = 1000$, $\lambda = 10^{-3}$ for $i = 5000$, $\lambda = 10^{-4}$ for $i = 10000$, $\lambda = 10^{-5}$ for $i = 30000$.
- Loss function: the one identified by (2.6).
- Latent space dimensionality: $d = 10$.
- Target mean: varies.
- Target variance: $\sigma_T = 1$.
- Kurtosis scaling factor: $\mathcal{F}_k = 0.2$.
- For each class the encoder is supposed to produce an array of zeros containing a single μ_T value in the position identifying the class.

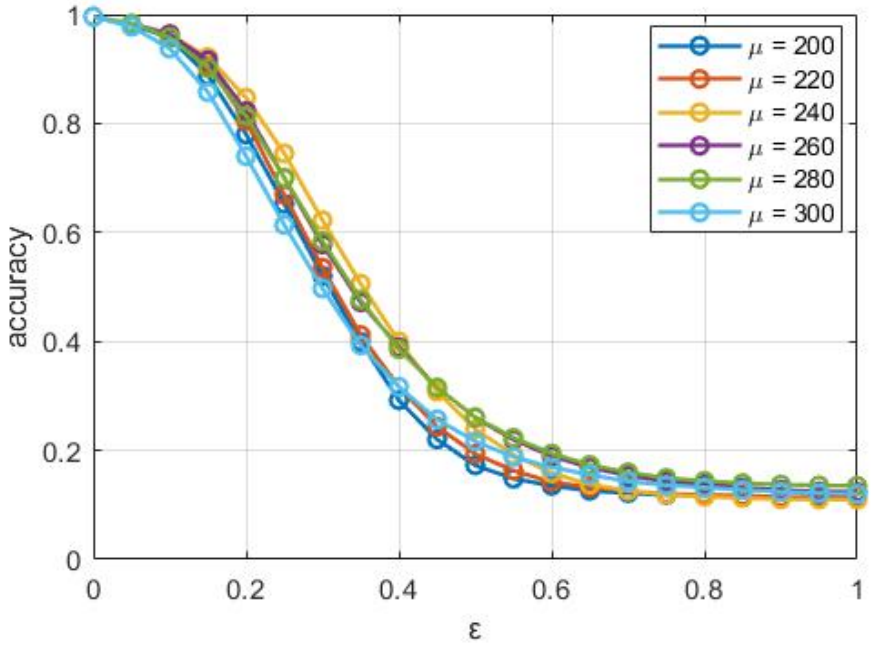


Figure 3.14: Accuracy for $\varepsilon \in [0, 1]$ and $\mu_T \in [200, 300]$. There is no trend connecting robustness and μ_T.

3.4.2 Variance introduced by training

Section 3.4.1 disproved the possibility that the variance of results be caused by μ_T. Therefore, in this section we explore the possibility that the training process introduces some randomness in the robustness of the system.

Experiment execution

- Dataset: MNIST, all 10 classes.
- Encoder structure: ResNet-18.
- Optimizer: Adam optimizer and stochastic gradient descent.
- Iterations: 60000 batches.
- Batch size: 200.
- Learning rate: starts as $\lambda = 10^{-1}$, becomes $\lambda = 10^{-2}$ at $i = 1000$, $\lambda = 10^{-3}$ for $i = 5000$, $\lambda = 10^{-4}$ for $i = 10000$, $\lambda = 10^{-5}$ for $i = 30000$.
- Loss function: the one identified by (2.6).
- Latent space dimensionality: $d = 10$.
- Target mean: $\mu_T = 70$.
- Target variance: $\sigma_T = 1$.
- Kurtosis scaling factor: $\mathcal{F}_k = 0.2$.
- For each class the encoder is supposed to produce an array of zeros containing a single μ_T value in the position identifying the class.

