UNIVERSITAT DE BARCELONA

Treball final de grau

DOBLE GRAU DE MATEMÀTIQUES I ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

EXPLORING SELF-SUPERVISED LEARNING IN DEEP LEARNING FOR FOOD RECOGNITION

Autor: Nil Ballús Riu

Directors:	**Dra. Petia Radeva i Dr. Alex Haro**
Tutor:	**Bhalaji Nagarajan**
Realitzat a:	**Departament de Matemàtiques i Informàtica**

Barcelona, **24 de gener de 2022**

Abstract

Deep Learning is a field that evolves at dizzying rates and that allows obtaining extraordinary results in many applications of scientific and, even, everyday scope. In the field of computer vision, and especially in the food image recognition task, Machine Learning allows us to achieve accuracies close to human capacity. However, Machine Learning-based models often require a large number of labeled example images to carry out the learning process. Although we currently have millions of images, for example from the Internet or social networks, the main problem is that the process of labeling images is very expensive, difficult, and sometimes impossible or unfeasible. Therefore, the question we ask ourselves is: can we take advantage of large volumes of unlabeled images to help solving the problem of Supervised Learning?

To answer the above question, in the project we have set out to explore the newest Self-supervised Learning methods to test whether the use of unlabeled images can help a food classifier improve its accuracy. Once analyzed in detail, we have proposed a new Self-Supervised Learning method, called OptSSL, which shows great performance in representing food images in a latent space. To demonstrate its potential, we have made an exhaustive comparison of this method with other recent Self-supervised Learning methods proposed in the literature.

Another problem discussed in this paper is the modeling of the uncertainty of the predictions of a model. Most current Deep Learning models are not able to decide whether their predictions are confident or not. Thus, in the project, this property is incorporated into the analyzed classifiers, and the performance of the classifier is analyzed not only from the point of view of its accuracy but also from its uncertainty, making a comparison of its robustness in this context.

As a result of the work, we show that OptSSL is an optimal method for data representation, and in particular of food images, which benefits from unlabeled images to learn the representation. Finally, we show how the OptSSL method can be used as a pre-training phase to subsequently train a food classifier, obtaining high accuracy and low uncertainty in predictions.

2020 Mathematics Subject Classification. 68T07

Resum

L'aprenentatge profund (en anglès *Deep Learning*) és un camp que evoluciona a ritmes vertiginosos i que permet obtenir resultats extraordinaris en moltes aplicacions d'àmbit científic i, fins i tot, quotidià. En el camp de la visió per computador, i en especial en el reconeixement d'imatges de menjar, l'aprenentatge automàtic permet assolir precisicions gairebé a l'alçada de la capacitat humana. No obstant, els models basats en aprenentatge automàtic sovint requereixen d'un gran nombre d'imatges d'exemple etiquetades per poder dur a terme el procés d'aprenentatge. Tot i que actualment disposem de milions d'imatges, per exemple provinents d'Internet o de xarxes socials, el principal problema és que el procés d'etiquetatge de les imatges és molt costós, difícil i a vegades impossible o inviable. Davant d'això, la pregunta que ens fem és: podem treure profit dels grans volums d'imatges no etiquetades per ajudar al problema de l'aprenentatge supervisat?

Per tal de donar resposta a la pregunta anterior, en el treball ens hem proposat explorar els últims mètodes d'aprenentatge auto-supervisat (en anglès *Self-supervised Learning*) per comprovar si l'ús d'imatges no etiquetades pot ajudar a un classificador de menjar a millorar la seva precisió. Una vegada analitzats en detall, hem proposat un nou mètode d'aprenentatge auto-supervisat, anomenat OptSSL, que mostra un gran rendiment en la representació d'imatges de menjar. Per demostrar el seu potencial, fem una comparativa exhaustiva d'aquest mètode amb altres mètodes d'auto-aprenentatge recents proposats a la literatura.

Paral·lelament, un altre problema tractat en el treball és el modelatge de la incertesa de les prediccions d'un model. La majoria de models d'aprenentatge profund actuals no són capaços de decidir si les seves prediccions són de confiança o no. Així, en el treball s'incorpora aquesta propietat als classificadors analitzats, i el rendiment del classificador s'analitza no només des del punt de vista de la seva precisió, sinó també a partir de la seva incertesa, fent una comparació de la seva robustesa en aquest context.

Com a resultat del treball, mostrem que OptSSL és un mètode òptim per la representació de dades, i en particular d'imatges de menjar, que permet beneficiar-se d'imatges no etiquetades d'una forma elegant per aprendre la representació. Finalment, mostrem com es pot utilitzar el mètode OptSSL com a pre-entrenament per posteriorment entrenar un classificador de menjar i obtenir una altra precisió i una baixa incertesa en les prediccions.

