UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Comparative implementation of Greedy InfoMax and Slow Feature Analysis for self-supervised neural networks

Degree Final Project

Degree in Computer Engineering

Author: Manuel Roselló Oviedo

Tutors: Oswin Krause (KU)
Francisco Casacuberta Nolla (UPV)

Course 2019-2020

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

etsinf

KØBENHAVNS
UNIVERSITET

UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

Abstract

Over the past years, supervised models with end-to-end backpropagation have dominated the Deep Learning paradigm. However, great potential can be observed in out-of-the-box approaches like *Greedy InfoMax*, an unsupervised approach based on the modularization of neural networks and driven by the goal of optimizing local information as an alternative to global error backpropagation. This novel technique has been proven useful for experiments in both visual and auditive domains. The experiments described in this report aim to implement this method and compare it with a well-established approach for Unsupervised Learning: *Slow Feature Analysis*. This second method focuses on the extraction of slowly varying features from a quickly varying input signal and has also been proven effective for numerous tasks.

Keywords— Unsupervised Learning, Slow Feature Analysis, Greedy InfoMax, Backpropagation, Mutual Information Optimization

Acknowledgements

The realization of this project was a journey full of both fascination and frustration. It all started with the excitement of a new city, a new culture and a new perspective brought by the independence of living alone. Everything took a strange turn with the COVID-19 pandemic and the expectations set for my exchange semester suddenly became foggy. The disappointment and the sudden changes lead to days of disorientation and impotence, not knowing how things would possibly look like in the upcoming weeks. However, uneasiness slowly faded away thanks to the unconditional support of my parents, Barb and both the newly met friends that stayed in Copenhagen and those sending me their kind words from home. Even if it might not seem much, this work is dedicated to all of you.

Nonetheless, the person who I would like to thank the most is my local supervisor from Copenhagen's University, Mr. Oswin Krause. Among the countless professors I tried to contact via e-mail, he was essentially the only one who considered my petition to partake of an Study Project, and the main ideas for the research were his. After four years of studying for a Bachelor's Degree, I felt that I had some general knowledge on many different fields but I was expert in none. Now, and despite the undeniable fact that I am still far from being able to consider my self an expert, I discovered the captivating world of Unsupervised Learning thanks to your guidance, and I got the chance to feel *"like a scientist"* for the first time in my life by actively researching state-of-the-art papers and struggling for weeks to properly grasp the techniques and mathematical tricks behind the latest innovations on Machine Learning. It was tough at times, but the outcome was always extremely satisfying. Moreover, I have felt fortunate to be mentored by you ever since the first day; your positive and friendly attitude —even when from time to time I went through some not-so-productive days— pushed me through the occasions when I felt stray and overwhelmed. I sincerely thank you for all your patience and support.

Finally, I would also like to thank the professors back at UPV that nourished my interest on Machine Learning and helped me make it this far. Notably, I want to thank Ms. Eva Onaindia for her proximity and encouragement; and Francisco Casacuberta for making one of my favorite courses even better with his teaching and supporting me as a home tutor for this project.

This crazy-paced experience is approaching its end and I can only look forward to a future in which I will hopefully find myself immersed again in international and multicultural contexts, applying the knowledge that I have acquired throughout these years while still expanding its limits in a process of constant learning.

Manuel Roselló, June 2020

Contents

List of Figures

List of Algorithms

List of Tables

1 Introduction

Over the past decade, Deep Learning models based on supervised methods with global back-propagation have become the flagship of their paradigm and are currently present in numerous fields. These models seek achieving one of the most ambitious goals Computer Science has been pursuing since its origins: imitating and improving the human intellect. Perhaps as a result of the wish to create them *in our image and likeness*, or simply by taking inspiration on how nature has built the very same intelligence we aspire to replicate; Artificial Intelligence (AI) has historically been firmly tied to the understanding of the ways of the human brain.

Machine Learning (ML) is often defined as a form of AI in which the machine tries to implicitly learn the rules to optimize its performance for a given task, as opposite to traditional AI methods where the rules are explicitly given to the machine, which then tries to use them in order to improve said performance.

Among the many techniques proposed through the years, it is probably Deep Learning the one whose popularity has grown faster in the last decade. Even though other truly competitive alternatives like Random Forests [1] and Support Vector Machines [2] have been in the game for a long time, Deep Learning's versatility when it comes to complex problems, as well as its widespread usage by big companies who handle massive volumes of data, have helped turning this approach into a trending topic. This can be easily checked by looking at the search frequency in Google Trends[1] for some terms like Machine Learning, Deep Learning or Neural Network. Figure 1 shows the search interest of these terms in relation to their highest point of the chart for a given time period, in a 100-point scale. Over the past decade, and specially over the past four years, the interest on Machine Learning topics has undergone a notable increase alongside the unprecedented technological growth that the field has experienced. More and more universities are including advanced and specific ML-based courses to their curricula, setting the base for an upcoming generation of specialized engineers that will try and push the existing boundaries even further.

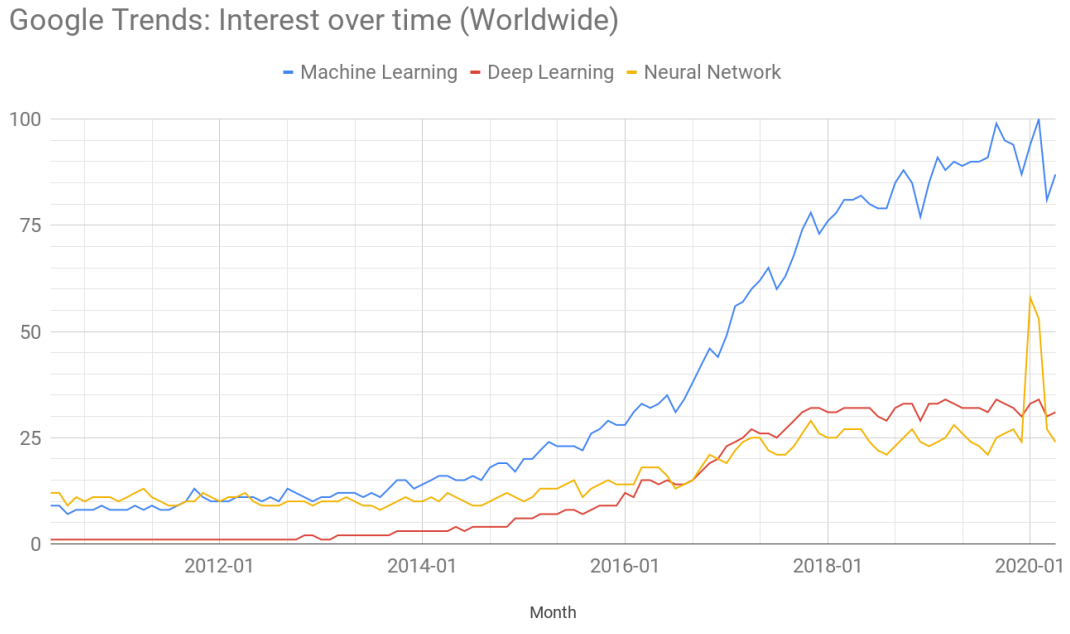

Figure 1: Google Trends historical search data from May 2010 to May 2020 (Worldwide).

Every day, news stories talk about new technologies based on Deep Learning. Despite the information often not being entirely accurate as tech-related press articles have been historically known for prioritizing nicely sounding words over scientific accuracy, it is a fact that —for better

1 Google Trends online tool by Google LLC

rather than worse— the Machine Learning research field is in the spotlight, and this growing interest can be highly beneficial for its further development in the upcoming years.

As initially stated, the vast majority of the models used for Machine Learning nowadays consist of Deep Learning, by means of layered —and typically Convolutional— Neural Networks (NN or CNN for the Convolutional case) with global backpropagation of the error. This is of course a wide generalization and, in practice, many other variations are used. Nonetheless, these characteristics have become predominant due to its repeatedly showcased power for various tasks, and are usually used as a reference example for teaching Deep Learning. However, the study of alternative approaches opens up a wide range of potentially useful models.

Unsupervised Learning (UL), although less popular than its Supervised counterpart, has also been proven useful for several tasks. It specially finds its time to shine in clustering, visualization and dimensionality reduction problems [3, 4, 5]. This type of learning will be covered in more detail in the Background section, but the key difference with respect to Supervised Learning is that UL does not require labeled samples for the training process. Imagine a simple classification task, where a CNN is trying to learn how to differentiate between different objects: animals, vehicles, plants... This is a common computer vision problem, where Supervised Learning would use labels as a way to assess both how *correctly* it is learning during training and how accurate it is when testing. A loss function such as cross-entropy can be used for the former and a simple error rate estimation for the latter. In both cases, class labels are used to compare the actual class of the objects with the class predicted by the model, and these labels have most likely been assigned manually by a human beforehand. Unsupervised Learning, on the other hand, would typically operate without using those labels —except perhaps for testing—, thus saving work time. Instead of following the *"This is a car"* logic that *teaches* the machine, UL makes use of different approaches that allow the machine to learn by using the data without being *taught*; simply by extracting features that would, following the previous example, allow the model to tell apart a car from a dog and identify them properly after having learned which features most cars have in common and which features they do not share with dogs. One could argue that a Supervised CNN would also find such features, but the main difference resides in the fact that it would do so after being *explicitly* taught on whether its decisions are right or wrong based on previously-gathered information about the data, while an UL model would have learn this criterion by experience and with no given information about *how to do it* or *what to look for*.

While UL might at first seems to work in mysterious or ambiguous ways, there is a good reason for its success. This reason usually takes the form of an algorithm that is in charge of the *magic* thanks to which the model can learn on its own. One of these algorithms is the well-known Slow Feature Analysis (SFA) [6], which focuses on finding features in the data that are invariant or, more precisely, vary *slowly* in a temporally (quickly) varying signal. In a self-descriptive fashion, these are called *slow features*. In order to properly understand their relevance, a definition must be provided for the concept of *slowness*, which is in fact relative. Imagine that the samples for an analysis are the picture frames of a nature documentary on different Savannah animals and their behavior: the lions are shown sleeping, then hunting, then there is a shot of a running prey... And after a while, the focus goes to some birds and their daily routine, and so on for other species. The takes are different and some "features" will vary quickly, such as the individual values for the pixels in the screen. However, there are other "features" that will take up to several seconds to vary, such as the species of the animal shown in the screen. This is due to the nature of real world objects, which are usually highly persistent and change in a mostly continuous way. These are of course —relatively— much *slower* than the pixel values when the time-scale in which they vary is compared. SFA will try to detect this second type of features and extract them from the sensory input via Unsupervised Learning.

Again, these concepts will be explained in their corresponding Background section. Nonetheless, it is important to introduce them now, since slow features are the basis of the novel Greedy InfoMax method [7] for Self-supervised (Unsupervised) Learning on which most of this experiment is based. This method does not only employ UL, but also renounces traditional end-to-end backpropagation by means of splitting the network into individual modules with greedy optimization of a local InfoNCE-based loss [9, 8], with the ultimate goal of maximizing the preservation of mutual information in natural data. The modules can be composed of one or several convolutional layers. This procedure encourages, in fact, the extraction of slow features.

In this document, both Greedy InfoMax and Slow Feature Analysis will be described and implemented with the ultimate goal of both comparing their performance in different tasks and architectures, and discussing whether such comparison is applicable. Namely, the two main tasks will consist of a first experiment based on Unsupervised Learning of image data applied to Supervised classification, and a second, fully-unsupervised experiment focused on feature extraction from images.

1.1 Structure

Up to this point, the project and its scope have been presented. In the following section, this will be expanded by an exposition on the motivation of the research, focusing on the state of the art and some related works, as well as the expected results and the relation of the main topics with previous courses.

The work plan of the project will be described next; enumerating the main objectives and describing the employed technologies and methodologies. Then, a theoretical background is provided in order to conduct a detailed explanation regarding the fundamental concepts on which the experiments are based. First, a full description of Unsupervised Learning develops some of the terms that were first featured in the Introduction section. Then, the concept of End-to-end backpropagation is explained, devoting particular attention to both its relevance in the current technologies and the problems or flaws this technique presents. Lastly, both Slow Feature Analysis and Greedy InfoMax are thoroughly explained and introduced as the main approaches for the experiments.

Subsequently, implementation details of the pre-processing patch extraction algorithm and the InfoNCE-based custom loss function are provided both formally and algorithmically, as well as the specifics on how the models were built for the different architectures. This section is followed up by the reports on both experiments. For each experiment, the dataset and followed procedure are first introduced and then the results comparing the different parameters and configurations are presented in the form of tables and plots.

Finally, the Conclusion section summarizes the key ideas and the knowledge acquired throughout whole working process, while also hinting at some possible improvements that could be potentially addressed in future projects.

2 Motivation

This section aims to provide arguments to justify the choice of the topic of research. First, the latest related works will be outlined, highlighting their impact and relevance. This overview of the state-of-the-art will be followed by a brief exposition on the expected results. Finally, the relation of the project with previously coursed subjects will be argued as to consolidate the coherence of the project with the Bachelor's curriculum.

2.1 State of the art

As indicated by the authors in its original paper [7], Greedy InfoMax is not alone when it comes to novel initiatives looking for viable alternatives to end-to-end backpropagation. Balduzzi et al. [10] (2015) decomposed backpropagation into interacting learning sub-algorithms and factorized the error signals to derive an non-parametric regression algorithm known as *Kickback*. Scellier and Bengio [11] (2017) introduced the concept of *Equilibrium Propagation* for energy-based systems, which only requires gradient computation for the prediction phase while simply performing a nudging of the prediction during the second phase, in a sort of propagating perturbation signal that is tantamount to backpropagation of error derivatives, but more biologically plausible. Kohan et al. [12] (2018) also advocated for biological plausibility, even if accepting that *"how the brain solves the credit assignment problem is unclear"*, referring to weight update or backpropagation, and proposed reusing the forward connections by feeding the errors to the input layer in an Error Forward-Propagation mechanism. In fact, recent studies [Bartunov et al. [13] (2018), Xiao et al. [14] (2019)] have focused on comparing different biologically-plausible approaches with promising results, proving that it is possible to relax the weight symmetry requirements imposed by backpropagation while performing on equal level with backpropagation-based models on increasingly complex architectures and datasets. On a different take closer to modularity, Belilovsky et al. [15] (2019) use 1-hidden neural networks sequentially as *auxiliary* layers capable of solving problems with a performance similar to well-established architectures, scaling to ImageNet [16].

Regarding Unsupervised Learning, the amount of works based on Slow Feature Analysis provide enough evidence as to why it is the reference to which variations such as Greedy InfoMax are compared. Ever since its formal introduction in 2002 by Professor Laurenz Wiskott [6], slow fature methods have experienced a flowering and are still present in state-of-the-art researches. Franzius et al. [17] (2018) encoded slow features from spatial, high dimensional image data via UL to represent camera positions and achieve an accurate system of self-localization, emulating orientation cells in the hippocampus of rats, an experiment that shares resemblance with Wiskott's own work on place, head-direction ans spatial-view rodent cells from 2007 [18]. Both experiments are based on the fact that cells selectively encode some aspects, such as position and orientation, while being invariant to others.

Zhang and Tao [19] had already tested with human action recognition in 2012 for both supervised and unsupervised SFA methods, demonstrating its capability to extract patterns and recognize complex multiperson activities. In a topic with more relation to this project, Ghosh et al. [20] combine SFA with Extreme Learning Machines [21] to perform pose-invariant object recognition by encoding vision events, achieving an incredibly low error rate after training with 8 objects varying on over 90 degrees (poses). One of the main inspirations for this project's experiments is the recent work by Wiskott et al. [22] where an SFA-based model is capable of learning the pose variations on grayscale objects. These experiments also lead to practical applications as seen in the research by Du et al. [23], which focuses on change detection in multi-temporal data via remote sensing technology and applies a custom variation of SFA named Deep Slow Feature Analysis, where two symmetric networks project data of bi-temporal imagery so that unchanged components are later suppressed to easily measure the changed features, outperforming state-of-the art methods for this kind of task —including other SFA-based algorithms. Even the futuristic Quantum Machine Learning [24] has been combined with Slow Feature Analysis [25] for both dimensionality reduction and classification tasks with high efficiency and accuracy on its results.

Finally, Contrastive Predictive Coding [26] (2018) proposes *"a universal unsupervised learning approach to extract useful representations from high-dimensional data"* and sets a precedent for Greedy InfoMax regarding future predictions in latent space and maximization of the mutual information between a temporal signal and its context via a contrastive loss function based on NCE.

These concepts will be explained in the upcoming sections as they are crucial for the experiment. This type of approach is also used by Bengio et al. [27] to study the unsupervised learning of representations with yet another variant labeled as *Deep InfoMax*, also based on InfoNCE, this time combining it with the power of Adversarial Networks [28]; being capable of outperforming other unsupervised methods for classification tasks. Tschannen et al. [29] (2020) recently published a paper emphasizing the relevance of Mutual Information and the InfoMax principle, a concept on which both Greedy InfoMax and CPC are based, explaining its success in many experiments for the past few years.

2.2 Expected results

The main goal of this research is to develop simple models capable of obtaining significant results so that a comparison between Greedy InfoMax and Slow Feature Analysis regarding performance for different tasks is possible. This comparison is not arbitrary; it is based on the fact that both methods work with the concept of *slowness* in data features (see Section 4.3). By the end of the project, two different Convolutional Network architectures will have been built: a Multi-Layer, Single-Module version and its Multi-Layer, Multi-Module counterpart. Both of them will be able to perform both classification and feature extraction tasks based on slow features, either via GIM or SFA. The NORB dataset [30] will be used for both experiments.

Image classification is a well-known task in the Machine Learning and Computer Vision fields. Both unsupervised methods have demonstrated their capability in individual experiments, so applying them for the same problem and with the same architectures will allow assessing which approach is capable of obtaining better results. Löwe's experiments achieved up to an 81.9% accuracy when working with a bigger dataset formed by RGB images. Therefore, a result above 80% will be considered acceptable given the smaller, grayscale dataset that will be used and the dimensional limitations of the model. To avoid any possible misconception note that, even if the training process will be unsupervised, labels will be used to test the classification accuracy of the models in a classical supervised fashion.

Regarding the other experiment, the feature extraction process will be focused on the specific case of pose (angle) variance detection with independence to lighting conditions. This experiment is inspired by the SFA-based work by Wiskott et al. [22] and the final model should be able to learn such variations in an unsupervised fashion; obtaining results similar to those in the reference paper, as the same dataset is being used.

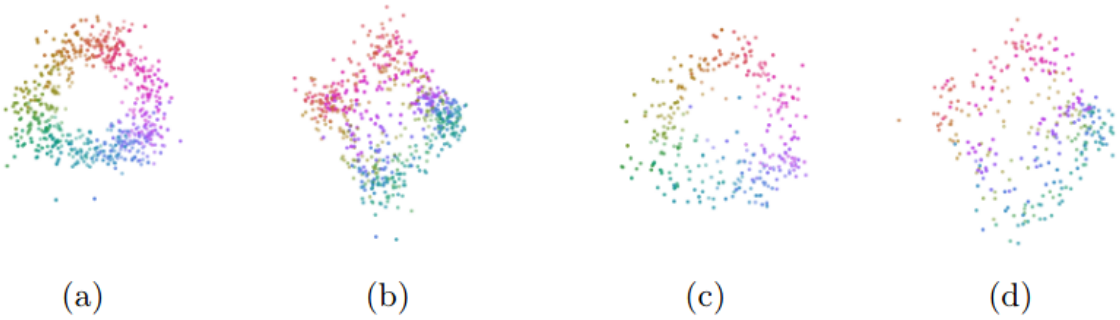

Figure 2: Training (a, b) and testing (c, d) results front and side visualizations on the NORB dataset, with azimuth colored. The circumference of the cylinder encodes the rotation of the plane. From [22].

The motivation for this goal is to visually analyze the results to assert the coherence of the learning results obtained by the model with the actual pose variations, such that the actual data for the azimuth and elevation values is translated into a radial shape whose correlation with the values can be confirmed by the continuity in the color mapping.