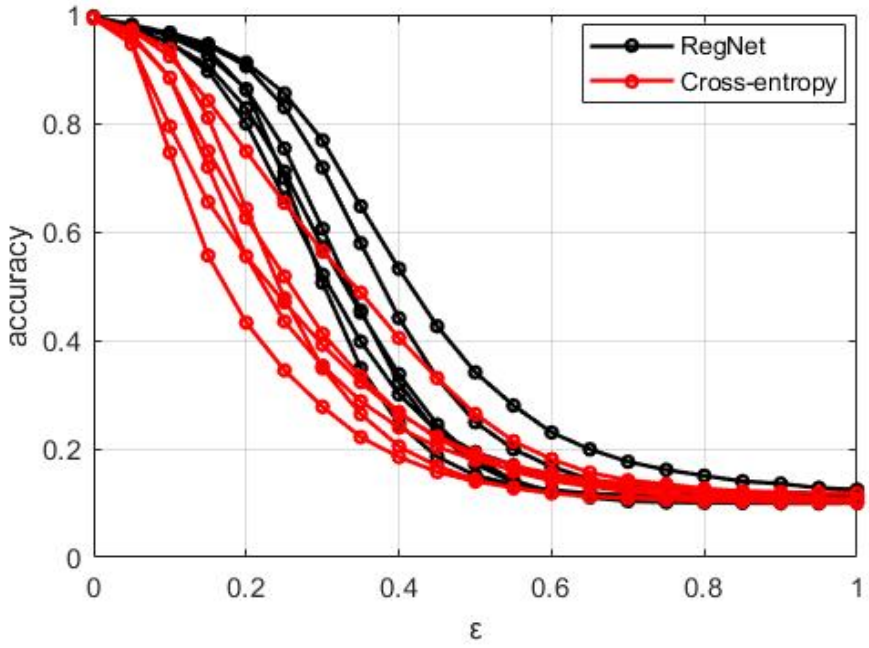


Figure 3.15: Accuracy for $\varepsilon \in [0, 1]$. Six curves for RegNet with $\mu_T = 70$, and six for cross-entropy loss.

Figure 3.15 shows the robustness of several training sessions with the same parameters. It's clear that the randomness of both, RegNet and the system with cross-entropy loss, is affected by the random variables of any training session, such as batch selection and optimizer. At any rate, this plot displays the ranges of robustness achievable by each system, and it's evident that, although the best cross-entropy performer and the worst RegNet performers overlap, RegNet has overall superior robustness.

3.4.3 Robustness for data augmented MNIST

In order to reduce the randomness in robustness performances introduced by the training process, we tried to perform the adversarial perturbations after training RegNet with a MNIST dataset modified with data augmentation.

It may be possible for the randomness to be caused by a lack of training, in which case the system would produce curves far apart from each other, as it happens, instead of having them converge towards univocal accuracy values. Data augmentation allows us to train the system for a greater number of epochs without the risk of overfitting, and would theoretically reduce the variance of the curves.

Experiment execution

- Dataset: MNIST, all 10 classes.
- Encoder structure: ResNet-18.

- Optimizer: Adam optimizer and stochastic gradient descent.
- Iterations: 60000 batches.
- Batch size: 200.
- Learning rate: starts as $\lambda = 10^{-1}$, becomes $\lambda = 10^{-2}$ at $i = 1000$, $\lambda = 10^{-3}$ for $i = 5000$, $\lambda = 10^{-4}$ for $i = 10000$, $\lambda = 10^{-5}$ for $i = 30000$.
- Loss function: the one identified by (2.6).
- Latent space dimensionality: $d = 10$.
- Target mean: $\mu_T = 70$.
- Target variance: $\sigma_T = 1$.
- Kurtosis scaling factor: $\mathcal{F}_k = 0.2$.
- Data augmentation: any combination of random rotations (maximum 20°), random shears (maximum 20% of original size), random shifts (maximum 20% of original size), random zooms (maximum zoom to 90% of original size).
- For each class the encoder is supposed to produce an array of zeros containing a single μ_T value in the position identifying the class.

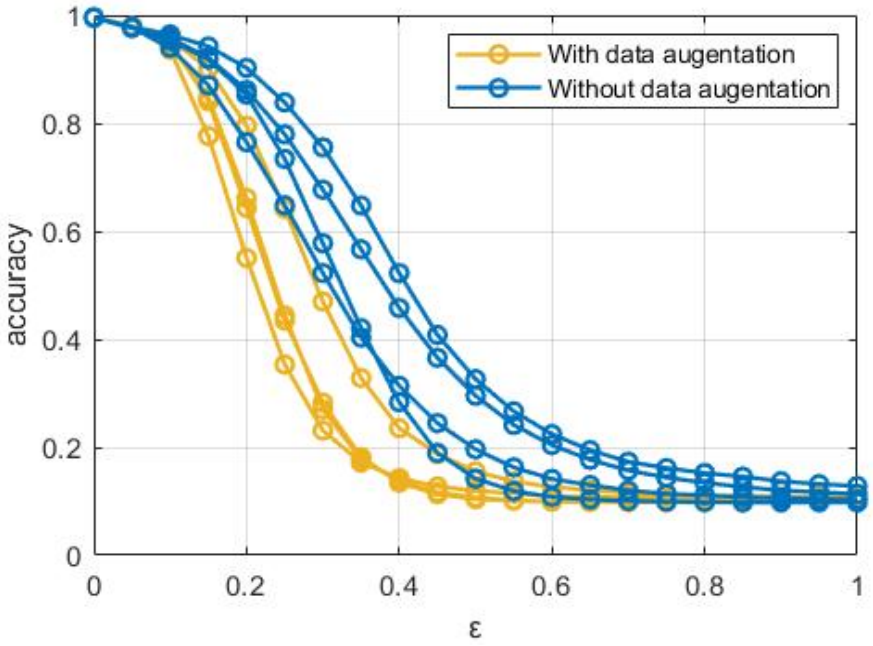


Figure 3.16: Accuracy for $\varepsilon \in [0, 1]$ for training sessions with and without data augmentation on MNIST dataset.