Resumen

El aprendizaje profundo (en inglés *Deep Learning*) es un campo que evoluciona a ritmos vertiginosos y que permite obtener resultados extraordinarios en muchas aplicaciones de ámbito científico e incluso cuotidiano. En el campo de la visión por computador, y en especial en el reconocimiento de imágenes de comida, el aprendizaje automático permite lograr precisiones cercanas a la capacidad humana. Sin embargo, los modelos basados en aprendizaje automático a menudo requieren de un gran número de imágenes de ejemplo etiquetadas para poder llevar a cabo el proceso de aprendizaje. Aunque actualmente disponemos de millones de imágenes, por ejemplo provenientes de Internet o de redes sociales, el principal problema es que el proceso de etiquetado de las imágenes es muy costoso, difícil y en ocasiones imposible o inviable. Ante esto, la pregunta que nos hacemos es: ¿podemos sacar provecho de los grandes volúmenes de imágenes no etiquetadas para ayudar a resolver el problema del aprendizaje supervisado?

Para dar respuesta a la pregunta anterior, en el trabajo nos hemos propuesto explorar los métodos más novedosos de aprendizaje auto-supervisado (en inglés *Self-supervised Learning*) para comprobar si el uso de imágenes no etiquetadas puede ayudar a un clasificador de comida a mejorar su precisión. Una vez analizados en detalle, hemos propuesto un nuevo método de aprendizaje auto-supervisado, llamado OptSSL, que muestra un gran rendimiento en la representación de imágenes de comida. Para demostrar su potencial, hacemos una comparativa exhaustiva de este método con otros métodos de auto-aprendizaje recientes propuestos en la literatura.

Paralelamente, otro problema tratado en este trabajo es el modelado de la incertidumbre de las predicciones de un modelo. La mayoría de los modelos de aprendizaje profundo actuales no son capaces de decidir si sus predicciones son de confianza o no. Así, en el trabajo se incorpora esta propiedad a los clasificadores analizados, y el rendimiento del clasificador se analiza no sólo desde el punto de vista de su precisión, sino también a partir de su incertidumbre, haciendo una comparación de su robustez en este contexto.

Como resultado del trabajo, mostramos que OptSSL es un método óptimo para la representación de datos, y en particular de imágenes de comida, que permite beneficiarse de imágenes no etiquetadas para aprender la representación. Para terminar, mostramos cómo se puede utilizar el método OptSSL como pre-entrenamiento para posteriormente entrenar a un clasificador de comida y obtener una alta precisión y una baja incertidumbre en las predicciones.

Acknowledgements

I am fully aware that many people contributed, directly or indirectly, to the realization of this project. For this reason, I would like to express my gratitude to all of them for the help and support received.

First of all, I would like to acknowledge, in a very special way, to my supervisor Dr. Petia Radeva for all the aid and assistance that she has provided me with. She was who suggested the topic of this study and because of this I have been able to enjoy the experience of developing a research project, something totally new for me. She was entirely involved in the project, constantly proposing hypotheses and leading me to find new lines of exploration. Furthermore, she was always available for any question or doubt that I could have.

I also want to thank the collaboration and support from Bhalaji Nagarajan, a PhD Candidate at UB. He contributed a lot with his knowledge on the topic, helping me with some implementation details, analyzing the results and proposing improvements for the models to outperform the results. Moreover, he always encouraged me to continue investigating when the results were not favorable and I got frustrated.

Finally, I would like to thank Dr. Alex Haro for his help in the theoretical part, in particular for the mathematical approaches and his willingness to contribute with particular details.

On a second note, even though they did not have a direct impact on the project, I also want to thank those people who showed emotional support in stress and tension moments. My parents, Josep and Sílvia, and my sister Estel who helped me in whatever they could, specially in difficult moments when I was disappointed due to poor results, and always encouraged me to have a more positive outlook and face the problems as a challenge with an optimistic vision. Last but not least, I want to thank all my friends who have been with me in this adventure and specially Marc Closa and Carla Rodríguez, childhood friends of mine who emotionally supported me a lot during the project.

Contents

Chapter 1

Introduction

The purpose of this chapter is to make a short introduction on the context of the project, the motivation of the author, and the main objectives that are intended to be achieved, as well as the description of the planning. Furthermore, a brief explanation of the organization of the memory is provided.