The resulting models will therefore be capable of performing object classification problems, with pose and lighting independence, as well as being capable of (implicitly) identifying such pose. Even if currently limited by scheduling and technological constraints, a more complex version of the models can be built in the future, allowing for further improvement on both tasks. Both main experiments focus on grayscale images, although a previous toy experiment was performed on RGB

imagery. Even if it was not the main objective, this challenging variation could also be addressed in future projects since [7] demonstrated how Unsupervised approaches based on the maximization of mutual information can be competitive for such tasks.

2.3 Relation with previous courses

The realization of both the theoretical and practical parts of the project was only possible thanks to the knowledge previously acquired from the undertaken courses during the four years of Bachelor's Degree studies, which are listed hereafter. Not only that, but given that learning is a continuous and cumulative process, the inclusion the courses undertook in Copenhagen was considered to be imperative: one of which was just recently finished; and a second one that is currently on-going.

- **Polytechnic University of Valencia**
  - **Intelligent Systems** (2018). This subject served as a formal introduction to Form Recognition and Automatic Learning. It provided with basic notions on discriminant functions, the Perceptron algorithm, K-means and other concepts that later allowed taking the first steps in the ML field.
  - **Perception** (2019). As a continuation of Intelligent Systems, this course focused on some key concepts for ML works, such as feature extraction, dimensionality reduction, kernels and classifiers; all of which have either explicitly or implicitly been applied in this project.
  - **Machine Learning** (2019). The main foundations of this work are directly related to the syllabus: learning models, optimization techniques, error backpropagation and multilayer neural networks. In fact, this course served as an inspiration to pursue a project related to Neural Networks.
  - **Others**. Several other courses have a minor relation with this work, but also had an invaluable impact that made it possible: Databases and Information Systems (2018) served as an introduction to Python programming and in-code data management, Project Management (2018) provided with valuable guidelines regarding planning and scheduling; and Bioinformatics (2019) significantly widened the views of the students on the real applications of ML.
- **University of Copenhagen**
  - **Signal and Image Processing** (2020). Grasping the concept of convolution can be tricky at first, and many students find themselves working with Convolutional Neural Networks way before they are taught about how they operate, or rather, how they *convolve* in order to operate. This course helped fully understanding the meaning and relevance of convolutions. In addition to that, the PyTorch library was used for an assignment and many other ML concepts were applied during the practical sessions.
  - **Large Scale Data Analysis** (2020). This course, even if quite similar to the Machine Learning one, expands its contents and goes beyond with MSc-level lectures on topics such as the Adam algorithm, Recurrent Networks and Distributed Data Analysis. This course was also taken in parallel with this project, and it served as an additional source of knowledge.
- **University of Liège - Board of European Students of Technology**
  - **Root for Deep Learning** (2019, summer course). Even though probably attended one year too early, this summer course was an entire MSc course on Deep Learning summarized in one week of daily classes. Most of the work was done in PyTorch and the syllabus went through all the aforementioned key concepts and even more. In spite of the lack of preparation back then, it still served as an initial contact with the big picture of Convolutional Neural Networks, their —at first, overwhelming— complexity and their outstanding potential.

3 Work plan

This section will delve into the organization and scheduling of the project. Namely, the main objectives will be presented, followed by an exposition on the main technologies used and an account on the specific stages of the work plan.

3.1 Objectives

Two specific objectives were initially set in order to develop this project, from which other sub-objectives emerged as the project evolved:

1. Construction of a simple Unsupervised Learning model with PyTorch [31], implementing a custom loss function based on Greedy InfoMax's InfoNCE.
   (a) Construction and comparison of a single-module architecture versus a multi-module architecture.
   (b) Performance evaluation of the model for both image classification and pose-related feature extraction independent to lighting.
2. Comparison of the models against an SFA-based version of them; both given the same tasks and architectures.

3.2 Technologies

The practical nature of the project entailed the usage of modern coding technologies. The Python Programming Language[2] was used for all the experiments. Being a general-purpose language, the recent development of some specialized libraries such as PyTorch[3] or TensorFlow[4] have helped it become one of the most popular programming languages for Machine Learning tasks. Readability and user-friendliness are its key strengths, and the final choice to use PyTorch over other libraries for the development of the experiments was in part due to the fact that this library was built such that it would feel native and fully integrated in PyTorch, as opposite to TensorFlow.

However, the main reason for the use of PyTorch was to ensure consistency with the original experiments performed by Löwe et al. [7]. The library stands out in Deep Learning due to its automatic differentiation feature *autograd*, which will handle the gradient backpropagation during training (see Section 4.1). Moreover, the data is manipulated in complex structures called *tensors*; multidimensional numerical arrays similar to NumPy arrays but capable of being operated with hardware acceleration thanks to the CUDA[5] technology, as long as a capable GPU is available. PyTorch also offers a wide range of pre-built functions for data preprocessing, network construction and optimization.

Instead of using traditional Python files, the code was written in Colab[6] notebooks, browser-based files which allow for the interactive execution of code in independent cells within the same session. Cells can also contain text and images, which enables creating illustrative textbooks and tutorials. The main difference with respect to standard Jupyter Notebooks[7] is that Colab offers the user an option to run their code on a cloud server, being able to use remote hardware acceleration via GPU or TPU, which is especially handy when training of big Deep Learning models. Of course, the service is limited for free users, and even those with a paid subscription have certain restrictions since hardware availability changes dynamically depending on the current overall usage. Due to these limits, the experimentation was sometimes delayed when the personal usage was exceeded. Nonetheless, the models were trained locally for the feature extraction experiment as the dataset was smaller and the training process could be performed with a simple mid-market NVIDIA GPU, thus parallelizing this process with the remote training of the classification modules in Colab.

2 Python Programming Language
3 PyTorch, open source ML library.
4 TensorFlow by Google Brain.
5 CUDA by NVIDIA Corporation.
6 Colaboratory by Google LLC.
7 Jupyter Notebook by Project Jupyter.

3.3 Methodology and work plan

An Individual Study Project worth 15 ECTS is equivalent to 412 hours of work. This project was developed over a span of 18 weeks, with an average of 20 hours of workload per week, including a weekly 1-hour meeting with the project supervisor. Due to the COVID-19 situation, most of these meetings took place remotely via Skype, and e-mail communication was frequently used for both problem solving and update reporting outside meeting hours.

The work plan was split into 4 stages:

1. Initial research and familiarization with the technology (1 month). The first weeks were devoted to thoroughly studying the reference papers via both individual reading during the weekly meetings in which specific goals were established progressively.

   The models were built using the PyTorch Deep Learning library for Python. During this first stage, most of the work time was dedicated to learning with tutorials and performing small experiments in order to properly assimilate the basics of PyTorch and grow accustomed to working with Google Colab notebooks.

2. Greedy Info Max implementation, CIFAR-10 experiment (1 month). A toy experiment based on image classification with the CIFAR-10 dataset (Sections 5.1.1 and 6.1) was built in order to test possible ways of implementing Greedy InfoMax. Specifically, two architectures were built: a Single-Module model with four convolutional layers and a Multi-Module version that split those into two modules of two convolutional layers each. Both architectures used a custom InfoNCE-based loss for training. Most of the work was focused on coding the different blocks: the custom loss, the patch extraction functions, the custom dataloaders and the architectures —mostly aiming to find an optimal way to implement the modularization.

3. Slow Feature Analysis and the NORB experiments (2 months). Once the CIFAR experiment was up and running, it was the turn for the main experiments based on the NORB dataset (Sections 5.1.2, 6.2 and 6.3). The models and dataloaders were adapted to the new data and an SFA-based version was implemented. Since every execution would take up to several hours and the usage of Google Colab was limited, this process took place in parallel with early drafts of the project report.

4. Report completion (1 week). Once the experiments were finished, a week was devoted to compiling the results into plots and tables, and focusing on completing the written report.

5. Final revisions (1 week). A final draft was sent to both supervisors so that the last days of work could be invested in reviewing and fixing possible issues before the official hand-in.

4 Background

Before diving into the experiment details and their implementation, a detailed exposition of the main theoretical concepts on which they are based is imperative. First, the Unsupervised Learning paradigm will be explained and some examples of its applications will be presented. Then, the basics of backpropagation will be explained to justify its relevance and its downsides, which Greedy InfoMax attempts to subside. Following this, the main logic behind Slow Feature Analysis will be described as an introduction to the implementation that was later applied in the experiments. Lastly, a thorough justification of Greedy InfoMax will take place, focusing on its relation with Contrastive Predictive Coding and the InfoNCE loss.

4.1 Unsupervised Learning

Traditionally, the most common approach for dealing with Machine Learning tasks has been *Supervised Learning*. The term is named based on the fact that the machine is given *both* inputs and their desired outputs during training, with a goal to produce a correct output for any new inputs once it has been trained. The Supervised Learning application par excellence is classification: physical objects, handwritten digits or words, spoken language, etc.

Another popular type of machine learning is *Reinforcement Learning*, which can be generalized as an application of *game theory*. There are many possible variations, but the main precept is letting the machine interact with its environments via actions that can be rewarded or punished. The machine will then aim to improve its behavior progressively so that its score is maximized.

However, the approach that was selected for the experiments in this project is the so-called *Unsupervised Learning* or *Semi-Supervised Learning*. Unlike the aforementioned methods, Unsupervised Learning (UL) obtains neither target outputs nor rewards [32]. Instead, the goal of UL techniques is to extract internal representations of the implicit structures present in the input data [33]. Even if this might sound vague or even unattainable at first sight, there are several techniques that have been proven capable of achieving such premise. For *semi-supervised*, just a small amount of the data is labeled, that is, has a given output. Of course, the lack of need for sample labeling is highly appealing since this is often done manually, resulting in a tedious but fundamental task to perform before training any Supervised model.

Unsupervised Learning has repeatedly proven its usefulness for various tasks in several fields, as commented in the Motivation section. Perhaps one of the most relevant applications is anomaly detection, for instance, in multivariate time series data for *wereables* or even power plants [34]; or in potentially contaminated, high-dimensional data [35] via Generative Adversarial Networks [28]. Reviewing some of the examples from the previous sections, the variety of applications for UL techniques becomes evident. Recapitulating, related work has been done for networking problems such as traffic engineering and classification, quality of service optimization and, again, anomaly detection [36]. Some data mining methods [37] use UL to determine association rules in data [38]. In addition to these fairly novel uses, UL has been a staple approach for traditional tasks like classification and dimensionality reduction for decades. For instance, [32] shows how an unsupervised model is perfectly capable of surpassing an equivalent supervised model for a simple pattern classification task.

Some authors emphasize the neuroscience inspiration and applications of UL [39], mainly based on the argument of the biological implausibility of current Deep Learning networks, an argument which is also mentioned as part of the motivation for Greedy InfoMax [7]. All three mentioned variants of Machine Learning are somehow based on different ways of reproducing the way human beings learn and process information in their brains. UL models can be formally explained within a mathematical and statistical framework, but the original idea was to imitate how the brain is able to extract statistical patterns from complex sensory data.

Nowadays, there are several UL techniques available, most of them focused on clustering methods or K-nearest neighbors [40]. This project will focus on two approaches based on the detection and analysis of *slow features* —which will be described in Section 4.3— with the usage of Deep Learning models in the form of Convolutional Neural Networks, albeit a cluster-related technique (nearest centroid) will be used for classification. This technique assigns for the data the label of the class of training samples whose mean is spatially closer.

4.2 End-to-end Backpropagation

Along the past sections, the term *backpropagation* has been repeatedly used without further explanation other than it referring to a backpropagation "of the error along the network". The history of the algorithm is, surprisingly enough, a sequence of discoveries and rediscoveries followed by years in the shadow just to become a relevant topic again later thanks to new scientific breakthroughs. Even if some preceding studies in control theory [42] and derivation [43] in dynamic programming are often regarded as the initial steps towards its development, it was not until 1986 that a paper published by David Rumelhart, Geoffrey Hinton, and Ronald Williams [41] helped it start to gain popularity and eventually become the workhorse it is today. They described how backpropagation allowed for faster performances when compared to its contemporary alternatives but, more than that; it allowed solving certain problems that were considered practically insoluble with the available technology back then.

The relevance of backpropagation is excellently summarized in this quote by Balduzzi et al. from [10]: *"The discovery of error backpropagation was hailed as a breakthrough because it solved the main problem of distributed learning* [...] *Decades later, Backprop is the workhorse underlying most deep learning algorithms, and a major component of the state-of-the-art in supervised learning."* Of course, this statement can be extended to Unsupervised Learning as well, as backpropagation is also a standard algorithm used in most unsupervised models.

To put it bluntly, end-to-end backpropagation is a mechanism that allows a learning model to properly adjust its *weights* based on its previous performance(s), with the goal of finding an optimal value for them. The concept of *weight* must be interpreted in its mathematical sense, that is, how relevant a piece of data is for the model: for instance, an input value for the well-known *Perceptron* algorithm [44], whose multilayer version is possible thanks to backpropagation. It is not in the scope of this document to explain how models such as a multilayer Perceptron or a neural network operate, and it is assumed that the reader is knowledgeable or has at least some basic notions on these topics. The objective of this section is to focus on the reason why backpropagation is indispensable in Deep Learning models, as well as to outline some of its biggest flaws and counterarguments.

After a neural network model is initialized, the input data undergoes a *forward* propagation from the first layers and until the output layer. This output is then evaluated by a *loss function*, such as MSE (Mean Squared Error), that will estimate the correctness of the results. The goal of backpropagation is to force the network to change its internal parameters in such a way that this loss or error will be optimally low. Many optimizers handle this task in different ways, such as the classic SGD [45] or other novel, fast algorithms such as Adam [46] or RMSProp [47]; but all of them are based on the common goal of computing the gradient of the loss function with respect to the weights. In other words, it is an iterative differentiation process based on the chain rule that allows updating the weights of a network in an —ideally— optimal way.

Take for instance a simple multilayer perceptron with one input layer, one hidden layer and one output layer. For simplicity, let us assume that the function to minimize is the Mean Squared Error. For a given training set $S = \{x_1, \dots, x_N\}$, the goal is to find the set of weights Θ such that the error

$$\varepsilon_S(\Theta) = \frac{1}{N}\sum_{n=1}^{N}\varepsilon_n(\Theta); \qquad \varepsilon_n(\Theta) = \frac{1}{2}\sum_{i=1}^{M_L}\left(t_{ni} - s_i^L\left(x_n;\Theta\right)\right)^2 \tag{4.2.1}$$

is minimized, where M_L is the number of nodes in the output layer L, t_{ni} is a target value (label) for a sample n and s_i^L is the output of the node i in the output layer as s_i^k is the output value for node i in layer l, which in the current case example of $L = 2$ would be calculated for the hidden and output layers as

$$s_i^1(x;\Theta) = g\left(\sum_{j=0}^{M_0}\theta_{ij}^1 x_j\right); \qquad s_i^2(x;\Theta) = g\left(\sum_{j=0}^{M_1}\theta_{ij}^2 s_j^1(x)\right), \tag{4.2.2}$$

where g is an activation function. For each weight θ connecting neuron j in layer $l-1$ with neuron i in layer l, its gradient $\Delta\theta_{ij}^l$ in a simple gradient descent algorithm with a learning rate α would

be computed as

$$\Delta\theta_{ij}^{l} = -\alpha\frac{\partial\varepsilon_S(\Theta)}{\partial\theta_{ij}^{l}} = \frac{1}{N}\sum_{n=1}^{N} -\alpha\frac{\partial\varepsilon_n(\Theta)}{\partial\theta_{ij}^{l}}. \tag{4.2.3}$$

When dealing with high-dimensional data, the GPU must be able to fit in the entire computational graph at once (weights, activation functions and gradients). A simple approach would be reducing the batch size; but the main issue is that, as a network becomes deeper, the backpropagation operation becomes increasingly expensive due to the amount of trainable (differentiable) parameters present in the layers. Asynchronous training of the layers might be a solution, but traditional backpropagation requires sequential continuity, that is, each layer requires the activations from its predecessors. In addition to this, tracking issues becomes more difficult as well since often not even the designer can be certain about which layer is faulty and how the modifications to one layer would affect the overall performance and the final output.

Moreover, deep networks can suffer from the infamous *vanishing gradient problem*. This issue appears when the gradients of the loss function produce values close to zero, preventing the model from properly updating its weights and consequently preventing it from learning. The source of the problem are certain activation functions such as sigmoid, whose derivative will approach zero when the input values are large. There are many popular solutions for this. The simplest approach is to simply use a different activation function whose derivatives will not easily decrease, such as ReLU. With time, other tricks such as using residual networks or batch normalization layers [48] have also been popularized, allowing for the fast growth of Deep Learning networks in the past decades.

Lastly, modular networks such as the one built for this project also address this issue by helping the backpropagation operate locally, in shallower networks where the gradient values are less likely to decrease excessively. Usually, neural networks use end-to-end or global backpropagation, meaning that there are no modules and the differentiation process has to be performed from the final layer to the first one for every optimization iteration. Even if trained for the same amount of epochs, a modular network is expected to have way less computational overload per module since only the gradients of a few layers will be stored simultaneously in memory. Note that the concept of *module* can refer tom both a single layer or a combination of them and, overall, a modular network can be seen as a combination of smaller networks, which will not necessarily be connected in a sequential way —although this will be the case for Greedy InfoMax and the project's model architecture.

In Pytorch, backpropagation is completely automatic via *Autograd* (automatic differentiation). This is handled by the `backwards()` operation, as long as the given function is differentiable and has its gradient computation enabled (`requires_grad=True`).

4.3 Slow Feature Analysis

Usually, when *features* are mentioned in a Machine Learning context they refer to common patterns found in static data. Take for instance a child who is learning what a dog is. There are many different breeds of dogs, so their brain will have to find those common structures or patterns: four legs, fur, waving tail, barking sounds... These characteristic traits o *features* will be the key to differentiate them from a cat, a horse or a bird. Modern convolutional networks are designed learn to find these features so that they are able to perform, for example, the traditional classification task described in the previous section, just like the child learns to tell different species apart.

However, the so-called *slow features* are —mind the pun— a *slightly* different animal. As defined by T. J. Sejnowski [6], they are "*invariant features of temporally varying signals*". In order to clarify this statement, the first example given by his article *Slow Feature Analysis: Unsupervised Learning of Invariances* will be summarized.

Assume a bounded visual field, through which three objects in the shape of striped letters are displaced in a straight line, each one of them in a different direction. Also assume that only one object will be visible within the field at a time. The stimulus can be represented by three variables changing over time: object identity, vertical location and horizontal location. These stimuli are perceived by photoreceptors that detect grayscale changes, a kind of information that

changes rapidly and thus is a low-level representation containing relevant information about the variables. However, if the photoreceptors were to cover the entire visual field then a high-level representation is obtained, which varies on a different timescale. The key fact is that "*a slowly varying representation can be considered to be of higher abstraction than a quickly varying one*". While object translation induces quick changes in the primary sensor signal, object identity *and* object location vary slowly and can therefore be expressed as slow features from such sensory signal. The goal of *Slow Feature Analysis* (SFA) is to find a function that generates slowly varying output signals for the given input, so that certain aspects of this output signal can be useful for a high-level representation.