The results of Figure 3.16 show that even with a data augmented dataset (yellow), the curves maintain a variance similar to the previous experiments. For reference, we also plotted several curves (blue) which are generated with the same parameters as the black curves of Figure **??**, and it can be evinced that the difference in accuracy between the best and worst performer of each group of curves is about equal. This

would suggest that the randomness cannot be reduced and that some training sessions are intrinsically more robust than others.

Furthermore, the curves corresponding to data augmented trainings demonstrate a consistently lower accuracy than their counterparts. The data augmentation process has moderately increased the complexity of the classification problem, and it could be that ResNet-18 is not powerful enough for such dataset, as it already proved in the experiments with CIFAR-10.

3.5 CIFAR-100

CIFAR-100 is a dataset very similar in structure to CIFAR-10: 50000 images for training, 10000 for testing, the only difference being the number of classes, which is obviously 100.

In order to test RegNet against classification problems with more than ten classes, we attempted to repeat our set of experiments on the CIFAR-100 dataset. However, we obtained no definitive result, and in this section we'll discuss the main obstacles that ultimately impeded a 100 classes implementation of RegNet.

Description of the issue

The Python3 scripts we used to perform most of our experiments are able to operate on a variable number of classes according to an input argument that can be specified for each run. When expanding the system from 10 to 100 classes, the experiment is able to run for the intended number of epochs, but at some point the loss function assumes value NaN, causing the subsequent gradient calculation and weight update to assume NaN values too; when the weights are all NaN, the output of each layer becomes NaN under any circumstance, causing a never-ending loop of malfunctions.

The interesting aspect is that up to a certain amount of classes the system works correctly, but a greater number causes the system to malfunction; more specifically, we found this threshold to be dependent on the encoder used, and for ResNet it is 62 classes, while for the shake-shake encoder it is around 20; this means that theoretically we could produce results for a ResNet classification problem on 62 classes, but not on 63.

Indeed, we attempted a RegNet implementation on 50 classes by halving the CIFAR-100 dataset, with no success. In fact, the loss assuming NaN value is not a deterministic process, but rather, it happens with a probability proportional to the number of classes. When we trained ResNet with 50 classes, the system started malfunctioning only after several thousand epochs; 63 classes instead, is the threshold where the probability of the loss assuming NaN value is so high that it happens almost always after the first batch. However, even in this case, repeating the experiment with 63 classes many

times could reveal a lucky case where the system runs correctly for several batches. In all honesty, at times the system malfunctioned even when training with 10 classes (MNIST), but it happened so rarely that we decided to just repeat the simulation without further investigations.

Debug

In the end, we haven't been able to solve the problem, but we were at least able to identify it. When the number of classes is high enough, the loss optimizer may cause the loss function of one of the classes to become NaN. When Tensorflow sums numeric values to a NaN value, the result is always a NaN; therefore, being the loss function as in (2.6), a malfunction in the loss of a single class i causes the total loss to break.

$$\mathcal{L}_{cov,\,x} = \frac{1}{x}\left[\log\frac{|\boldsymbol{\Sigma}_{Tx}|}{|\boldsymbol{\Sigma}_{Ox}|} - d + \mathrm{tr}(\boldsymbol{\Sigma}_{Tx}^{-1}\boldsymbol{\Sigma}_{Ox}) + (\boldsymbol{\mu}_{Tx} - \boldsymbol{\mu}_{Ox})^{\boldsymbol{T}}\boldsymbol{\Sigma}_{Tx}^{-1}(\boldsymbol{\mu}_{Tx} - \boldsymbol{\mu}_{Ox})\right] \tag{2.1}$$

$$L_{TOT} = \sum_{i=0}^{d} \mathcal{L}_{cov,\,i} + \mathcal{F}_k(\mathcal{K}_i - 3) \tag{2.6}$$

We traced the error through all the encoder, and we found out that, after the optimizer updates the weights, it is always the first convolutional layer to produce the NaN values first. So the malfunction is caused either by the calculation of gradients in the loss optimizer or by the implementation of the convolutional layer function.