1.1 Context of the Project

Artificial Intelligence (AI) and Deep Learning (DL) are changing our lives. Self-driving cars, virtual assistants, auto-face detection when taking a photo or mobile face unlocks could not be understood without the application of the deep learning technology. Nevertheless, not all is *cakes and ale*. In 2015, the Google Photos classification system wrongly identified two Brooklyn-natives as gorillas, a racist act that caused much controversy [3]. In May 2016, there was the first fatality with a Tesla's assisted driving system. The company reported that the system could not recognize a white truck from the bright sky, which caused the fatal car crash [61].

After acknowledging that AI and DL are still not perfect, some questions come to our minds. As it is well-known, DL methods are very greedy methods. In order to achieve high performance, they need thousands of images at least. Today, finding millions of images is relatively feasible but annotating them is the real bottleneck. So the first question we have is how to leverage the huge amount of unlabelled data.

Another not less important question is how much we can rely on a model's prediction. We often assume that the results are accurate which, as we have seen, is not always true. In some critical applications, it is important that a model can just say "I do not know" or "I am not really sure". Technically speaking, it means that a model should be able to model uncertainty, so that when giving a prediction, if its uncertainty is high, exceptional decisions can be performed.

DL would not achieve so much popularity if it was not capable to solve real complex

problems. One emerging field that can take advantage of DL and uncertainty modeling is the food recognition task. We live in a society where "time is money". It causes that, inevitably, we want to reduce the time spent on basic everyday actions, like meals. This leads into unhealthy eating habits that can end in a drastically way, such as a chronic disease. It is difficult to auto-detect bad eating habits, but if we could see an overview of the unbalanced nutrients ingested in a week, we would be surprised. However, doing such tracking manually is tedious and time-consuming.

More extreme cases are with people who suffer malnutrition diseases, like undernutrition, overweight, obesity or diabetes. In such cases, medical treatment is required and it is fundamental to control daily food intake. To this purpose, patients must record whatever they eat. Historically, this daily recording has been a manual task, based on filling in daily questionnaires, but it is dreary and a susceptible forgettable task.

Due to these needs, systems that can automate this monitoring process have a great opportunity. For instance, would it not be pleasant to just have to take a photo of what we are eating at the moment and the tracking process were done automatically? Here is where the food recognition task appears. Regarding this concern, several successful applications to recognize food and estimate calories from photos based on DL have been launched in the last years, like FoodTraker [55] or Calorie Mama [34]. However, such applications are not aware about the uncertainty in their predictions. Thus, there is still progress to do and more reliable models can be designed.

1.2 Project Motivation

Image analysis is a widespread topic inside Computer Vision with big explorations in the last years. It involves different visual tasks, such as object recognition, object detection, or object segmentation. In this project, will focus on what is considered the fundamental problem, the object recognition task, applied to food images. The great progress in the deep learning field has enabled to develop powerful algorithms for this task, being Convolutional Neural Networks (CNN) the state-of-the-art methods. However, there is still work to do. The main challenges on the food recognition task are to deal with the low inter-class and high intra-class variation, illustrated in Fig. 1.1.

Figure 1.1: **Top row**: examples of high intra-class variation ("Curry rice"). **Bottom row**: examples of low inter-class variation ("Fried noodles" vs. "Spaghetti", "Donuts" vs. "Bagel", and "Frech fries" vs. "Poutine"). Image taken from [4].

Even for neural networks, the state-of-the-art algorithms for most visual tasks, the food recognition task is a complex problem due to the variation property of the data (Fig. 1.1). Therefore, we contemplate the following question: how can the food recognition task be improved and how much confidence are food classifier models' predictions? This leads us to the first motivation of the project, which is to apply unlabeled data in a self-supervised learning mode to improve a model and use the uncertainty quantification to evaluate the food classifier's predictions. The motivation comes from the point that traditional neural networks need a large amount of labeled data to get a good performance. We believe that also unlabeled data can help improve the performance and reduce the uncertainty of a model. From the growing interest in the literature and the potential of Self-supervised learning (SSL), we want to explore novel SSL techniques in the food recognition field to see if they can help improve a baseline DL model for food recognition.

Furthermore, there are some other personal motivations for the project. The first one is to introduce myself to DL and neural networks and understand and apply their theoretical basis. The second one is to have a first contact with research and experience how researchers work in their everyday life.