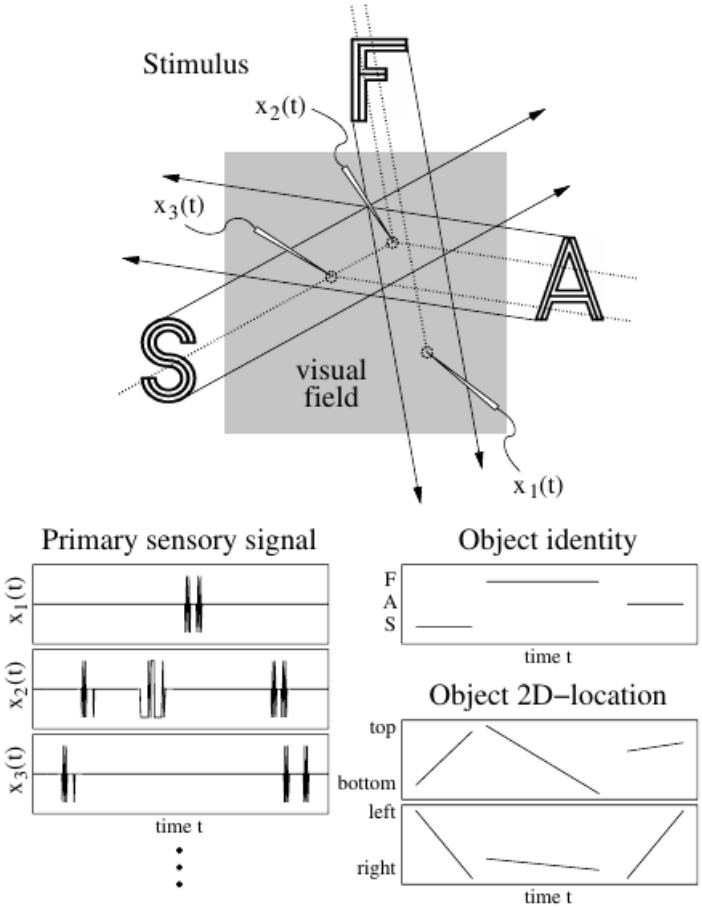

Figure 3: Relation between slowly varying stimulus and quickly varying sensor activities. High-level representation of the stimulus in terms of object identity and object location over time. From [6].

Mathematically, Slow Feature Analysis can be described as follows: given an I-dimensional input signal $\mathbf{x}(t) = [x_1(t), ..., x_I(t)]^T$, consider an input-output function $\mathbf{g(x)} = [g_1(\mathbf{x}), ..., g_J(\mathbf{x})]^T$, each component of which is a weighted sum over a set of K nonlinear functions $h_k(\mathbf{x})$: $g_j(\mathbf{x}) := \sum_{k=1}^{K} w_{jk} h_k(\mathbf{x})$. Usually $K > \max(I, J)$. Applying $bfh = [h_1, ...h_K]^T$ to the input signal yields the nonliearly expanded signal $\mathbf{z}(t) := \mathbf{h}(\mathbf{x}))$. Thus, the problem is now linear. The weight vectors are subject to learning, and the jth output signal component is given by $y_j(t) = g_j(\mathbf{x}(t)) = \mathbf{w}_j^T \mathbf{h}(\mathbf{x}(t)) = \mathbf{w}_j^T \mathbf{z}(t)$. Therefore, the function to be minimized by optimizing the weights is

$$\Delta(y_j) = \langle \dot{y}_j^2 \rangle = \mathbf{w}_j^T \langle \dot{\mathbf{z}} \dot{\mathbf{z}}^T \rangle \mathbf{w}_j, \tag{4.3.1}$$

where the angle brackets indicate temporal averaging, that is,

$$\langle f \rangle := \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} f(t) dt. \tag{4.3.2}$$

Moreover, the minimization must be performed under the constraints

$$\langle y_j \rangle = \mathbf{w}_j^T \langle z \rangle = 0 \qquad \text{(zero mean)}, \tag{4.3.3}$$

$$\langle y_j^2 \rangle = \mathbf{w}_j^T \langle z z^T \rangle \mathbf{w}_j = 1 \qquad \text{(unit variance)}, \tag{4.3.4}$$

$$\forall j' < j : \ \langle y_{j'}, y_j \rangle = \mathbf{w}_{j'}^T \langle z z^T \rangle \mathbf{w}_j = 0 \qquad \text{(decorrelation)}. \tag{4.3.5}$$

The combination of the first two constraints avoids the trivial solution where $y_j(t) = const.$. Note that (4.3.4) will not achieve unit variance without (4.3.3) as the solution $y_j(t) = 1$ would still be possible. (4.3.5) guarantees both that the output components carry different information and a strict order of optimality in the components such that $\Delta(y_{j'}) \leq \Delta(y_j)$ if $j' < j$. The motivation for this constraints is that the goal of SFA is to find features that vary over time and are uncorrelated, so constant features must be avoided. Consequently, the target function will be able to select those features which are non-trivial and present a low temporal-variation rate, that is, the *slowest* features. A better understanding can be obtained by analyzing the effect these expressions have on a simple time-varying signal $x(t) = t$. The solution will constitute a Fourier basis[8] [18] with the form $\sin(kx)$ or $\cos(kx)$ for $k \in \mathbb{Z}^+$.

All three constraints will be automatically fulfilled if h_k are chosen such that $\mathbf{z}(t)$ has zero mean and a unit covariance matrix; and also if and only if the weight vectors are constrained to be an orthonormal set.

Therefore, for the ith component of the input-output function the optimization problem can be expressed as finding the normed weight vector that minimizes $\Delta(y_i)$. The solution is the normed eigenvector of matrix $\langle \dot{\mathbf{z}} \dot{\mathbf{z}}^T \rangle$ that corresponds to the smallest eigenvalue [49]. The eigenvectors of the next higher eigenvalues produce the next components of the input-output function with the next higher Δ values.

Note that the input signals are normalized to zero mean and unit variance, a process which will be exact for training data. However, for test data only an approximate normalization can be produced due to the variability of the samples, which will have slightly different means and variances, while the offset and factor for the normalization are constant and obtained from the training data.

In practice, an approximation via whitening was performed in order to translate the aforementioned equations into a valid training method for convolutional neural networks such that the constraints are fulfilled. This adaptation will be described in detail in Section 5.3. The main difference resides in the fact that, while $\mathbf{z}$ values are considered to be constant for Slow Feature Analysis as explained above, this is not the case when backpropagating the function $\mathbf{h}$ in a neural network. Instead, an optimal value is sought.

4.4 Greedy InfoMax

This project is mostly based in the article *Putting An End to End-to-End Gradient-Isolated Learning of Representations* [7], where Löwe et al. describe *Greedy InfoMax* (GIM), a novel approach for Unsupervised Learning of unlabeled data. The idea and experiments were originally the core of Löwe's Master Thesis [50], but was later published as a paper and defended in the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019) at Vancouver, Canada. This proposal advocates for local information preservation. The authors propose an alternative to the omnipresent end-to-end backpropagation technique for Deep Learning by means of a modular neural network, where each module is trained individually and implements local backpropagation via a within-module loss function based on a *InfoNCE* [8]. As previously stated, the modules can be composed of one or several convolutional layers. The output is then gradient-detached and can be stored, allowing for an asynchronous optimization of the modules. The presence of slow features in natural data as described in the previous section is one of the axioms for GIM.

One of the main arguments in favor of this method is the biological discordance of global backpropagation and brain connections for optimization [51], being modularity based on local information a more realistic approach [52]. Biological synapses are mostly adjusted based on local information, that is, their immediate neighboring neurons, without having to wait for a global signal. Different areas of the brain can learn highly asynchronously and in parallel.

8 *The Fourier Basis.* Brown University, Providence RI. Source

However, while actual neurology can be seen once again as inspiration for Deep Learning, Greedy InfoMax also has algorithmic and mathematical motivations. Avoiding end-to-end back-propagation and not requiring labeled inputs are great advantages that can save computational and temporal resources. Local losses are backpropagated within each individual module, allowing for the whole process to be much faster and require less memory to store intermediate values. This loss is inspired by the log-bilinear model used by Öord et al. in their *Contrastive Predictive Coding* (CPC) work [26], known as *InfoNCE*:

$$\mathcal{L}_N = -\sum_k \mathop{\mathbb{E}}_X \left[\log \frac{\exp\left(z_{t+k}{}^T W_k c_t\right)}{\sum_{x_j \in X} \exp\left(z_j^T W_k c_t\right)} \right]. \tag{4.4.1}$$

This is an expectation over the samples in a set $X = \{x_1, \ldots, x_N\}$ of N random sample containing one positive sample from $p\left(x_{t+k}|c_t\right)$ and $N-1$ samples from $p\left(x_{t+k}\right)$. Where a sequence of observations x_t is encoded as $z_t = g_{enc}(x_t)$ and x_{t+k} are its future observations, modelled a the density ratio

$$f_k(x_{t+k}, c_t) \propto \frac{p(x_{t+k}|c_t)}{p(x_{t+k})}, \tag{4.4.2}$$

being $f_k(x_{t+k}, c_t) = \exp\left(z_{t+k}^T W_k c_t\right)$ a log-bilinear model used as a positive scoring function, where W_k is a linear transformation matrix.

In order to explain the relevance of InfoNCE and its application in Greedy InfoMax, let us first define the concept of *mutual information*, which refers to the information shared between the data x and its context c as

$$I(x; c) = \sum_{x,c} p(x, c) \log \frac{p(x|c)}{p(x)}. \tag{4.4.3}$$

When using Greedy InfoMax, an input signal x is encoded at a given instant of time t, producing $g_{enc}(x_t) = z_t$. For each module, the contextual information $g_{enc}(x_t) = c_t$ is a representation of the aggregated patches up to time-step t, that is, z_t^m. Each z_{t+k} is the representation of such encoding for a given delay k. These representations are considered "positive" samples while the other $N-1$ samples z_{j_n} are drawn from a bag $X = z_{t+k}, z_{j_1}, z_{j_2}, \ldots, z_{j_{N-1}}$ existing for each delay k and are considered as "negative" samples, following the negative sampling methodology of NCE (Noise Contrastive Estimation) [8, 9]. The goal is to maximize the lower bound of nearby patches representations, such that the so-called *future* representation, a neighboring encoding z_{t+k}^m, keeps as much information from z_t^m as possible. First, using the previous equation we redefine the mutual information as

$$I(z_{t+k}^m; z_t^m) = \sum_{z_{t+k}^m, z_t^m} p(z_{t+k}^m, z_t^m) \log \frac{p(z_{t+k}^m|z_t^m)}{p(z_{t+k}^m)}. \tag{4.4.4}$$

In the same way, it is also desired to maximize the mutual information between the future input of a module and its current representation $I(z_{t+k}^{m-1}; z_t^m)$. It is considered that, in some cases, a broader context might be necessary and the authors propose the addition of an optionally appended autoregressive module. However, this module was not included in the experiments of this project since the chosen tasks did not seem to require so much contextual information.

Before proceeding any further, there are a couple of assumptions that must be clarified. Both CPC and GIM are based on the estimations of the form described in [8], that is,

$$p(y|x; \theta) = \frac{\exp\left(s(x, y; \theta)\right)}{Z\left(x; \theta\right)}; \tag{4.4.5}$$

where $s(x, y; \theta)$ is an unnormalized score for a given label y and an input x under parameters θ. The denominator is a partition function for a given input x and a set of labels $\mathcal{Y}$ under θ such that $Z\left(x; \theta\right) = \sum_{y \in \mathcal{Y}} \exp\left(s(x, y; \theta)\right)$. This denominator is considered to operate a normalization

constant as long as it is finite. In the case of Greedy InfoMax —omitting the parameters for an easier notation—, and adapting the score function previously defined for InfoNCE:

$$p(z_{t+k}^m|z_t^m) = \frac{\exp\left({z_{t+k}^m}^T W_k z_t^m\right)}{Z\left(z_t^m;\theta\right)}; \qquad Z\left(z_t^m;\theta\right) = \sum_{z_t^m \in X} \exp\left({z_{t+k}^m}^T W_k z_t^m\right), \tag{4.4.6}$$

where $Z\left(z_t^m\right) = \sum_{z_t^m \in X} \exp\left({z_{t+k}^m}^T W_k z_t^m\right)$, being X a bag containing N contrastive samples. In a similar way to CPC, GIM assume a proportionality between the log-bilinear score function and a density ratio to preserve the mutual information such that

$$\exp\left({z_{t+k}^m}^T W_k z_t^m\right) \propto \frac{p(z_{t+k}^m|z_t^m)}{p(z_{t+k}^m)}. \tag{4.4.7}$$

Now, under the assumption that the input sequence is time-stationary and therefore $p(z_{t+k}^m) = p(z_t^m)$; and assuming a finite set of samples, an unbiased estimate of $p\left(z_t^m\right)$ can be considered as

$$p(z_{t+k}^m) = \sum_{z_t^m \in X} p(z_{t+k}^m|z_t^m)p(z_t^m) \approx \frac{\sum_{z_t^m \in X} p(z_{t+k}^m|z_t^m)}{N} \tag{4.4.8}$$

Then, (4.4.4) can be rewritten and then reduced using (4.4.6) and (4.4.8) as follows:

$$\begin{aligned}
I(z_{t+k}^m; z_t^m) &= \sum_{z_{t+k}^m, z_t^m} \log \frac{p(z_{t+k}^m|z_t^m)}{p(z_{t+k}^m)}, \\
&= \sum_{z_{t+k}^m, z_t^m} \log \frac{\exp\left({z_{t+k}^m}^T W_k z_t^m\right)}{Z\left(z_t^m;\theta\right)} : \frac{\sum_{z_t^m \in X} p(z_{t+k}^m|z_t^m)}{N} \\
&= \sum_{z_{t+k}^m, z_t^m} \log \frac{\exp\left({z_{t+k}^m}^T W_k z_t^m\right)}{Z\left(z_t^m;\theta\right)} : \frac{\sum_{z_t^m \in X} \frac{\exp\left({z_{t+k}^m}^T W_k z_t^m\right)}{Z(z_t^m;\theta)}}{N}, \\
&= \sum_{z_{t+k}^m, z_t^m} \log \frac{\exp\left({z_{t+k}^m}^T W_k z_t^m\right)}{\sum_{z_t^m \in X} \exp\left({z_{t+k}^m}^T W_k z_t^m\right)} + \log(N), \\
&= \mathop{\mathbb{E}}_{z_{t+k}^m, z_t^m \in X} \left[\log \frac{\exp\left({z_{t+k}^m}^T W_k z_t^m\right)}{\sum_{z_t^m \in X} \exp\left({z_{t+k}^m}^T W_k z_t^m\right)}\right] + \log(N).
\end{aligned} \tag{4.4.9}$$

In the Appendix A.1 of [8], it is proved how $I(xt+k, c_t) \geq \log(N) - \mathcal{L}_N^{opt}$ for the optimal loss. This is also the case of Greedy InfoMax since $L_N \approx -I(z_{t+k}^m; z_t^m) + \log(N)$ and therefore Greedy InfoMax can be expressed as a per-module variant of (4.4.1), where each module m performs the encoding $z_t^m = g_{enc}^m(GradientBlock(z_t^{m-1}))$, being $GradientBlock$ a gradient blocking operator applied in between modules to guarantee the absence of a global backpropagation. The custom module-local InfoNCE is then given by using (4.4.6) to minimize the mutual information such that, for a delay k,

$$\begin{aligned}
\mathcal{L}_{N_k} &= -\left(\mathop{\mathbb{E}}_{z_{t+k}^m, z_t^m \in X} \left[\log \frac{\exp\left({z_{t+k}^m}^T W_k z_t^m\right)}{\sum_{z_t^m \in X} \exp\left({z_{t+k}^m}^T W_k z_t^m\right)}\right] + \log(N)\right) + \log(N), \\
&= -\mathop{\mathbb{E}}_{z_{t+k}^m, z_t^m \in X} \left[\log \frac{\exp\left({z_{t+k}^m}^T W_k z_t^m\right)}{\sum_{z_t^m \in X} \exp\left({z_{t+k}^m}^T W_k z_t^m\right)}\right];
\end{aligned} \tag{4.4.10}$$

and finally

$$\mathcal{L}_N = -\sum_k \mathop{\mathbb{E}}_{z_{t+k}^m, z_t^m \in X} \left[\log \frac{\exp\left({z_{t+k}^m}^T W_k z_t^m\right)}{\sum_{z_t^m \in X} \exp\left({z_{t+k}^m}^T W_k z_t^m\right)}\right]. \tag{4.4.11}$$

The goal of this function is to measure the similarity between a sample x_t and its "future" or predicted patch x_{t+k}, as well as the dissimilarity with other patches from different patches (negative

samples). In other words, the mutual information $I(z^m_{t+k}, z^m_t)$ is maximized between nearby patch representations, aiming to extract *slow features* (described in the next section). The function basically compares the quality of a prediction of the future patch Z_{t+k} based on an observation of the current patch z_t with the quality of such prediction without having information about z_t. Therefore, the mutual information is measured in regard to the amount of information gained in z_{t+k} by observing z_t. Although similar to Slow Feature Analysis, this method could also easily predict fast changing signals, while SFA enforces slowness by means of MSE:

$$p(z^m_{t+k}|z^m_t) = \frac{\exp(-||z^m_{t+k} - z^m_t||)}{Z'} \tag{4.4.12}$$

where Z' is the standard normalizing constant of a normal distribution and and the constraints (4.3.3, 4.3.4, 4.3.5) must be fulfilled for z_{t+k}.

The first original experiment by Löwe et al. focused on image classification. A ResNet-50 v2 [53] model was split into three gradient-isolated modules and then fed with 16×16 patches of images. An accuracy of $81.9 \pm 0.3\%$ was obtained, the best result when compared with other methods such as the aforementioned CPC or some other supervised models. The memory consumption might not seem too low when accumulating all modules, but the key strength of this method is the possibility of training modules asynchronously. Unlike for classic, single-module architectures, only the backprop data for the module (layers) that is currently being trained needs to be loaded in the GPU. After training, the results are *frozen*, that is, stored locally to simply load a pre-trained module that the following modules can use. Therefore, the memory consumption *at a time* will always be bounded by the most demanding module, thus reducing simultaneous consumption by a factor of up to 2.8 in their experiment.

A second experiment was performed, this time in the audio domain. Both *speaker* (global task) and *phone* (phonetics of the words, local task) classifications were attempted, with quite different results: *speaker* classification had a great performance with a 99.4% accuracy, only falling behind of CPC (99.6%) but still ahead of the supervised alternatives; while *phone* classification obtained a 62.5% accuracy, lower than both CPC and the supervised models. This difference clearly indicates that Greedy InfoMax is a better solution for downstream tasks where a bias towards sequence-global feature extraction is desired.

5 Implementation

This section is devoted to the explanation of the implementation details of the project's code. Namely, the description of the preprocessing operations applied to both datasets, as well as the step-by-step code translation of the mathematical formulas for both Slow Feature Analysis and Greedy InfoMax, and the analysis of the model architectures will be complemented with Figures and pseudocode algorithm snippets to provide an insightful report on the programming specifics of the three experiments.

5.1 Preprocessing

Two image datasets have been employed in the experiments, both of which will be further explained in the next subsections. The first one is CIFAR-10, which is a collection of RGB images representing different classes of animals and vehicles; while the second one is the NORB-small dataset, comprising grayscale images of several toys with variations in the pose and lighting. In order for the network to properly handle the data, preprocessing techniques were applied to the images. Most of these techniques are quite common and can be found in most ML libraries and frameworks nowadays: resizing, normalization, center-cropping, etc. However, an extra step was taken by means of a custom patch selection function for each dataset.

5.1.1 CIFAR-10

CIFAR-10 is a labeled subset of the *80 million tiny images* dataset [54]. It consists of 6000 32×32 color images divided into 10 equally-sized, mutually exclusive classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships and trucks. These images were collected from the web by groups from MIT and NYU.