The first NaN value, that later propagates through the network, is probably caused by some calculation producing underflow. This is not unlikely, as we already encountered something similar with equation (2.1): when we were working on expanding the system from three to ten classes, the calculation of the covariance matrix's determinant often produced underflow and caused NaN values to appear in the loss function; in the end we solved it by increasing the precision of the calculation from float32 to float64. Therefore, it may be that the optimizer operates with float32 numbers, or that underflow happens even with float64 precision; in both cases, solving this underflow problem would be beyond our powers and time because it would require accurate knowledge of the implementation of Tensorflow optimizers. In synthesis, it's likely that, since the loss's complexity increases with the number of classes, a 100 class implementation of RegNet is too complex for common loss optimizers.

A hypothesis of this type would explain why the number of classes increases the likelihood of malfunction; the matrix of the outputs, after all, is of size (batch size x number of classes), and since it contains many values close to 0, a calculation requiring the multiplication of all elements, like a determinant, could produce underflow if the number of classes is high enough.

At any rate, as mentioned, we weren't able to solve the issue, because we felt the mechanisms of the optimizer were beyond our control. Aside from Adam optimizer,

which is the standard for RegNet, we tried employing RMS propagation and Momentum optimizers, seemingly with no difference in results.

3.6 Conclusions

In this chapter we explored the robustness behavior of RegNet. Experiments on the MNIST and CIFAR-10 datasets revealed that the couple ResNet + RegNet is able to grant a significant increase in robustness compared to a cross-entropy loss. Instead, the couple shake-shake encoder + RegNet proved less effective, and results suggest that the robustness boost provided by RegNet is probably heavily dependant on the choice of encoder used.

Further analyses suggest that the depth of the residual encoder doesn't affect robustness, that there is no connection between target mean value μ_T and robustness, that the training process intrinsically introduces some variance in the robustness behavior, and that this cannot be corrected with data augmentation.

An implementation of 100 classes RegNet on the CIFAR-100 dataset was attempted, but never completed, likely because the loss function was incompatible with Tensorflow's methods for calculation of gradients.

Future works

A continuation of this work should include an implementation of RegNet for the CIFAR-100 dataset, as well as other datasets with different amounts of classes. However, this will probably require a full reformulation of RegNet's loss function, which will also help solve the inconsistencies analyzed in Chapter 2.

Moreover, it would be interesting to analyze the robustness performance of different encoder types coupled with RegNet, how heavily the encoder choice impacts robustness, and what is the real contribution that the RegNet loss applies to adversarial robustness behavior.

Bibliography

[1] M. Testa, A. Ali, T. Bianchi, E. Magli, *Learning mappings onto regularized latent spaces for biometric authentication*, arXiv preprint arXiv:1911.08764, 2019.

[2] J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, arXiv preprint arXiv:1412.6572v3, 2015.

[3] A. Kurakin, I. Goodfellow, S. Bengio, *Adversarial Examples in the Physical World*, ArXiv preprint arXiv:1607.02533v4, 2017.

[4] M. Sharif, S. Bhagavatula, L. Bauer, M. Reiter, *Accessorize to a Crime Real and Stealthy Attacks on State-of-the-Art Face Recognition*, Vienna, Austria, CCS'16, 2016.

[5] T. Brown, D. Mané, A. Roy, M. Abadi, J. Gilmer, *Adversarial Patch*, arXiv preprint arXiv:1712.09665v2, 2018.

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, *Intriguing properties of neural networks*, ArXiv preprint arXiv:1312.6199v4, 2014.

[7] N. Papernot, P. McDaniel, X. Wu, S. Jha, A. Swami, *Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks*, Arxiv preprint arXiv:1511.04508v2, 2016.

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

[9] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, B. Frey, *Adversarial autoencoders*, arXiv preprint arXiv:1511.05644, 2015.

[10] X. Gastaldi, *Shake-Shake regularization*, ArXiv preprint arXiv:1705.07485v2, 2017.

[11] D. Warde-Farley, I. Goodfellow, *Perturbation, Optimization and Statistics*, Chapter 1:*Adversarial Perturbations of Deep Neural Networks*, Cambridge, Massachusetts, The MIT Press, 2016.

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

[13] A. Almahairi, S. Rajeswar, A. Sordoni, P. Bachman, A. Courville, *Augmented cyclegan: Learning many-to-many mappings from unpaired data*, arXiv preprint arXiv:1802.10151, 2018.

[14] A. Fawzi, S. Moosavi-Dezfooli, P. Frossard, S. Soatto, *Classification regions of deep neural networks*, arXiv preprint arXiv:1705.09552, 2017.

[15] Bengio, Samy; Goodfellow, Ian J.; Kurakin, Alexey (2017). *Adversarial Machine Learning at Scale*, arXiv preprint arXiv:1611.01236, 2016.