1.3 Objectives of the Project

In this project, we want to dive into the food recognition problem using CNNs. The first goal that comes to our minds when dealing with classification problems is to only get a model with the highest possible performance, in terms of accuracy. However, we try to go a little further, and also study the reliability of models' predictions. The optimization of the food recognition problem is a well-known although still hard issue. Moreover, the application of Self-supervised Learning (SSL) and the study of uncertainty are not so much explored, so our research goes on this direction. The particular objectives of this project are:

- Understand and present the theoretical basis of SSL and uncertainty modeling on neural networks.
- Explore state-of-the-art SSL methods to try to improve the performance and reduce the uncertainty of a baseline model.
- Investigate and introduce new strategies on existent state-of-the-art SSL methods to try to overcome their performance.
- Quantify the uncertainty of the baseline model through different measures and analyze the results.
- Explore the food recognition problem and define a baseline model using SSL with a relatively good performance.
- Report, validate and analyze all the results obtained.

1.4 Project Planning

This project was carried out during the summer and the fall semester of 2021-2022 course. My first contact with my director, Petia Ivanova Radeva, was on the 10th of May, 2021. We scheduled a first meeting to exchange motivations, personal interests and objectives. After this meeting, we first decided to begin with an introduction to Deep Learning and Convolutional Neural Networks (CNNs), given that this subject is not covered by the current subjects of the Computer Science study. We focused on the Standford course CS231n [44] and the book Dive into Deep Learning [65]. After this first introduction, we continued with the first steps by working for the first time with a CNN. We faced a common initial problem which is the fine-tuning problem, exploring different hyperparameters and checking their effect on the performance.

Once finished, we sat and talk again about the domain, the purposes of the project, and the open problems in the field at the time. In that meeting, Dr. Petia Radeva introduced Bhalaji Nagarajan, a doctoral student at UB who would become my tutor, to me. They posed me the possibility of studying the SSL opportunities and the uncertainty estimation problem on DL models, which I found very interesting. In the next months, we began developing all the points described in Section 1.3 using multiple models, hypothesis, comparisons and a lot of patience.

1.5 Organization of the Memory

The memory is structured in a linear way, beginning with general and basic concepts and going to the detailed architectures and notions used in the experimental part to achieve the goals. The memory can be divided basically into two parts: the theoretical framework, consisting of Chapters 2 and 3, and the experimental section, consisting of Chapters 4, 5 and 6. In particular, each chapter includes:

2. **State of the Art**: a brief report about the last Self-supervised methods proposed in the literature and the last studies on uncertainty modeling and analysis.
3. **Methodology**: the necessary knowledge and mathematical basis to understand Self-supervised learning, uncertainty modeling, uncertainty quantification and the architectures and metrics used in the experimental part.
4. **Our proposal: OptSSL - A New Method for SSL**: the architecture of a new proposed SSL method to learn data representation, which shows competitive results in the food image representation domain.
5. **Validation**: an exposition of the results obtained from all the experiments carried out and a comparison and analysis of the different methods used.
6. **Conclusions and Future Work**: conclusions of the project and possible lines to continue exploring beyond.
7. **Bibliography**: a list of papers, articles and literature consulted for the project.

Chapter 2

State of the Art

2.1 Artificial Intelligence, Machine Learning and Deep Learning

Sometimes the terms Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL) are wrongly exchanged or confused, even though they are clearly distinguishable. AI is an academic discipline that comprises any technique used by computer systems to do tasks that presumably require human intelligence. Regarding ML, it is a subfield of AI that includes the methods which, instead of being explicitly programmed, they iteratively learn from training data and improve through experience. Finally, DL is a subfield of ML based on artificial neural networks with three or more layers, that attempt to simulate the learning process of the human brain. This relationship of inclusion is represented in a Venn diagram in Fig. 2.1.

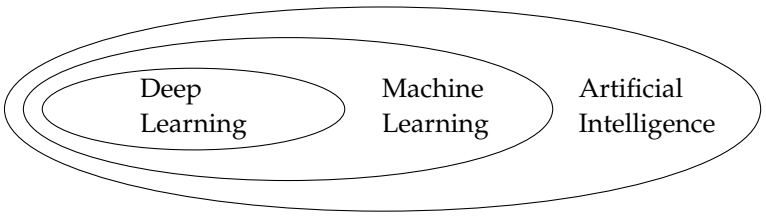


Figure 2.1: Venn diagram of Artificial Intelligence, Machine Learning and Deep Learning.

2.1.1 Types of Machine Learning

Regarding the learning process, ML can be divided into three main types [35]: Reinforcement Learning, Supervised Learning and Unsupervised Learning.