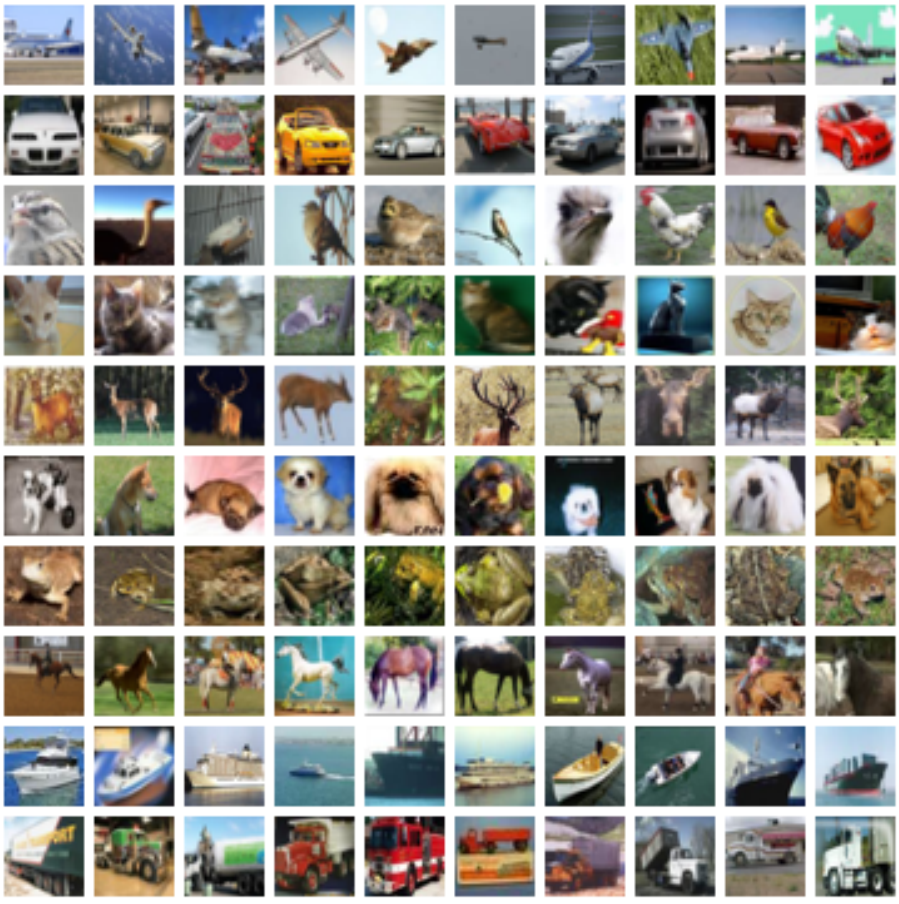

Figure 4: 10 random images from each category in the CIFAR-10 dataset, by rows. From CS Toronto.

Python's libraries `torch` and `torchvision` include predefined functions that allow easily downloading CIFAR-10 and creating data loaders for it. Inspired by Löwe's [7] vision experiment, a batch size of 16 was used and the images were reshaped to 64×64 pixels. Then, they were normalized and a two random 16×16 patches were extracted: the first one acts as the sample x_t, while the second one is the *future* sample or *prediction* x_{t+k}, extracted randomly among the four possible overlapping patches at a vertical or horizontal distance k, thus having an overlapping area of $16 \times (16 - k)$ pixels with the original patch. This is based on the intuition that natural data is ordered, and neighboring patches can be used as predictions of the current information.

Algorithm 1: Patch extraction for CIFAR-10

Input: sample image s, patch size p, distance k
Output: patch of the sample and its future patch

```
x, y ← random_coordinates(sample)          // Top-left corner of the patch
patch ← s[:, x:x+p, y:y+p]
do
    sign_x, sign_y← random_directions()    // Direction signs can be -1 or 1
    xf, yf ← x + k*sign_x, y + k*sign_y
while xf, yf are not valid coordinates
future ← s[:, xf:xf+p, yf:yf+p]

return patch, future
```

Note that, in the algorithm, a pair of coordinates is considered valid if it is within the image and a patch can be built within the image limits taking those coordinates as its top-left corner pixel. Consequently, no padding is required for patch extraction.

An example of the resulting patches can be observed in the next Figure, which depicts the extracted patches and original images for half of the samples in a random batch. The first and fourth columns show the original image and the following two columns show the current patch x_t and the future patch x_{t+k}, respectively. Again, note that the patch has been extracted after an enlargement of the original image to 64×64 and thus the relative sizes in the Figure are 1 : 16.

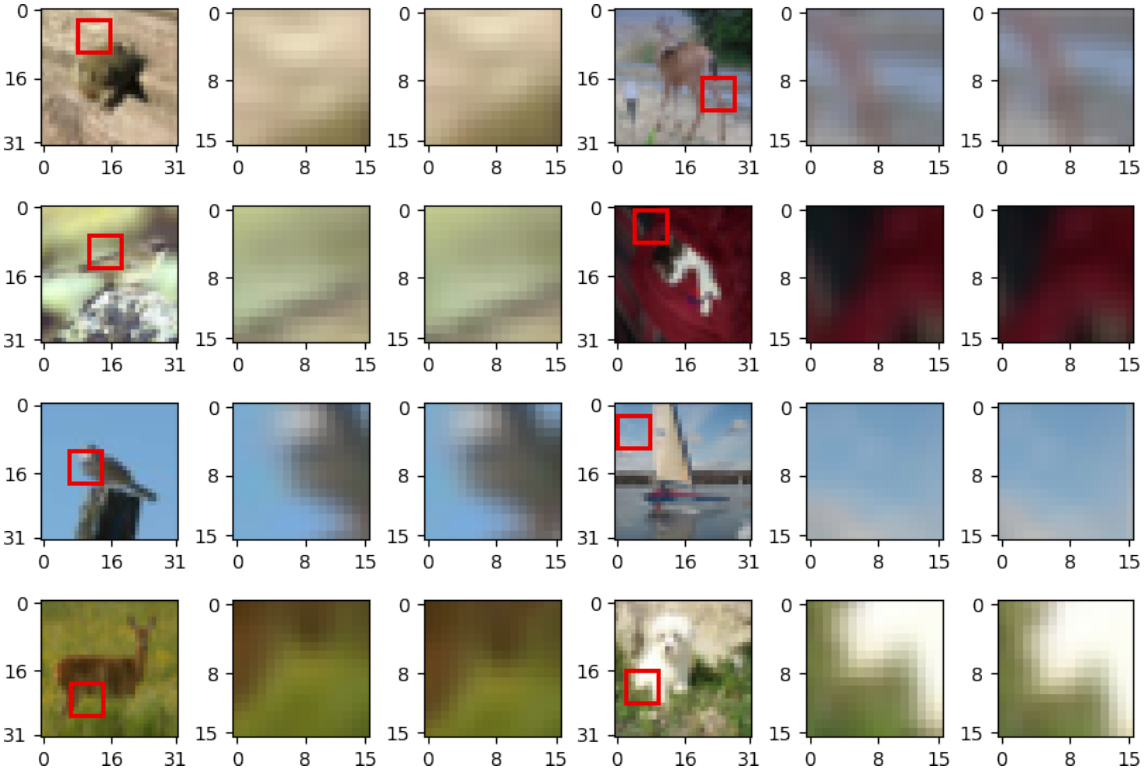

Figure 5: Patch extraction examples for CIFAR-10.

5.1.2 NORB-small

The NORB Dataset [30] comprises images of 50 uniform-colored toys belonging to 5 generic and mutually exclusive categories: four-legged animals, human figures, airplanes, trucks, and cars. For each category, there are 10 different instances (toy models); for each instance, there are several stereo image pairs with some variations. For the *small* version that is used in this experiment, they can vary under 18 different azimuths (0 to 340 degrees every 20 degrees), 9 elevation angles (30 to 70 degrees every 5 degrees) and 6 lighting conditions. In total, the dataset is formed by 48600 grayscale images. By design, five instances of each category (4, 6, 7, 8, 9) are designated for training, and the other five (0, 1, 2, 3 and 5) for testing.

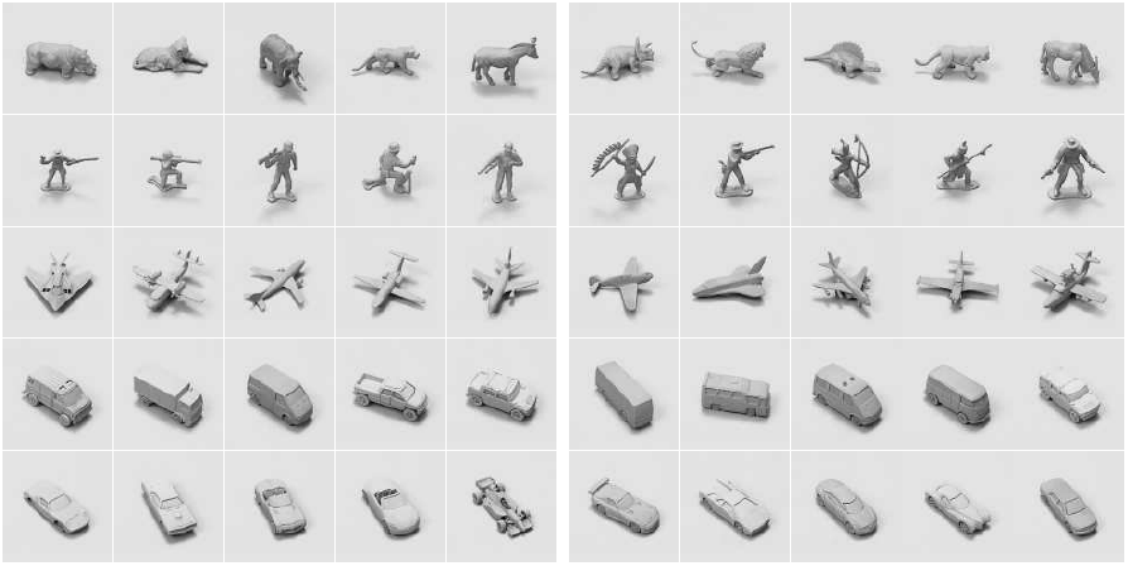

Figure 6: The 50 object instances in the NORB dataset. For each of the 5 categories (rows), the training instances are on the left side and the testing instances are on the right side. From [30].

The dataset is first loaded and grouped by category and instance thanks to Andrea Palazzi's auxiliary functions[9]. Then, both the normal and the grouped versions are saved via `pickle` in order to save time for upcoming executions. The loaders use `drop_last=True` so that the last batch is ignored when the number of samples is not divisible by the batch size (16).

Note that each feature ϕ is stored as a value $v_\phi \in [0, N_\phi - 1]$, where N_ϕ is the total amount of possible values for the feature. The real elevation angle ϵ can be computed as $\epsilon = 30 + 5v_\epsilon$, and the real azimuth α as $\alpha = 10v_\alpha$. In the example Figures, the lighting value is represented by l.

For the NORB classification experiment, the concept of patch was simplified to a simple 64×64 center crop of the original sample. The key difference is that in this case the *future* patch will be another image of the same toy instance, but it will have a different elevation and/or azimuth and/or lighting —at least one of them will be different from the sample's corresponding value. This selection is based on the the main aim of working with the NORB dataset, which is no other than finding a way to learn despite object pose and setting lighting. The positive samples will therefore be those representing the same toy, regardless of possible variations; while the negative samples will be those belonging to different toys.

9 Andrea Palazzi (2018). *Small NORB*. GitHub repository: *https://github.com/ndrplz/small_norb*

Algorithm 2: Patch extraction for NORB-small (classification)

Input: random sample image x_t, dataset d
Output: patch of the sample and its future patch

d ← sort_dataset(d) `// sort by class and instance`
candidates ← d[x_t.class][x_t.instance `// use only images of the same toy`
do
  x_{t+k} ← candidates[randrange(0, len(candidates)-1)] `// random pick`
while $x_t == x_{t+k}$ `// avoid using the same image`

return x_t, x_{t+k}

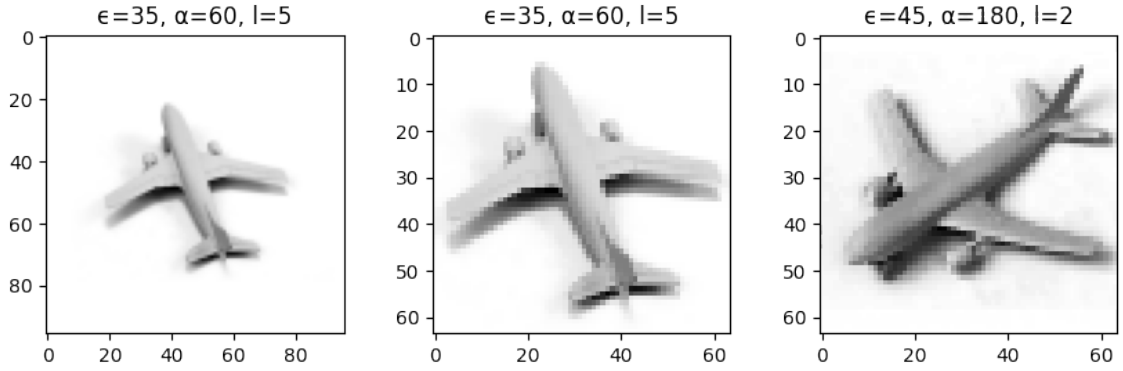

Figure 7: Patch extraction example for NORB-small (classification).

However, a different *modus operandi* was adopted for the feature experiment: the future patch will have either a immediately higher or lower (*neighboring*) azimuth or elevation. For instance, if a sample has an azimuth of 280 degrees and an elevation of 55 degrees, the future sample will have one of the following (*azimuth*, *elevation*) combinations: (280, 50), (280, 60), (260, 55), (300, 55). The lighting of the future patch can have any value, including the same as the original sample. This modification aims for a better detection of the angle variations as slow features.

Algorithm 3: Patch extraction for NORB-small (feature extraction)

Input: random sample image x_t, dataset d
Output: patch of the sample and its future patch

d ← sort_dataset(d) `// sort by class and instance`
candidates ← d[x_t.class][x_t.instance] `// use only images of the same toy`
do
  x_{t+k} ← candidates[randrange(0, len(candidates)-1)] `// random pick`
while *not_neighboring*(x_t, x_{t+k}) `// force neighboring angles`

return x_t, x_{t+k}

The *neighboring* condition is easy to check by applying Boolean comparisons to the parameters of both samples. The center cropping is performed by the dataloader after selecting the *patches*, along with normalization. One may argue that the term *patch* is not appropriate given this new approach, unlike with the initial CIFAR-10 algorithm. Nevertheless, the nomenclature was maintained in order to be consistent with the CIFAR experiment and the terminology used by Löwe et al. [7]. Note that in both algorithms x_t has been simplified as a single sample, while in the actual model this corresponds to a full batch of samples and the algorithm will be applied to each one of them.

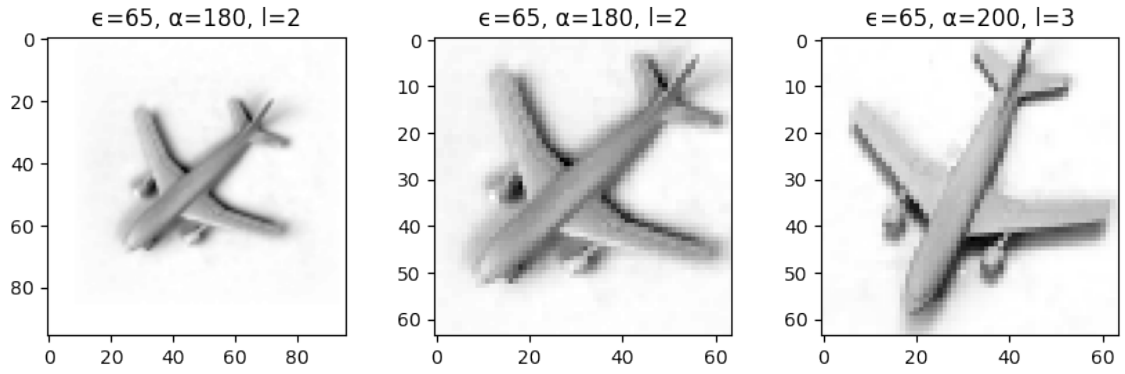

Figure 8: Patch extraction example for NORB-small (feature extraction).

The same toy plane has been used for both examples, so as to allow for an easier visual comparison of the results. Despite this, it is worth reminding that, while the patches extracted for classification use all the toys, the ones meant for feature extraction use only the images of this exact toy plane.

5.2 InfoNCE-based loss

An slightly simplified version of the InfoNCE-based loss from Greedy InfoMax was implemented and used for each module, relying on PyTorch's `backward()` method for backpropagation. For CIFAR-10, only a single value of k was considered —it was set to 2 for all executions—, while this parameter was not necessary for the CIFAR experiments since the patch pairs were based on a pose criteria and did not belong to the same image. The linearization matrix W_k was omitted in the implementation for simplicity since the last layer for the models is linear and acts as a replacement for a symmetric W_k, which is suitable for the experiments since a non-symmetric matrix would only allow rotations from z_t to z_{t+k} due to the polar decomposition $W = CU$, where C is a positive definite symmetric matrix and U is an orthogonal matrix. Since the patch pairs in the dataloaders are random and both a rotation from z_t to z_{t+k} and its inverse can occur, this type of transformation is undesired. Note that, Finally, the whole expression was adapted into a subtraction in order to avoid potential numerical errors coming from the division.

$$\mathcal{L}_N = -\log \frac{\exp\left({z^m_{t+k}}^T z^m_t\right)}{\sum\limits_{z^m_j \in X} \exp\left(z^{mT}_j z^m_t\right)} = -\left({z^m_{t+k}}^T z^m_t\right) + \log \sum_{z^m_j \in X} \exp\left(z^{mT}_j z^m_t\right). \tag{5.2.1}$$

It is stated in [7] that including the current patch in the bag of negative samples had no noticeable effects on the performance. For this reason, from this point on it will be assumed that $z^m_{t+k} \in X$ and all the future samples are used to for all patches in the batch and thus the generalized notation z^m_j.

Although not included in the early implementations, an auxiliary value $\zeta = \underset{z^m_j \in X}{max}(z^{mT}_j z^m_t)$ is used in order to normalize the values of the second term and avoid numerical anomalies in the log operation. After some initial experiments it became evident that a direct implementation of (5.2.1) was numerically unstable and the loss would often end up either becoming too small to fit in the range of values representable by floating-point variables due to the logarithmic operations or non-decreasing due to small values of the first term in comparison with the second.

$$\begin{aligned}
\mathcal{L}_N &= -\left({z^m_{t+k}}^T z^m_t\right) + \log \sum_{z^m_j \in X} \exp\left(z^{mT}_j z^m_t + \zeta - \zeta\right) \\
&= -\left({z^m_{t+k}}^T z^m_t\right) + \log \sum_{z^m_j \in X} \exp(\zeta) \exp\left(z^{mT}_j z^m_t - \zeta\right) \\
&= -\left({z^m_{t+k}}^T z^m_t\right) + \zeta + \log \sum_{z^m_j \in X} \exp\left(z^{mT}_j z^m_t - \zeta\right).
\end{aligned} \tag{5.2.2}$$

Now, in order to fully optimize the computation, all the data was vectorized to take advantage of the broadcast operations available in PyTorch. Loops are avoided by using multi-dimensional tensors for the data batches, which then are split into two matrices corresponding to all the *current* samples and its corresponding future samples in a per-column fashion for a given batch:

$$(z_u^m)' = \begin{pmatrix} s_{1,1}^m & s_{1,2}^m & s_{1,3}^m \\ s_{2,1}^m & s_{2,2}^m & s_{2,3}^m \\ \dots & \dots & \dots \\ s_{B,1}^m & s_{B,2}^m & s_{B,3}^m \end{pmatrix}; \qquad u \in \{t,\ t+k\}, \tag{5.2.3}$$

where $s_{i,j}^m$ represents the output value j for sample i in the batch. The loss per sample s is then computed as

$$N = \text{diag}\left((z_t^m)'^T (z_{t+k}^m)'\right), \tag{5.2.4}$$

$$\zeta' = \max_c\left((z_t^m)'^T (z_{t+k}^m)'\right), \tag{5.2.5}$$

$$D = \log_w\left(\text{sum}_c\left(\exp_w\left((z_t^m)'^T (z_{t+k}^m)' - \zeta'\right)\right)\right), \tag{5.2.6}$$

$$\mathcal{L}_{\mathcal{N}}(s_i) = (-N + \zeta' + D)_i, \tag{5.2.7}$$

where *diag* returns a vector containing the elements from the main diagonal of a matrix, max_c returns a vector after computing the column-wise maximum of a matrix, sum_c computes a vector resulting from a column-wise sum, log_w is the element-wise logarithm and exp_w is the element-wise exponential. Note that the additions and subtractions also take place in a broadcasting method, either element or column-wise, depending on the case. Note that the i subscript at (5.2.7) denotes the ith element of the vector, and also that $(z_{t+k}^m z_t^m)$ is computed in the opposite order due to the dimensions of the samples in the experiment. The final loss is an averaged sum of the loss per sample in every batch b of size B:

$$\mathcal{L}_N(b) = \frac{\text{sum}_c\left(-N + \zeta' + D\right)}{B} = \frac{\sum_{i=1}^{B} \mathcal{L}_N(s_i)}{B}. \tag{5.2.8}$$

When coding, the expression $(z_{t+k}^m z_t^m)$ is nicknamed `mul`. The first term, originally the numerator (`num`) is then computed as the values in the main diagonal of that matrix, thus providing one scalar value per patch within a given batch. The value of ζ is calculated per-column (per-sample) and then subtracted to `mul` and added to the logarithm of the sum to obtain the second term `den` (denominator). Since the result is a $1 \times B$ vector, an average value is computed to be the result for the loss of the whole batch (5.2.3).

Algorithm 4: InfoNCE-based custom loss function

Input: batch of processed samples *zt*, batch of processed future samples *ztk*, batch size B
Output: current loss value

```
mul ← matrix_multiplication(zt.transpose(), ztk)
num ← mul[diagonal_indices(B)]                    // Main diagonal
ζ ← max(mul, axis=0)                              // Per sample
den ← log(sum(exp(mul-ζ), axis=0))                // Per sample
return mean(-num + ζ + den)
```

5.3 Slow Feature Analysis

The Slow Feature Analysis is not focused on a custom loss function, but rather a combination of operations appended to the model's outputs. After the linear layer at the end of the network, a custom approximate whitening normalizer function is applied. Whitening is necessary so that the

data fulfills the SFA constraints of zero-mean (mean-free), having unit variance if projected onto any unit vector and having decorrelated projections onto orthonormal vectors.

An approximation to mean-free data is obtained by subtracting the mean from the values such that

$$x'_i = x_i - \frac{\sum_{i=1}^{N} x_i}{N}. \tag{5.3.1}$$

Then, a whitening matrix W must be found so that $y_i = Wx_i$ is a whitened version of x_i, that is, mean-free and with unit covariance. Note that having unit covariance matrix fulfills the last two constraints simultaneously. A covariance matrix $C = \frac{\sum_i^N x_i x_i^T}{N}$ can be applied an eigenvalue decomposition such that

$$C = UDU^T, \tag{5.3.2}$$

where U is a orthogonal matrix and D a positive diagonal matrix. In the paper by Wiskott et al. [22], the data is whitened and then projected onto the minor components of the difference time-series $\{\dot{x}_t = x_{t+1} - x_t\}_{t=0\ldots\tau-1}$. In order to do so, an algorithm performs the calculation of the eigenvalues and eigenvectors via power iteration, which is costly and poses a handicap on the differentiation process. To justify the alternative approach that was used in this project's implementation, let us first have a whitening with the form $y_i = D^{-1/2}U^T x_i$, whose covariance matrix C' can be reduced as follows:

$$\begin{aligned} C' &= \frac{\sum_i^N \left(D^{-1/2}U^T x_i\right)\left(D^{-1/2}U^T x_i\right)^T}{N} \\ &= D^{-1/2}U^T \frac{\sum_i^N x_i x_i^T}{N} (D^{-1/2}U^T)^T \\ &= D^{-1/2}U^T C U D^{-1/2} \\ &= D^{-1/2}U^T U D U^T U D^{-1/2} \\ &= D^{-1/2} D D^{-1/2} \\ &= I. \end{aligned} \tag{5.3.3}$$

First, $D^{-1/2}$ and U^T are non-dependent on i and therefore can be extracted from the sum so that the definition of C is applied and later substituted by (5.3.2). Note that, due to the nature of the matrices, $\left(D^{-1/2}U^T\right)^T = UD^{-1/2}$ since $(AB)^T = B^T A^T$ for any pair of matrices A and B, and $D^T = D$ because it is diagonal. Finally, the expression is reduced to the identity matrix I^{NxN} by means of the inherent properties of the orthogonal matrix U and the diagonal matrix D.

The power iteration computations can be avoided by employing different decompositions, such as the Cholesky Factorization[10] $C = LL^T$ with $L = UD^{1/2}E$, where E is chosen so that L is a lower triangular matrix. Therefore, the final transformation can be expressed as

$$y_i = L^{-1}x_i \tag{5.3.4}$$

since

$$\begin{aligned} C' &= \frac{\sum_i^N \left(L^{-1}x_i\right)\left(L^{-1}x_i\right)^T}{N} \\ &= L^{-1}\frac{\sum_i^N x_i x_i^T}{N}(L^{-1})^T \\ &= L^{-1}C(L^{-1})^T \\ &= L^{-1}LL^T(L^{-1})^T \\ &= I^2 \\ &= I. \end{aligned} \tag{5.3.5}$$

10 Cholesky Factorization. Reference: L. Vandenberghe (2019). *Applied Numerical Computing - Lecture 12*. UCLA. Source

Note that $\left(L^{-1}\right)^T = \left(L^T\right)^{-1}$. With this implementation, the simple triangular matrix can be solved to obtain the final 3-dimensional outputs for each sample in the batch. However, the Module must be watched so that the values are coherent, since L will only be invertible as long as there are no zero-values on its main diagonal.

Algorithm 5: Approximate whitening normalization

Input: network outputs y, batch size B, output dimension d
Output: whitened values z

```
m ← mean(y, dim=0)                                          // Per output dimension
y′ ← y - m                                                  // Per output dimension
C ← matrix_multiplication(y′.transpose(), y′) / (B - 1)     // Bessel's correction
C ← C + eye(d)                                              // Add identity matrix
L ← C.cholesky()                          // Cholesky decomposition, lower triangular
z ← triangular_solve(y′.transpose(), L, upper=False)        // Find system's solutions
return z.transpose()
```

Note that in this case z_t^m and z_{t+k}^m correspond to the outputs of the algorithm instead of being the direct outputs of the network. After this process, the loss is finally calculated as a mean squared error (MSE):

$$\mathcal{L}_N = mean\left(\left(z_t^m - z_{t+k}^m\right)^2\right) \tag{5.3.6}$$

5.4 Model architecture

Initially, a simple architecture based on the popular VGG-16 [55] architecture was built, since the network was intended to be small but effective. The early versions contained six convolutional layers producing up to 512 output features for a final fully-connected (linear) layer. These were meant to be divided into three modules of two layers each. However, the training process for this model was slow and the Colab usage limits were often exceeded and the whole process was slowed down since there is a cooldown of several hours before the usage count is restarted.

So as to address this issue, the network was reduced after testing and confirming that the main NORB-based experiments did not require such a high amount of features to obtain proper results for neither classification nor feature extraction. Thus, the number of convolutional layers was reduced to four and only two modules were included for the Multi-Module version. The amount of features was initially set to 128 but the results were almost identical as those obtained with 64, which was the final choice for the eventual choice for the definitive versions of the models. The kernel size was arbitrarily set to 5 for all layers, taking inspiration from [7], with a stride of 1 and a padding of 2. A flattening function is applied to the result of the last Max-Pool layer before the data is fed into the Linear layer, which outputs three-dimensional values so that they can be interpreted as coordinates and plotted to both visualize the class clusters in the classification tasks and making possible the study of the results for the feature extraction in a similar way to the graphs from [22] that were shown in Section 2.2. Note that the flattening has been omitted from the Tables since it was not implemented as a layer but rather as an auxiliary function.

Layer	Input Size	Output Size	Kernel Size
Conv2D	1x64x64	32x64x64	5x5
Conv2D	32x64x64	32x64x64	5x5
Max-Pool	32x64x64	32x32x32	2x2
Conv2D	32x32x32	64x32x32	5x5
Conv2D	64x32x32	64x32x32	5x5
Max-Pool	64x32x32	64x16x16	2x2
Max-Pool	64x16x16	64x8x8	2x2
Linear	4096	3	-

Table 1: Single-Module network architecture.

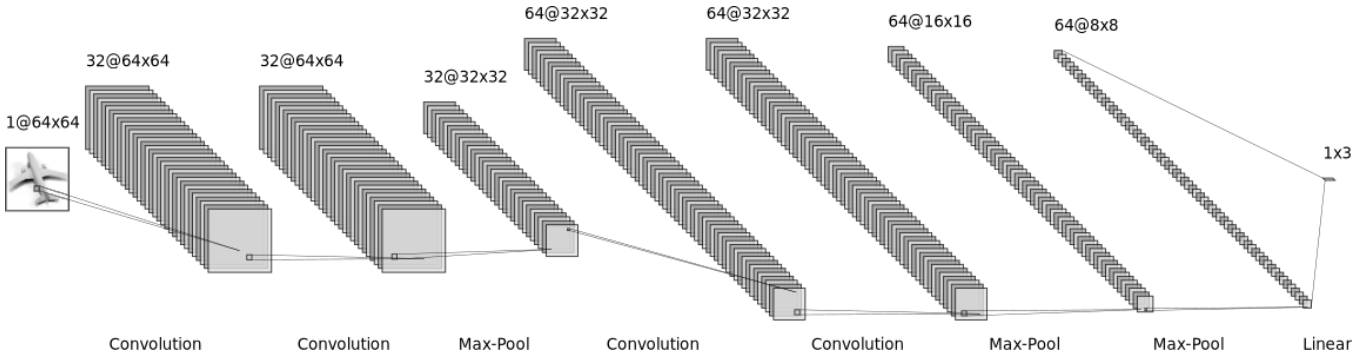

Figure 9: Layer architecture for the Single-Module model (NORB).

The conversion to a Multi-Module architecture was achieved by splitting the model after the first Max-Pool layer. The kernel size, stride and padding are the same as in the Single-Module architecture and in the experiments they were trained for the same amount of epochs so that the comparison would be as fair as possible. Again, the data was flattened in both modules before reaching the linear layer.

The first Module requires a linear layer of its own during training, but its weights are not stored since the training process for Module 2 plugs the output of the Max-Pool layer directly into its first convolutional layer. The training process can be summarized as:

1. Load an empty Module 1 and append a Linear layer to it.
2. Train Module 1 and store its weights, discarding the Linear layer.
3. Load the stored weights for Module 1 and an empty Module 2.
4. Pass the data through Module 1 and then use it as an input for Module 2.
5. Store Module 2 in its entirety for the testing phase.

Module 1			
Layer	**Input Size**	**Output Size**	**Kernel Size**
Conv2D	1x64x64	32x64x64	5x5
Conv2D	32x64x64	32x64x64	5x5
Max-Pool	32x64x64	32x32x32	2x2
Linear	32768	3	-
Module 2			
Layer	**Input Size**	**Output Size**	**Kernel Size**
Conv2D	32x32x32	64x32x32	5x5
Conv2D	64x32x32	64x32x32	5x5
Max-Pool	64x32x32	64x16x16	2x2
Max-Pool	64x16x16	64x8x8	2x2
Linear	4096	3	-

Table 2: Multi-Module network architecture

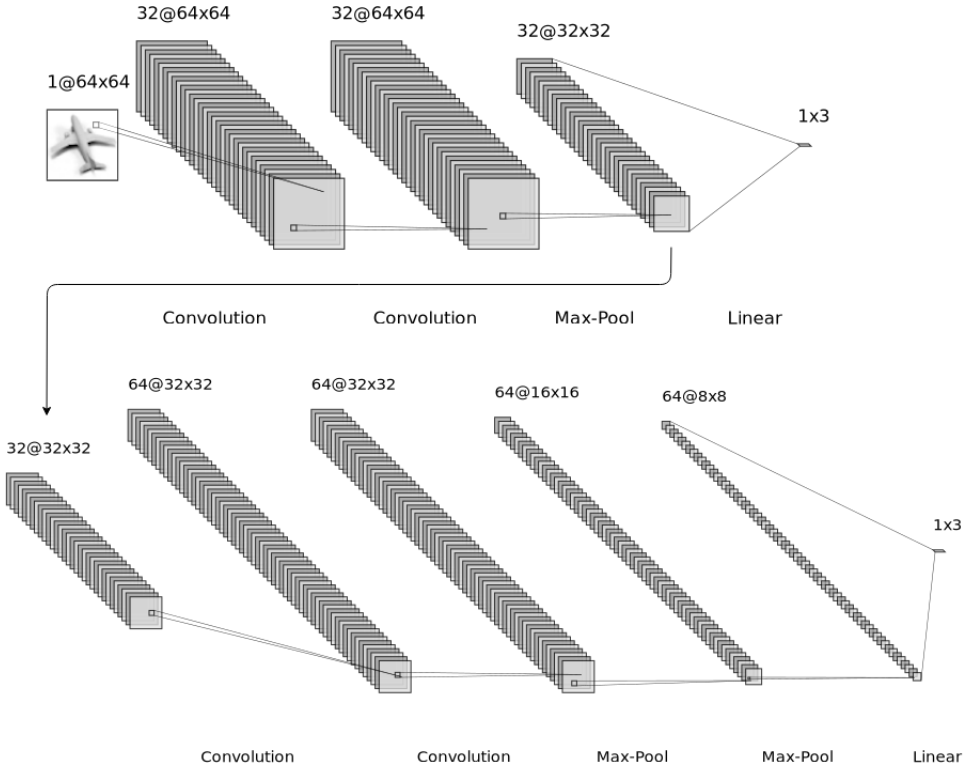

Figure 10: Layer architecture for the Multi-Module model (NORB).

Even if the Modules need to be trained in a pseudo-sequential fashion, it was practical to have them stored separately so that modifications on one module would not force repeating the training of the whole model. However, the execution times were lower for the classification task when dealing with two modules as each of them took the same amount of time to train than the Single-Module version, probably due to the big size of the dataset which dominated over the lack of layers, or perhaps because of Colab limits. Fortunately this was not the case for the feature extraction task, which was executed locally and the training time per individual module was lower than for the Single-Module version, although the total addition of the times was logically greater.

6 Experiments

This section will first present the first experiment based on a classification task with the CIFAR-10 dataset using the custom InfoNCE loss function, for both architectures. This will be followed by the two main experiments performed in the NORB dataset: classification with pose and lighting independence and pose feature extraction with lighting independence.

6.1 Classification of RGB images

As a first contact with Greedy InfoMax and PyTorch, a toy model was built in order to experiment with the custom loss and patch extraction. To this end, the CIFAR-10 dataset described in section 5.1.1 was used to train both the Single-Module and the Multi-Module versions of the network for a simple classification task. The decision of using CIFAR-10 was based on the fact that the original GIM experiment also operated with patches for RGB image classification and this specific dataset is easily accessible in PyTorch —in fact, it is currently being in their official tutorial page for neural networks. Löwe et al. [7] used the STL-10 dataset [56], which is similar to CIFAR-10 but uses a higher resolution for the images (96x96) and has less training examples in exchange for a large set of unlabeled test images, which makes it ideal for unsupervised experiments. The patch and batch sizes were set to 16 as in the original GIM experiment.

The purpose of this initial work was to serve as a blueprint for the later experiments. Many different architectures and combinations of parameters were tested during the first weeks, while configuring the custom loss. The results of the classification were always poor, due to the complexity of the task: the dataset was large and used RGB images with 3 information channels, there were many classes and the model was quite modest due to the hardware and Colab usage limitations. However, the accuracy was not so important in this case since the experiment was merely instructional. Eventually, when the final model was finished both models were trained for 300 epochs —for the Multi-Module architecture, when the number of epochs is mentioned in any experiment it is meant to be epochs per-module, that is, both modules are trained for the same amount of epochs— via batches of 16 patch pairs. The optimization was performed via PyTorch's in-built Adam optimizer, using the default parameters for the betas and epsilon while testing different learning rates. As per default in CIFAR-10, 5000 images of each class were used for the training set and the other 1000 were used for testing.

Initially, the classification was carried by a simple linear classifier, which was of course not enough for such a demanding task, so this method was replaced with a Nearest Centroid classifier from `slklearn.neighbors`, which significantly boosted the accuracy, specially for the NORB experiment.

There were no noticeable differences between both architectures in this experiment and variations in the number of epochs or learning rates didn't have a great impact. Although the trend might seem to be that of an increasing accuracy for smaller learning rates, values lower than $1e-5$ were observed to produce overfitting.

Learning Rate	Single (Train)	Single(Test)	Multi (Train)	Multi (Test)
1e-5	19.1%	19.4%	19.1%	19.3%
1e-7	17.2%	17.3%	17.6%	17.8%
1e-9	13.1%	13.5%	16.2%	16.3%

Table 3: Global accuracy of the models for the RGB classification experiment.

Although they might seem low, these accuracy values are an improvement with respect to the first versions of the model where no more than 13% was achieved, even with a deeper network. In spite of this improvement, the results above are averaged and the variation in global accuracy had strong variations depending on the execution, ranging between $14-21\%$. The accuracy per class also suffered from this inaccuracy, and it was often not much better than a random assignment.

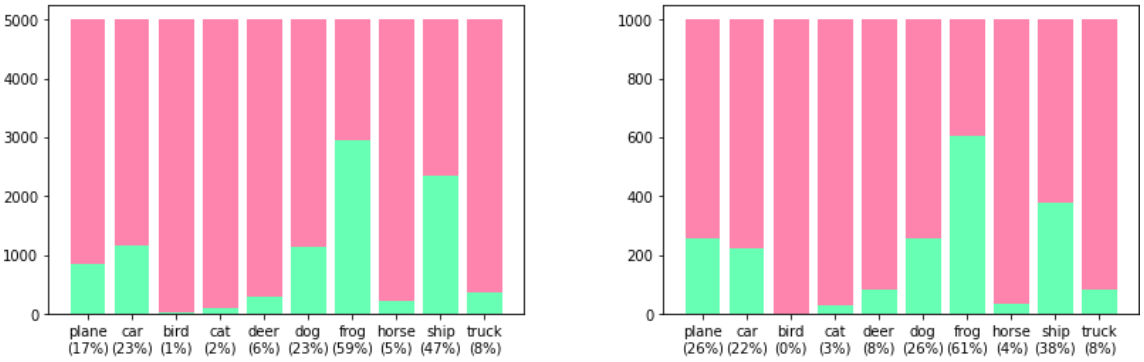

Figure 11: Train (left) and test (right) accuracy per class for the Single-Module architecture (CIFAR).

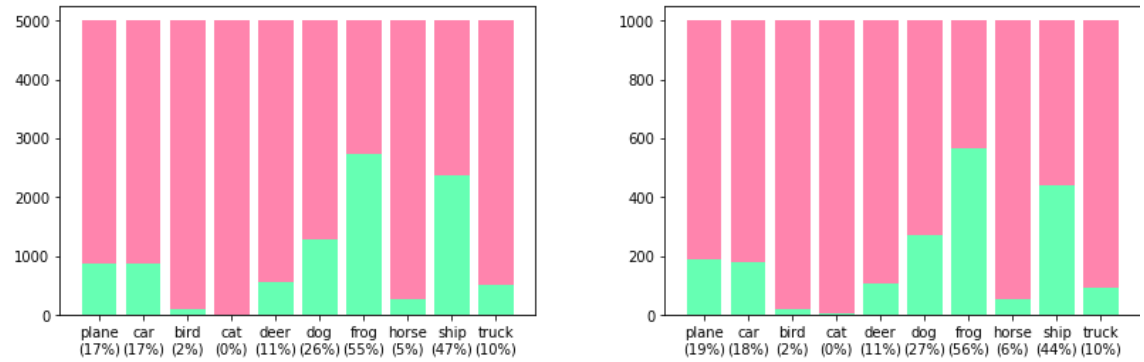

Figure 12: Train (left) and test (right) accuracy per class for the Single-Module architecture (CIFAR).

The point clouds per class can be observed in the next two Figures, which show the completely chaotic outcome. No clear clusters can be distinguished and most data is mixed and overlapping. Even if three features were good for visualization as it will be seen in the next sections, it might be too low for the given dataset. In the next section, the models prove their potential with much better results with a different dataset. Those results, along with the competitive outputs obtained by Löwe et al. [7] demonstrate that the approach is valid for larger learning tasks, given a different configuration of the models more targeted for this specific problem.

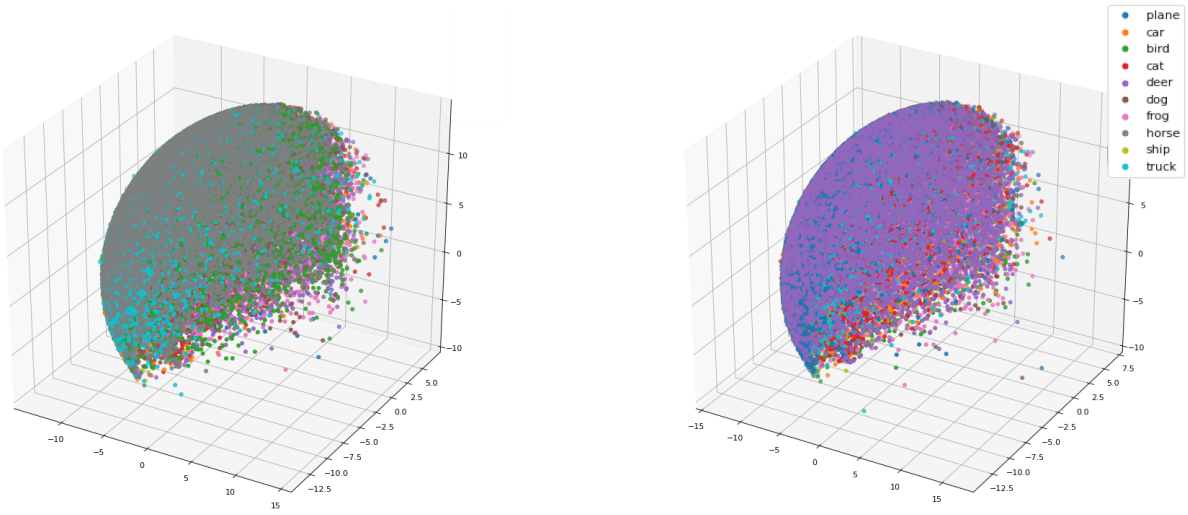

Figure 13: Train (left) and test (right) accuracy per class for the Single-Module architecture (CIFAR).

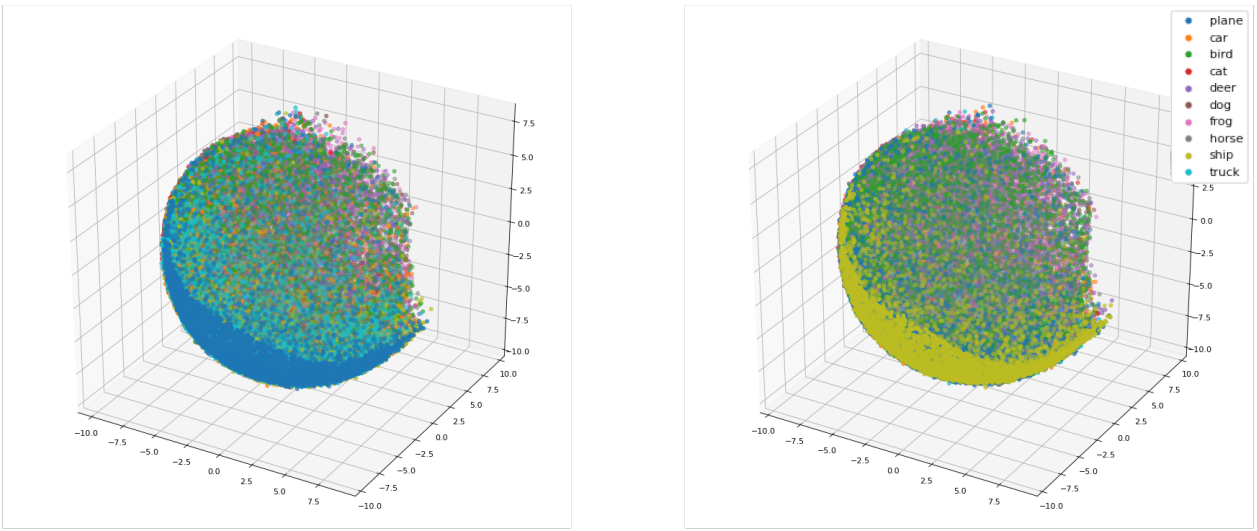

Figure 14: Train (left) and test (right) accuracy per class for the Single-Module architecture (CIFAR).

Note that the general orientation is random for every execution, but the overall structure and distribution is always similar. The colors for these plots might also be misleading since the classes are interleaved, rendering the nearest centroid classification almost useless.

6.2 Pose-and-lighting-invariant classification of grayscale images

The main experiments were performed on the NORB-small dataset (see section 5.1.2). For the classification task, image pairs produced with the *Algorithm 2* were fed into the models in batches of size 32. Initially, the size was set as 16, inspired by the Vision experiment in [7]. However, as explained in Section 5.3, Slow Feature Analysis would perform better with 32 elements and, after some tests, Greedy InfoMax showed the same performance with both amounts so 32 was used as a global parameter for all NORB experiments.

For Greedy InfoMax, the models were trained for 300 epochs and the learning rate for Adam that obtained the highest accuracy for both architectures was $1e-4$. The same amount of epochs were used in the Slow Feature Analysis versions, with the best learning rates also being approximately in the range between $1e-4$ and $1e-5$.

After training, classification was once again done via Nearest Centroid, which in this case proved to be much more successful due to the lower amount of classes and the overall better performance with the dataset. The output features were much more clustered in space than in the initial experiment and allowed for testing accuracy values over 80% for all four models.

Method	Learning Rate	Single (Train)	Single (Test)	Multi (Train)	Multi (Test)
GIM	1e-3	93.2%	79.3%	93.5%	81.2%
	1e-4	94.3%	88.7%	99.5%	82.9%
	1e-5	91.9%	81.7%	93.2%	80.9%
SFA	1e-3	92.7%	82.8%	94.9%	75.3%
	1e-4	93.4%	84.9%	92.9%	82.3%
	1e-5	93.3%	76.3%	95.4%	83.6%
	1e-6	90.8%	78.5%	90.5%	78.2%

Table 4: Global accuracy of the models for the NORB classification experiment.

The best performing model was the Single-Module, Greedy InfoMax version, with almost a 4% difference with respect to its Slow Feature counterpart and over a 5% improvement when compared to the best results obtained by the Multi-Module models. Again, all accuracies in the table were averaged, but the variance among executions was much smaller (about $\pm 0.4\%$) than in the CIFAR

experiment. Note how, in most cases, the accuracy for the training set is always high even for lower values of the testing accuracy, implying that a cerain degree of overfitting probably took place in those executions.

In most cases, the sequential architecture displayed a better performance than the modular version, although this highly depends on the learning rate —the same learning rate was always used for training both of the modules— and the difference is usually small. On the one hand, this can be considered as a failure since Greedy InfoMax was expected to seize the greedy optimization of information while giving up global backpropagation. On the other hand, the original experiments by Löwe et al. involved more modules and a deeper network that could potentially take further advantage of this approach. In any case, the results demonstrate that GIM is indeed comparable to Slow Feature Analysis, even showing better accuracy values for the same task.

These results show how a different dataset can radically change the performance of a model, even for a similar task. The lower amount of channels, images and classes allowed for a boost of over 60% with respect to the initial experiment. Nonetheless, the models were not flawless and, as it can be seen on Figure 16, the per-class accuracy was not always balanced. In many executions, objects belonging to the "car" class were often mistaken as "truck" and even if the overall accuracy was high, it was at the cost of misidentifying one of the toys.

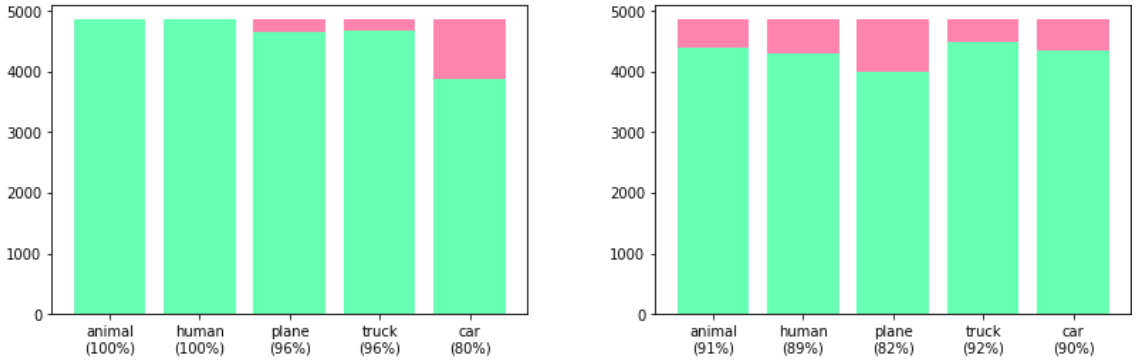

Figure 15: Train (left) and test (right) output visualization for the CIFAR experiment with the Single-Module architecture.

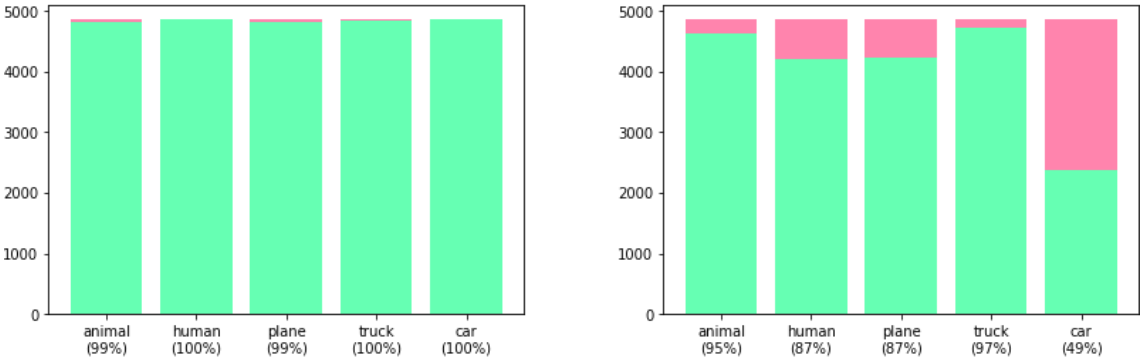

Figure 16: Train (left) and test (right) output visualization for the CIFAR experiment with the Multi-Module architecture.

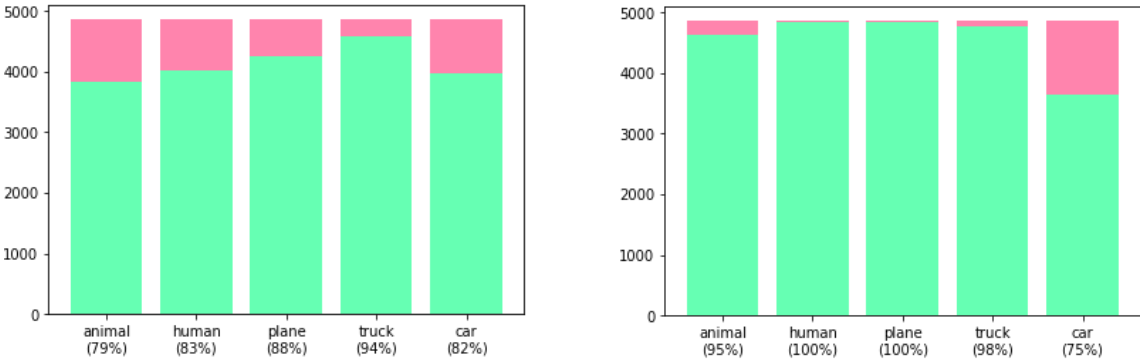

Figure 17: Train (left) and test (right) accuracy per class for the Single-Module architecture (SFA).

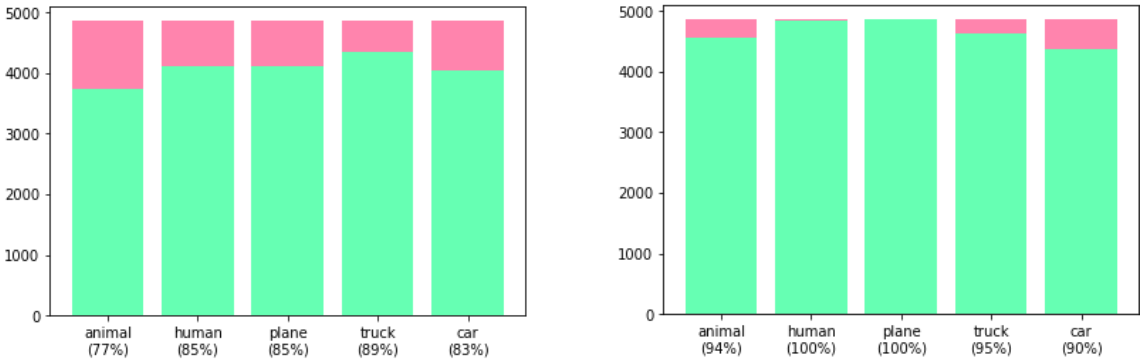

Figure 18: Train (left) and test (right) accuracy per class for the Multi-Module architecture (SFA).

The linear layer was purposely set to output three values so that they could be plotted in order to visualize the classification. Figures 19 and 20 can be found on the following two pages. They show the position in space for each sample after using the output of the model as a coordinate. The plots for the CIFAR-10 experiment showed its poor output, with the high amount of misclassification resulting in an almost meaningless depiction. That is not the case for the NORB experiment, since the plots perfectly show the clusters belonging to each class and, in the case of the Greedy InfoMax training models, even for each individual toy. This behavior might be seen as overfitting, but the best performing model when testing was in fact the one were the individual object clusters show a greater separation among themselves, implying that this kind of output does not necessarily imply that the network has overlearned.

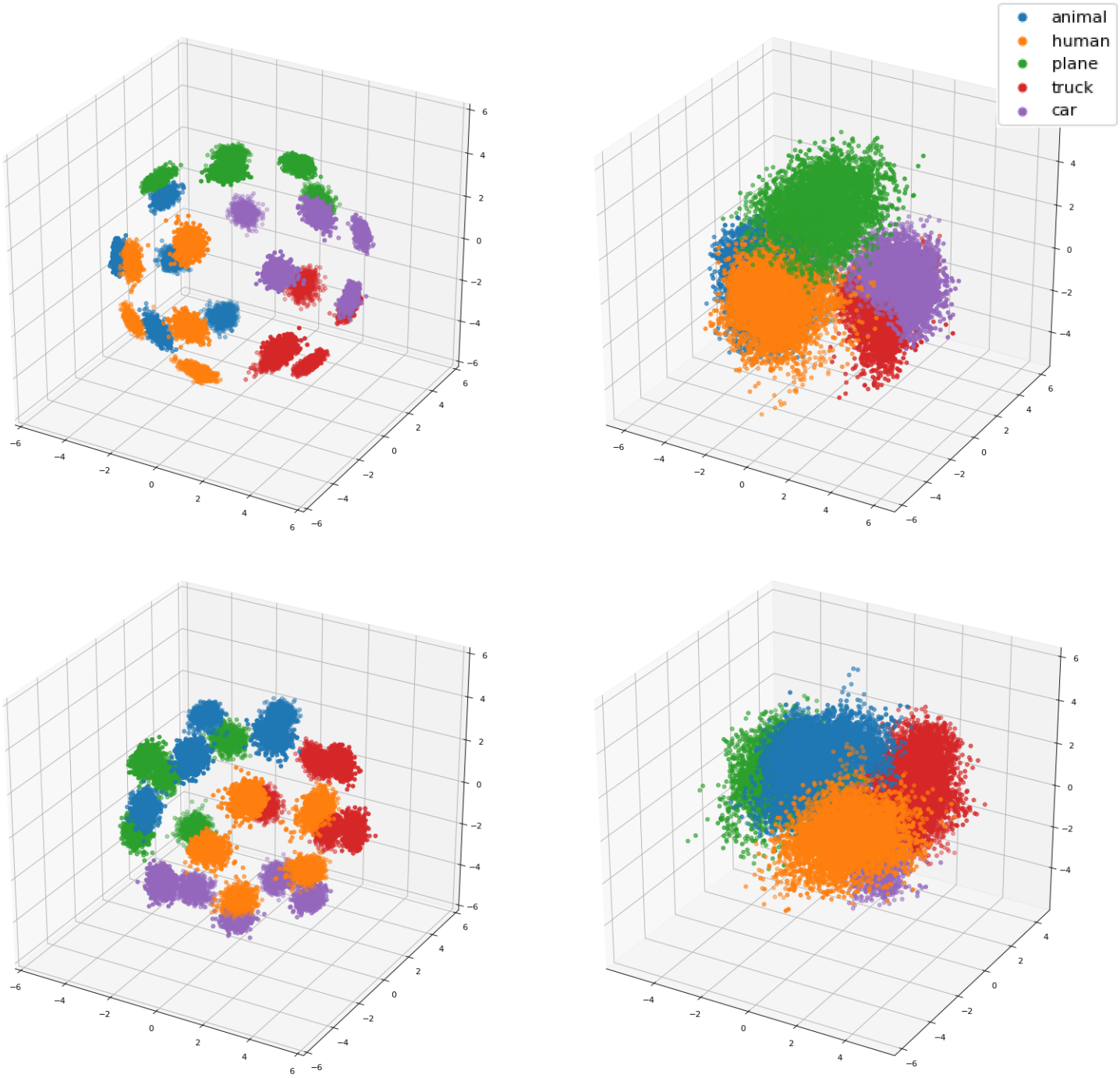

Figure 19: Output visualization per class for the Greedy InfoMax versions of the Single-Module architecture (top) and the Multi-Module architecture (bottom). The plots on the left depict the outputs of the training set, while those on the right side belong to the testing set.

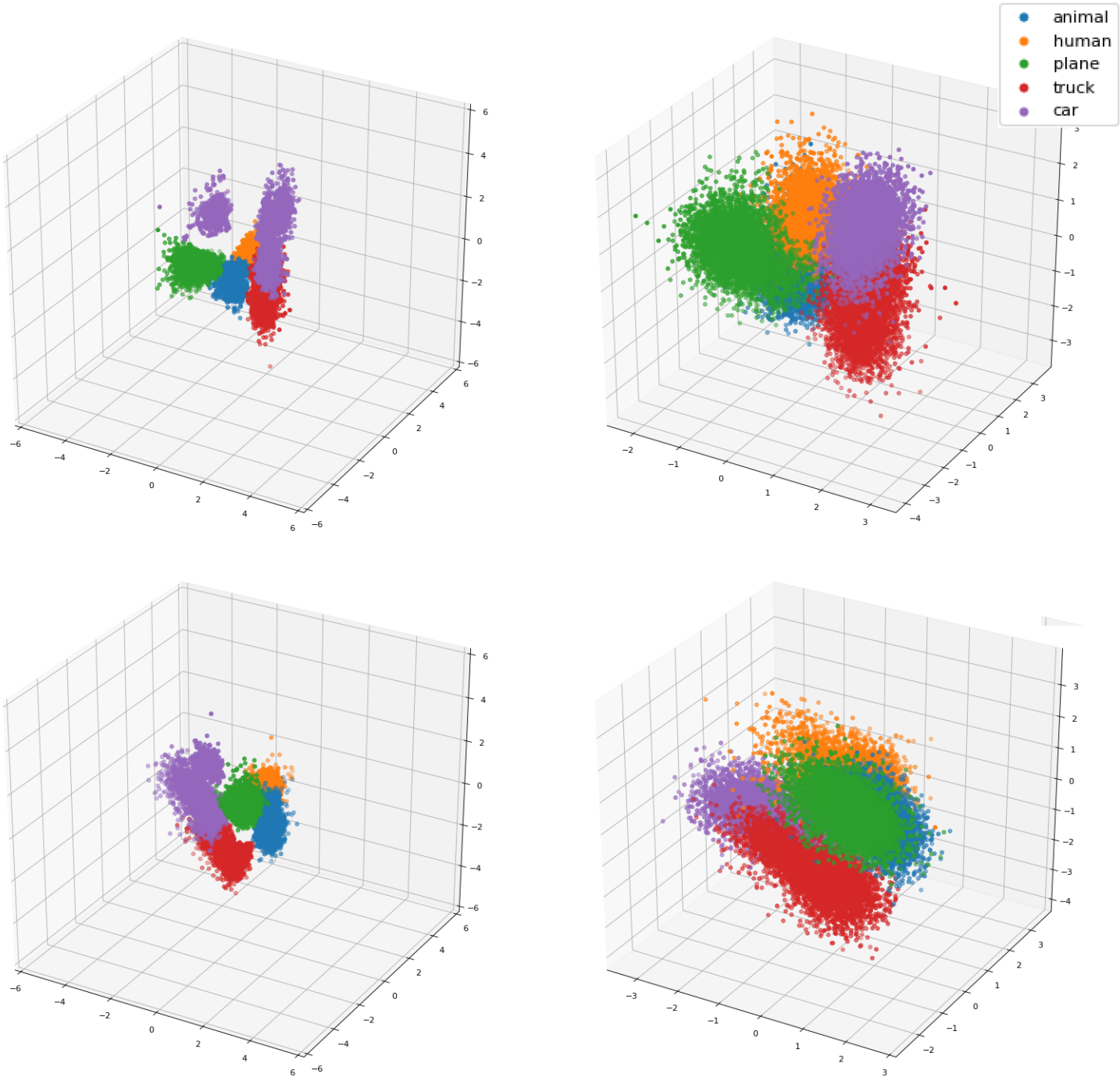

Figure 20: Output visualization per class for the Slow Feature Analysis versions of the Single-Module architecture (top) and the Multi-Module architecture (bottom). The plots on the left depict the outputs of the training set, while those on the right side belong to the testing set.

6.3 Lighting-invariant pose feature extraction on grayscale images

The last entry belongs to, perhaps, the most interesting experiment. The objective was for the network to learn the variations in the pose (azimuth and elevation angles) of a single toy. The idea comes from the NORB experiment by Wiskott et al. [22], where a color map was applied to the output points of a model trained with Slow Feature Analysis. Their plots can be seen in Section 2.2. Since the models are based on Unsupervised Learning, extracting meaning out of the outputs is a task of which that the analyst must take charge. The intelligent mapping from the aforementioned paper achieves exactly this goal: the fact that the network has learnt to detect those pose variations that it was given, and they can be visualized in a meaningful way. In a similar fashion, the models were fed with image patches based on *Algorithm 3*, that is, belonging to a single toy and having only a neighboring difference in either their azimuth or elevation parameters. The results can be seen in Figures 22 to 25, which show the outputs with a similar color mapping depending on the elevation and azimuth of the samples for all 4 variations: Single-Module and Multi-Module, each with both Greedy InfoMax and Slow Feature Analysis.

This experiment was much lighter since only one object was used for both training and testing, and therefore it was possible to train the model locally with a 3GB NVIDIA GTX 1650 card in parallel with the remote Colab executions. For Greedy InfoMax, the models were trained for 400 epochs since longer training sessions didn't display any significant differences, while 500 epochs were used for Slow Feature Analysis.

Even though the learning rate did not seem to affect Greedy InfoMax, variations of its value were visually noticeable for Slow feature Analysis as it can be seen in the Figure below. Since the plots for smaller learning rates seem to indicate overfitting, only those with a lower value were used as the final results of the experiment, being $1e-5$ for GIM and $5e-7$ for SFA. For the later, smaller learning rates did not seem to fully capture the ideal circular (cylindrical) shape. Such shape is considered ideal since, thanks to the color-map, it can show how the original relation between data is maintained in the network's interpretation. As stated in Wiskott's work, the circumference of the cylinder is encoding the rotation of the plane toy. Neighboring angles become neighboring points in the plot in a symmetrical, radial way.

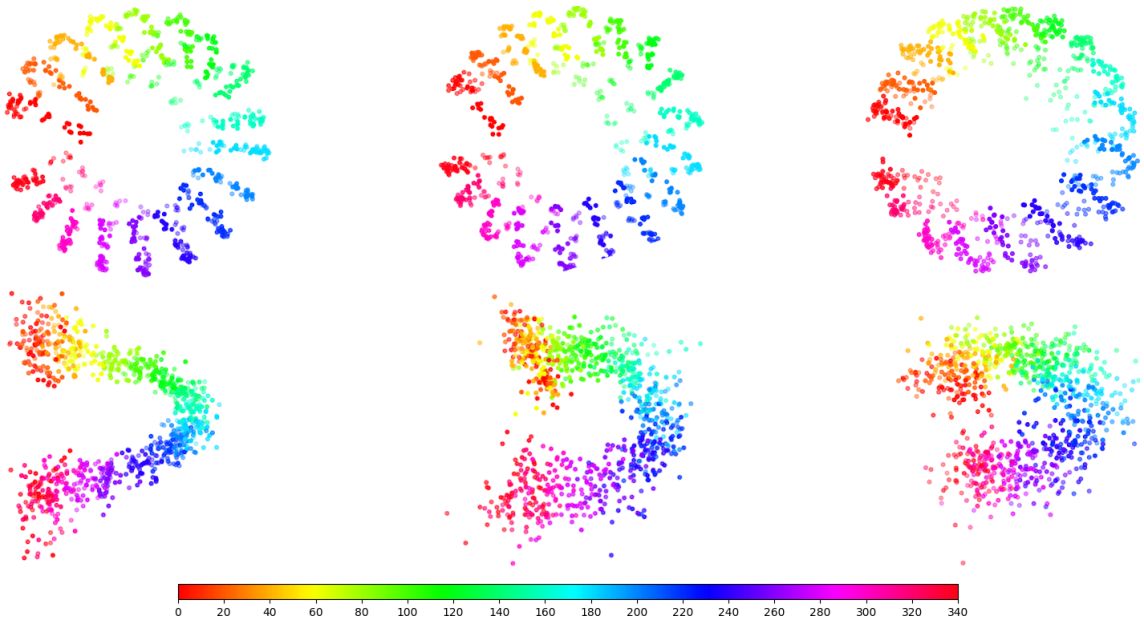

Figure 21: Top-view of color-mapped azimuth values for the outputs obtained in the Single-Module architecture with different learning rates for both GIM (top: 1e-3, 1e-4, 1e-5) and SFA (bottom: 5e-5, 1e-6, 5e-7).

Even if the constructed models were not able to fully capture the circularity with SFA, Wiskott's experiment serves as evidence that it is indeed attainable and it should be possible to come closer to their results by improving the model and the training process, or using a different ratio of training and testing data, as they used 70-30 instead of 50-50. In any case, Greedy InfoMax displayed a

more defined and coherent shape than Slow Feature Analysis, specially for the training data. The consistency between Single and Multi-Module architectures was also much higher for the former, resulting in almost identical shapes, with the exception of a reversed relation between the spatial mapping of elevation and azimuth. This is mainly due to the non-fully-deterministic learning process of the unsupervised model. The outputs are learnt features, but the model was never told *how* to learn them, and therefore it is not surprising that the colors for some outputs differ in their clockwise or counter-clockwise relation. This can be observed in the plots, which also were positioned manually via `pyplot`'s interactive plotting and the equivalence between Figures might not be exact, but the 30 degrees value for the elevation was always placed on the bottom for the Top and Side views to make visualization easier. For the same reason, 0/360 degrees of azimuth were attempted to be placed always on one side for those views. Finally, and due to the process of image capturing being manual, a loop that switched between color maps was implemented so that capturing the same perspective for both azimuth and elevation was possible.

It is important to quote the following statement from [30]: *"Note that each object instance was placed in a different initial pose, therefore "0 degree angle" may mean "facing left" for one instance of an animal, and "facing 30 degree right" for another instance."* In addition to this, and also due to the small size of the dataset —less than 1000 samples since it is just one toy—, the plots for the test set are particularly chaotic, even though the dominant pattern is still that corresponding to the expected color gradient —even though the *colors* (angles) for the testing set might not correspond to those from the training set. However, the goal was not to output any specific number for a pose variation, but rather figure out whether the model was able to learn those variations in such a way that the results of that learning process could be understood. In order to take the alternative approach of predicting or *classifying* the angles, all data samples should be labeled according to a common reference.

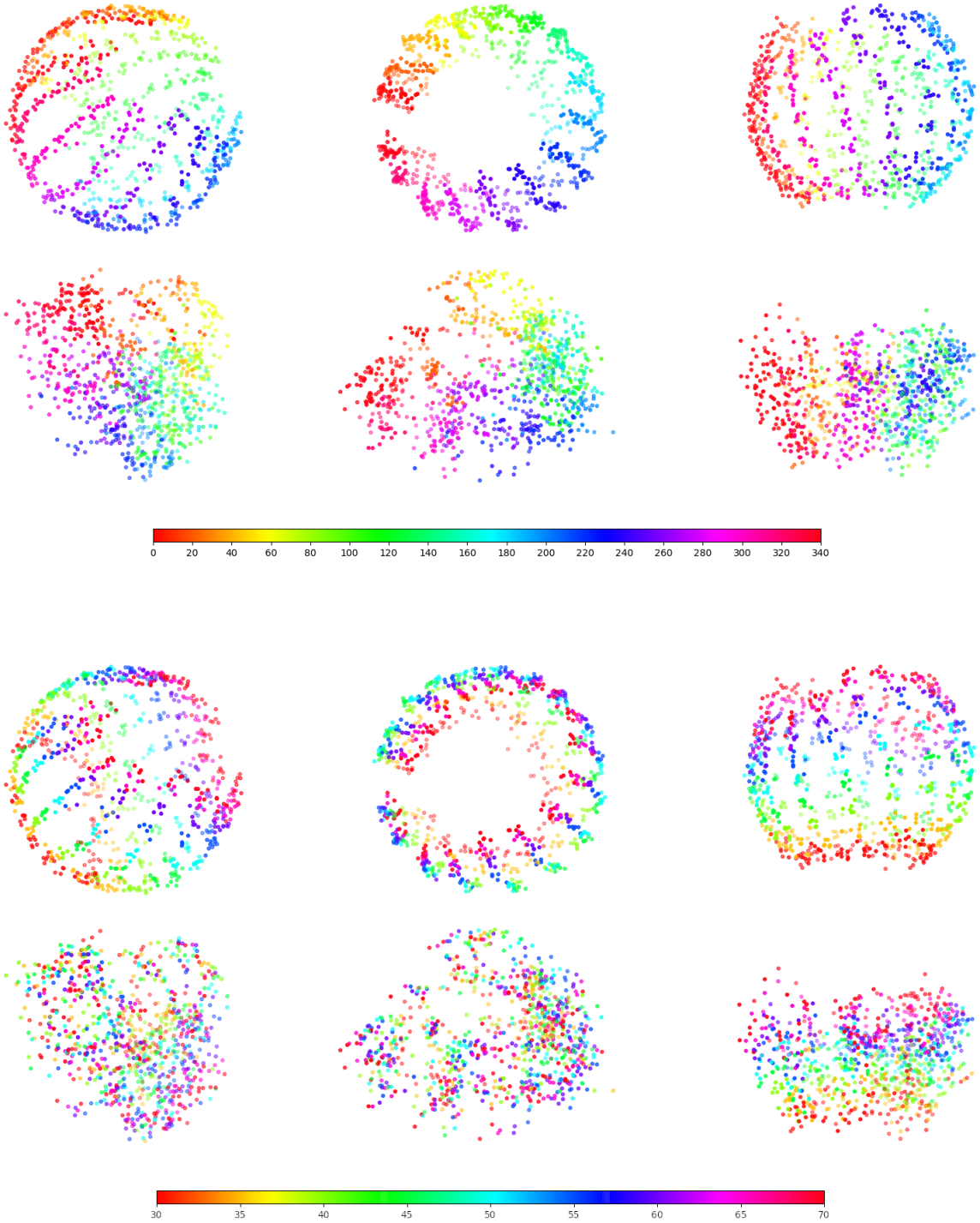

Figure 22: From left to right: Oblique, Top and Side views of color-mapped azimuth (top group) and elevation (bottom group) angles for the outputs obtained in the Single-Module architecture with GIM. For each group, the top row depicts the training data and the bottom row corresponds to the testing data.

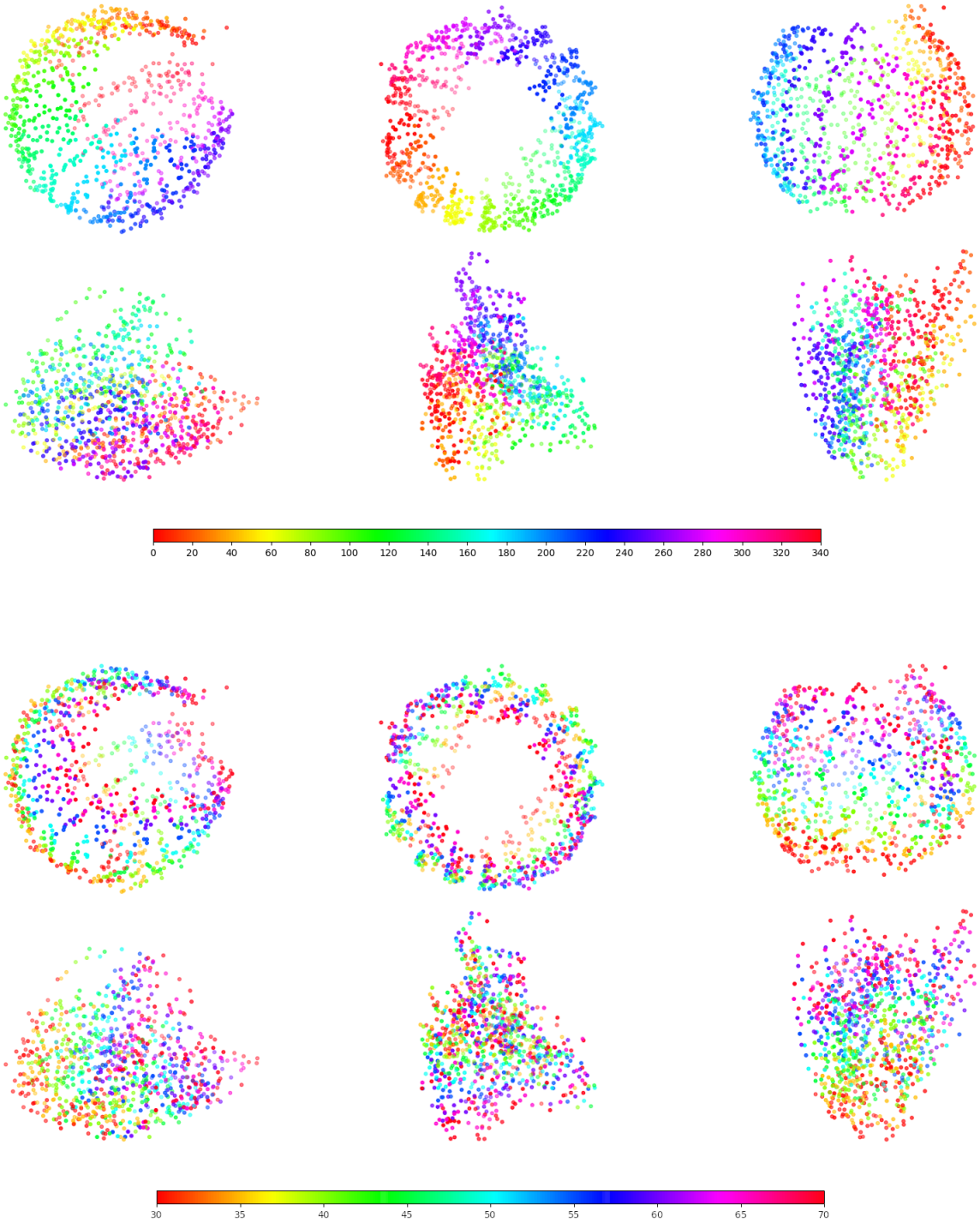

Figure 23: From left to right: Oblique, Top and Side views of color-mapped azimuth (top group) and elevation (bottom group) angles for the outputs obtained in the Multi-Module architecture with GIM. For each group, the top row depicts the training data and the bottom row corresponds to the testing data.

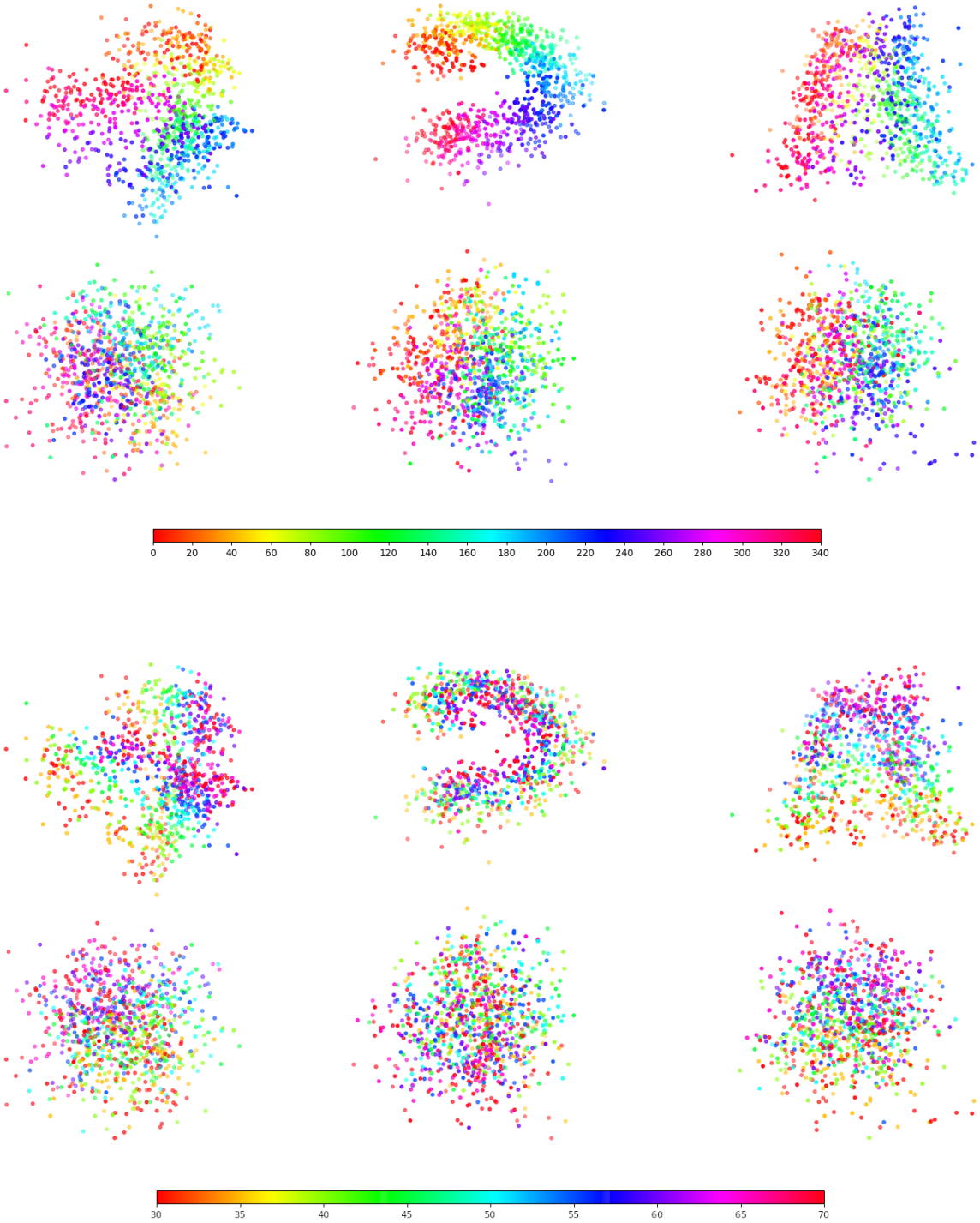

Figure 24: From left to right: Oblique, Top and Side views of color-mapped azimuth (top group) and elevation (bottom group) angles for the outputs obtained in the Single-Module architecture with SFA. For each group, the top row depicts the training data and the bottom row corresponds to the testing data.

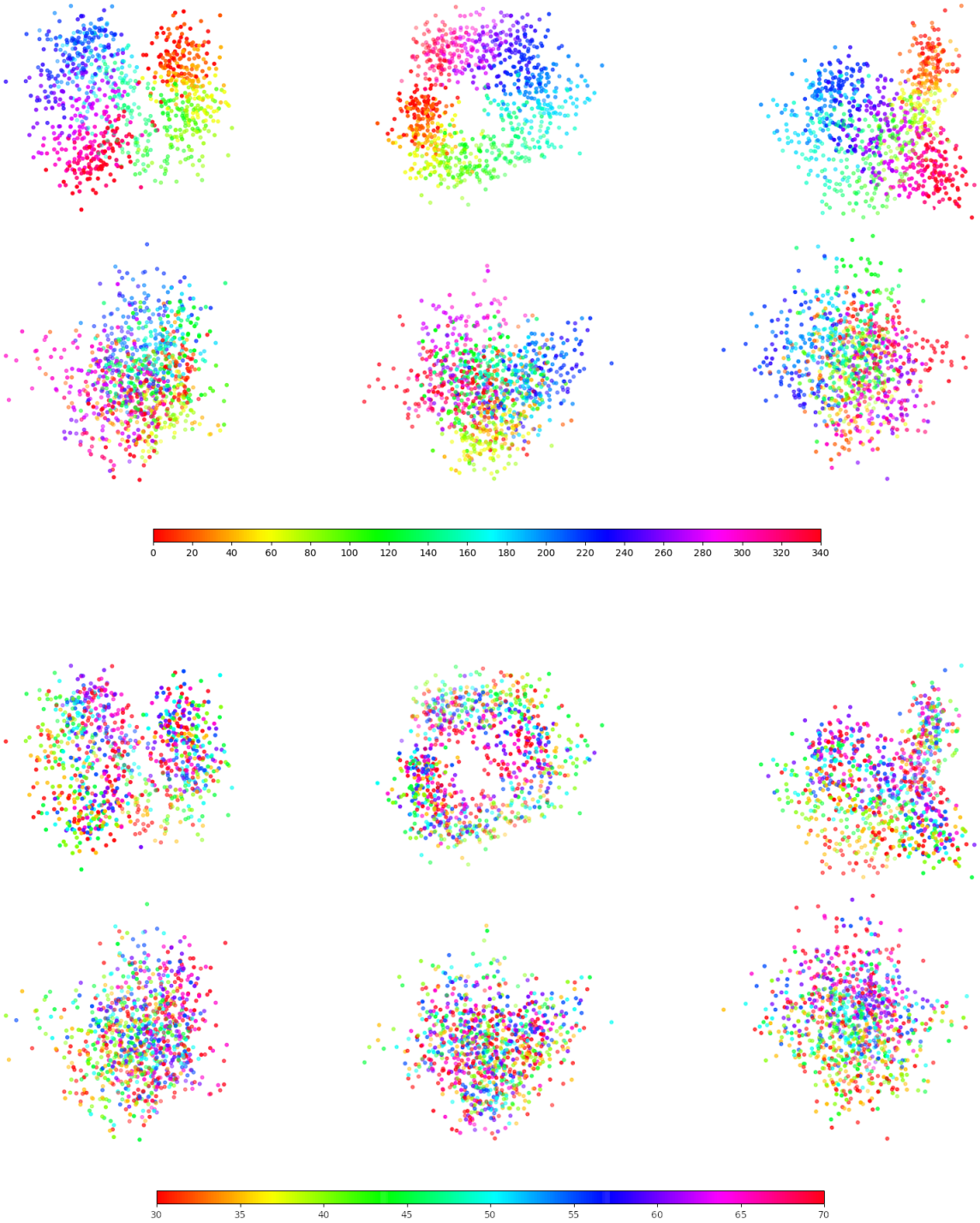

Figure 25: From left to right: Oblique, Top and Side views of color-mapped azimuth (top group) and elevation (bottom group) angles for the outputs obtained in the Multi-Module architecture with SFA. For each group, the top row depicts the training data and the bottom row corresponds to the testing data.

7 Conclusion

This project presented a comparative implementation of the Greedy InfoMax and Slow Feature Analysis algorithms for both the unsupervised extraction of angle variation features and the supervised classification of images after unsupervised model training. The experiments were performed for both a Single-Module and a Multi-Module architectures for a simple convolutional neural network coded with the Machine Learning tools provided by the PyTorch library in Python.

Unsupervised Learning is nowadays a promising field of research, with the potential of keeping up with its supervised counterpart and even surpassing its achievements for certain tasks, while having the right to flaunt its lack of necessity for manual labeling. Methods such as Greedy InfoMax or Slow Feature Analysis prove this capability and the experiments performed during this project provide clear evidence of its versatility.

For a task as common and overexploited as classification, both methods were proven viable, albeit neither excelled in its performance. Greedy InfoMax, even if not observed to take as much advantage of the Multi-Module architecture as expected, has demonstrated how InfoNCE and its variations can be a smart and effective way to compute the loss in unsupervised models. The results, even if not close to the state-of-the-art due to setup limitations, show how all versions of the model accomplished the objective of surpassing 80% accuracy and serve as a proof that both Greedy InfoMax and Slow Feature Analysis variations can be the core of valid and competent Self-Supervised Learning models. There is room for improvement and experimentation, especially with regard to the methods employed for the final classifier and the amount of extracted features. Löwe et al. [7] demonstrated how even classification tasks based on RGB imagery are viable with similar approaches.

The feature extraction experiment was the main highlight due to its fully-unsupervised nature, and it can be considered a success based on the initial goal of achieving meaningful visual information to show that the model is completely capable of learning pose variations in an unsupervised way and with independence to the lighting conditions. For the given architectures, this was specially noticeable for the Greedy InfoMax implementation, establishing itself again as a noteworthy alternative to traditional Slow Feature Analysis. The results demonstrate how the features resulting from the unsupervised training process were logical and informative. It is a fact that Slow Feature Analysis performed worse, but this should not be interpreted as a general truth since the experiments by Wiskott et al. [22] prove otherwise. Implementing and tuning a model based on SFA turned out to be a more difficult task than doing so for Greedy InfoMax. The later seemed to adapt with ease to the architecture and was consistent in its good performance for different hyperparameter combinations, while SFA did not quite operate as nicely with the models. This might perhaps be caused by its requirement of having greater batch-sizes for a proper estimation of the covariance matrix, an issue of which Greedy InfoMax can presume to lack and that sets limitations on the dimensionality of the final features. Overall, Greedy InfoMax seemed more accessible and reliable for small and simple models. Although out of the experiment's scope, this task could be pushed further so that the variations are interpreted *live* and even prompted explicitly for a moving target. Following the theme of the dataset, a similar model could for instance detect incorrect poses for toy pieces in an assembly line so that they can be corrected on time to help avoid the manufacturing of defective products.

In terms of computation time, even if unfortunately the dynamic and traffic-dependent variations of Colab did not allow for the computation of exact and deterministic measurements, both methods displayed a similar performance. Other factors, such as the amount of layers used in the models or the number of training epochs had a much greater impact on the time consumption for the training stage. On average, a complete training with the final architecture would usually take between 70 and 120 minutes for the whole NORB dataset (24300 images), independently of the employed algorithm, and given the remote hardware allowances currently offered by Colab as of 2020.

Despite the above-mentioned aspects, the key comparison lies perhaps in the abstract concept of *slowness*. Even though difficult to asses, the capability of the models to capture slow features is paramount to the experiments. The classification task might not be the best choice for this specific study, although slowness comprehension is intrinsic to the employed methods. For this problem, the models were fed pairs of images corresponding to the same class, in which case the

slow features would be expected to represent information related to the shape of the object, as the pose and lighting will randomly vary from pair to pair but the general shape will remain similar for those toys belonging to the same class. However, in light of the accuracy results, one may argue that these features were not optimally captured, probably due to the reduced amount of samples and the variety present among different toy instances, for which certain classes such as *animals* might display different shapes. Moreover, using the shape as the reference slowest feature might be counterproductive for cases in the likes of the car/truck issue observed in some executions (Figure 16).

On the other hand, slowness detection can be better evaluated with the feature extraction experiment, for which the target features were those corresponding to the angle variations. Only images belonging to a single toy instance were used, and the pairs were forced to display contiguous angles, such that the model will learn to interpret the slowest features to be those corresponding to the slow variations in position of an object with an overall constant shape over time. Other features such as lighting vary much more quickly and the algorithms will disregard them in favor of other subtly changing information sources as the azimuth and elevation poses. The results show a good proof of this behavior, as it can be easily observed how neighboring angles are fully represented in the cylindrical shapes, being Greedy InfoMax particularly accurate for the given model configurations and data. Figures 22 to 25 show that both the azimuth and the elevation are captured and represented in the different axes for all cases, which indicates that learning two main features at the same time is not an issue for either method as long as such features vary with equal or similar slowness over time. The downside of this approach is that pose evaluation requires the test data to be measured with the same references to ensure consistency, which was not the case for the NORB dataset but should not be an issue for unlabeled data since, in theory, it is expected that the learned references will be used by default.

Overall, both methods are presented to be comparable at least in terms of slow feature detection, with the novel Greedy InfoMax outshining its veteran adversary for the conducted experiments, and within the limits of the chosen setup. The outcome of this project leads to a positive vision regarding the future and potential of InfoNCE-based learning algorithms and unsupervised methods inspired by Slow Feature Analysis; advocating for the inherent possibilities offered by the unsupervised paradigm, of which Machine Learning will likely benefit in the upcoming years.

References

[1] Breiman, L. (2001). *Random Forests*. Machine Learning, 45, pp. 5-32. Source

[2] Vapnik, V. N., Boser, B. E. and Guyon, I. M. (1992). *A training algorithm for optimal margin classifiers*. Proceedings of the fifth annual workshop on Computational Learning theory, pp. 144-152. doi:10.1145/130385.130401

[3] Sathya, R. and Abraham, A. (2013). *Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification*. International Journal of Advanced Research in Artificial Intelligence, 2(2). doi:10.14569/IJARAI.2013.020206

[4] Mahboob, T., Khanam, M. and Imtiaz, W. (2015). *A Survey on Unsupervised Machine Learning Algorithms for Automation, Classification and Maintenance*. International Journal of Computer Applications, 119(13), pp. 34-39. doi:10.5120/21131-4058

[5] Ch. Aswani Kumar (2009). *Analysis of Unsupervised Dimensionality Reduction Techniques*. Computer Science and Information Systems, 6(2), pp. 217-227. doi:10.2298/csis0902217K

[6] Wiskott, L. and Sejnowski, T. J. (2002). *Slow Feature Analysis: Unsupervised Learning of Invariances*. Neural Computation, 14(4), pp. 715-770. doi:10.1162/089976602317318938

[7] Löwe, S., O'Connor, P. and Veeling, B. S. (2019). *Putting An End to End-to-End: Gradient-Isolated Learning of Representations*. 33rd Conference on Neural Information Processing Systems. arXiv:1905.11786v3

[8] Gutmann, M. and Hyvärinen, A. (2010). *Noise-contrastive estimation: A new estimation principle for unnormalized statistical models*. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 297–304. Source

[9] Ma, Z. and Collins, M. (2018). *Noise Contrastive Estimation and Negative Sampling for Conditional Models: Consistency and Statistical Efficiency*. 2018 Conference on Empirical Methods in Natural Language Processing. arXiv:*1809.01812v1*

[10] Balduzzi, D., Vanchinathan, H. and Buhmann, J. (2015). *Kickback cuts Backprop's red-tape: Biologically plausible credit assignment in neural networks*. Twenty-Ninth AAAI Conference on Artificial Intelligence. arXiv:1411.6191

[11] Scellier, B. and Bengio, Y. (2017). *Equilibrium propagation: bridging the gap between energy- based models and backpropagation*. Frontiers in computational neuroscience, pp. 11-24. doi:10.3389/fncom.2017.00024

[12] Kohan, A. A., Rietman, E. A. and Siegelmann, H. T. (2018). *Error Forward-Propagation: Reusing Feedforward Connections to Propagate Errors in Deep Learning*. arXiv:1808.03357

[13] Bartunov, S., Santoro, A., Richards, B. A., Hinton, G. E. and Lillicrap, T. (2018). *Assessing the scalability of biologically-motivated deep learning algorithms and architectures*. Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp. 9390–9400. arXiv:1807.04587

[14] Xiao, W., Chen, H., Liao, Q. and Poggio, T. (2019). *Biologically-plausible learning algorithms can scale to large datasets*. Proceedings of the 7th International Conference on Learning Representations. Source

[15] Belilovsky, E., Eickenberg, M. and Oyallon, E. (2019). *Greedy layerwise learning can scale to imagenet*. International Conference on Machine Learning, pp. 583–593. arXiv:1812.11446v3

[16] Deng, J., Dong, W., Socher, R. Li, L-J, Li, K. and Fei-Fei, L. (2009). *ImageNet: A Large-Scale Hierarchical Image Database*. Proceedings of the 2009 Conference on Computer Vision and Pattern Recognition. doi:10.1109/CVPR.2009.5206848

[17] Metka, B., Franzius, M. and Bauer-Wersing, U. (2018). *Bio-inspired visual self-localization in real world scenarios using Slow Feature Analysis*. PLOS ONE, 13(9). doi:10.1371/journal.pone.0203994

[18] Franzius, M., Sprekeler, H. and Wiskott, L. (2007). *Slowness and Sparseness Lead to Place, Head-Direction, and Spatial-View Cells*. PLOS Computational Biology, 3(8). doi:10.1371/journal.pcbi.0030166

[19] Zhang, Z. and Dacheng, T. (2012). *Slow Feature Analysis for Human Action Recognition*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 34(3), pp. 436-450. arXiv:1907.06670

[20] Ghosh, R., Siyi, T., Rasouli, M., Thakor, N. V. and Kukreja, S. L. (2019). *Pose-invariant object recognition for event-based vision with slow-ELM*. Proceedings of the 25th International Conference on Artificial Neural Networks, pp. 455-462. doi:10.1109/TPAMI.2011.157

[21] Huang, G-B., Zhu, Q-Y. and Siew, C. K. (2006). *Extreme learning machine: Theory and applications*. Neurocomputing, 70(13), pp. 489–501. doi:10.1016/j.neucom.2005.12.126

[22] Schüler, M., Hlynsson, H. D. and Wiskott, L. (2018). *Gradient-based Training of Slow Feature Analysis by Differentiable Approximate Whitening*. Proceedings of The Eleventh Asian Conference on Machine Learning, pp. 316-331. arXiv:1808.08833

[23] Du, B., Ru, L., Chen Wu and Zhang, L. (2019). *Unsupervised Deep Slow Feature Analysis for-Change Detection in Multi-Temporal RemoteSensing Images*. IEEE Transactions on Geoscience and Remote Sensing, 57(12). doi:10.1109/TGRS.2019.2930682

[24] Schuld, M., Sinayskiy, I. and Petruccione, F. (2014). *An introduction to quantum machine learning*. Contemporary Physics, 56(2). doi:10.1080/00107514.2014.964942

[25] Kerenidis, I. and Luongo, A. (2018). *Quantum classification of the MNIST dataset via Slow Feature Analysis*. arXiv:1805.08837

[26] Van den Oord, A., Li, Y. and Vinyals, O. (2018). *Representation Learning with Contrastive Predictive Coding*. Preprint, work in progress. International Conference on Learning Representations 2020. arXiv:1807.03748v2

[27] Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A. and Bengio, Y. (2018). *Learning deep representations by mutual information estimation and maximization*. International Conference for Learning Representations 2019. arXiv:1808.06670

[28] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2014). *Generative Adversarial Networks*. Advances in Neural Information Processing Systems, 27. arXiv:1406.2661

[29] Tschannen, M., Djolonga, J., Rubenstein, P. K., Gelly, S. and Lucic, M. (2020). *On Mutual Information Maximization for Representation Learning*. International Conference on Learning Representations 2020. arXiv:1907.13625

[30] LeCun, Y., Huang, F. J. and Bottou, L. (2004). *Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting*. Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. doi:10.1109/CVPR.2004.1315150

[31] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimselshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. and Chintala, S. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Conference on Neural Information Processing Systems 2019, pp. 8024-8035. arXiv:1912.01703

[32] Ghahramani, Z. (2004). *Unsupervised Learning*. Advanced Lectures on Machine Learning, pp. 72-112. Source

[33] Hinton, G. E. and Sejnowski, T. J. (1999). *Unsupervised Learning: Foundations of Neural Computation*. The MIT Press. doi:10.7551/mitpress/7011.003.0002

[34] Zhang, C., Song, D., Chen, Y., Feng, X., Lumezanu, C., Cheng, W., Ni, J., Zong, B., Chen, H. and Chawla, N. V. (2018). *A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data*. arXiv:1811.08055v1

[35] Amanda Berg, Jörgen Ahlberg, Michael Felsberg (2019). *Unsupervised Learning of Anomaly Detection from Contaminated Image Data using Simultaneous Encoder Training*. arXiv:1905.11034v2

[36] Usama, M., Qadir, J., Raza, A., Arif, H., Yau, K-L. A., Elkhatib, Y., Hussain, A. and Al-Fuqaha, A. (2017). *Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges*. IEEE Access, 7. doi:10.1109/ACCESS.2019.2916648

[37] Cios, K. J., Swiniarski, R. W., Pedrycz, W. and Kurgan, L. A. (2007). *Unsupervised Learning: Association Rules*. Data Mining, pp. 289-306. Source

[38] Agrawal, R., Imielinski, T. and Swami, A. (1993). *Mining association rules between sets of items in large databases*. Proceedings of the 1993 ACM-SIGMOD International Conference on Management of Data, pp. 207–216. doi:10.1145/170035.170072

[39] Pehlevan, C. and Chklovskii, D. B. (2019). *Neuroscience-inspired online unsupervised learning algorithms*. IEEE Signal Processing Magazine, 36(6). doi:10.1109/MSP.2019.2933846

[40] Peterson, L. E. (2009). *K-nearest neighbor* Scholarpedia, 4(2), pp. 1883. doi:10.4249/scholarpedia.1883

[41] Rumelhart, D., Hinton, G. E. and Williams, R. (1986). *Learning representations by back-propagating errors*. Nature, 323(6088), pp. 533-536. doi:10.1038/323533a0

[42] Kelley, H. J. (1960). *Gradient theory of optimal flight paths*. ARS Journal, 30(10), pp. 947–954. doi:10.2514/8.5282

[43] Dreyfus, S. (1962). *The numerical solution of variational problems*. Journal of Mathematical Analysis and Applications, 5(1), pp. 30–45. doi:10.1016/0022-247x(62)90004-5

[44] Rosenblatt, F. (1961). *Principles of neurodynamics: Peceptrons and then Theory of Brain Mechanisms*. Brain Theory, pp. 245-248. doi:10.1007/978-3-642-70911-1_20

[45] Bottou, L., Curtis, F. and Nocedal, J. (2018). *Optimization Methods for Large-Scale Machine Learning*. SIAM Review, 60(2), pp. 223-311. doi:10.1137/16M1080173

[46] Kingma, D. P. amd Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. 3rd International Conference for Learning Representations. arXiv:1412.6980v9

[47] Hinton, G. E. (2012). *Lecture 6 - Neural Networks for Machine Learning*. Source

[48] Ioffe, S. and Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. Proceedings of the 32nd International Conference on International Conference on Machine Learning, 37, pp. 448-456. arXiv:1502.03167v3

[49] Mitchison, G. (1991). *Removing time variation with the anti-Hebbian differential synapse*. Neural Computation, 3(3), 312–320. doi:10.1162/neco.1991.3.3.312

[50] Löwe, S. (2019). *Greedy InfoMax for Self-Supervised Representation Learning*. Master Thesis, MSc Artificial Intelligence, University of Amsterdam. Source

[51] Marblestone, A. H., Wayne and G., Kording, K. P. (2016). *Toward an integration of deep learning and neuroscience*. Frontiers in computational neuroscience, 10(5), pp. 10-94. doi:10.3389/fncom.2016.00094

[52] Caporale, N. and Dan, Y. (2008). *Spike timing–dependent plasticity: a hebbian learning rule*. Annual Review of Neuroscience, 31, pp. 25–46. doi:10.1146/annurev.neuro.31.060407.125639

[53] He, K., Zhang, X., Ren, S. and Sun, J. (2016). *Deep Residual Learning for Image Recognition*. 2016 IEEE Conference on Computer Vision and Pattern Recognition. doi:10.1109/CVPR.2016.90

[54] Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*. Source

[55] Simonyan, K. and Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. ImageNet Large Scale Visual Recognition Challenge 2014. arXiv:1409.1556v6

[56] Coates, A., Ng, A. and Lee, H. (2011). *An analysis of single-layer networks in unsupervised feature learning*. Journal of Machine Learning Research, 15, pp. 215-223. Source