In **Reinforcement Learning**, the models learn by the principle of trial and error. Given the current state of the system, we specify a target and provide all the available actions

with their environmental constraints. Then, the system learns by experience to achieve the target maximizing a reward, trying the possible actions efficiently. **Supervised Learning** (SL) has a different paradigm. The training process of methods based on SL is carried out by using a training dataset, which contains example inputs as well as their expected target output (label). These input/target pairs are used during training to adjust the model, so that it can gradually give outputs more similar to the targets. Summarizing, the training dataset has the following structure: $D_{tr} = \{X, Y\} = \{(x_i, y_i)\}_{i=1}^{N}$, where N is the size of the dataset, x_i are the example inputs and y_i their corresponding labels $\forall i \in \{1, ..., N\}$. In **Unsupervised Learning**, however, the training dataset only contains example inputs, which are unlabeled. In this case, the models try to detect patterns, as common properties (clustering) or data representations, among the examples without any other information aside from the inputs. The training dataset used by these methods is like: $D_{tr} = \{X\} = \{x_i\}_{i=1}^{N}$, where N is the size of the dataset and x_i, $i \in \{1, ..., N\}$, are the example inputs.

Apart from this main division, there are other hybrid types of learning, namely Semi-supervised and Self-supervised Learning. **Semi-supervised Learning** is a mixture between Supervised and Unsupervised learning, where only a portion of the training example inputs in the dataset are labeled [52]. It can be seen as an extension of Supervised Learning, where non-labeled examples are also provided. In **Self-supervised Learning** (denoted as SSL in this project), the training dataset does not contain labels or target outputs, like in Unsupervised Learning. However, instead of detecting specific data patterns, SSL methods have a pretext task generator $\mathcal{P}$ that gets free available labels from the data. Then, these labels are used during training to solve a pretext task, in the line of Supervised Learning, to learn data representation [19]. Fig. 2.2 summarizes the ML types exposed.

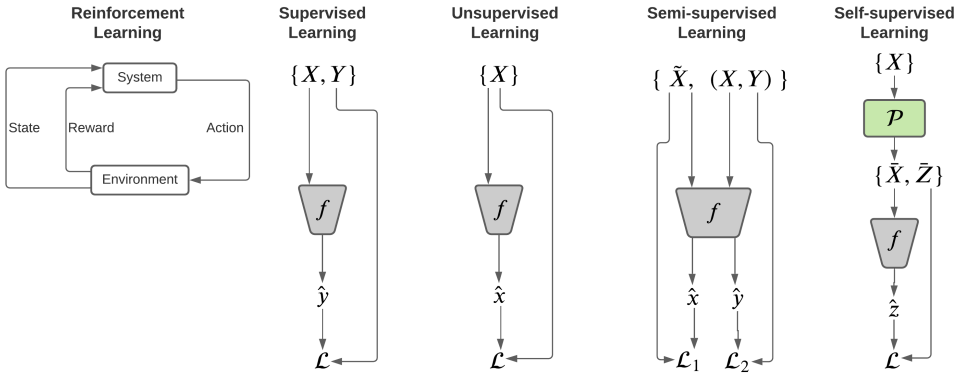


Figure 2.2: Comparison of different types of Machine Learning.

2.2 Self-supervised Learning

SSL aims to leverage large volumes of unlabeled data to learn effective data representation without relying on manual labels. It has been shown that, for many computer vision tasks, SSL is very useful and is closing the gap with other supervised methods that rely on a large amount of labeled data. Many studies have shown that Self-supervised strategies can be successfully applied to learn data representation in a great variety of data sources, such as images [12][64], videos [60], speech [5], text [17], or graphs [57]. Moreover, numerous works prove the potential of SSL to solve many downstream tasks, namely recognition [10], detection [27], or anomaly detection [24].

Regarding the image classification downstream task, the last works on SSL methods are based on the same basis: to learn representations that are invariant to image distortions, such as brightness, contrast or blurring changes. This goal is commonly achieved using different variants of Siamese networks [9][13]. Siamese networks are weight-sharing neural networks, each one receiving one input, used to compare output representations. Recent methods use two distorted views of an image as inputs and attempt to maximize the similarity of their latent representations (outputs). An inherent problem of this approach is the existence of trivial constant solutions (collapse problem), which must be avoided. Collapse occurs when the representation of all images is the same, so different images cannot be differentiated from their representations. To face this problem, different successful methods have been used. In Fig. 2.3 a summary with the latest model architectures proposed in the literature is shown and in Table 2.1 there is a benchmark of these methods under the linear evaluation protocol on ImageNet [15], with comparison to the related supervised method. Next, we introduce the models illustrated in Fig. 2.3, explaining their bases and contributions.

The first type of methods, avoid collapse using contrastive learning [30]. Let us consider positive and negative pairs of images, where positive pairs are images obtained from the same source image after a certain transformation and negative pairs are images from a different image source (or distorted views of two different images). In contrastive learning, both positive and negative pairs are compared to learn data representation. The idea is that the neural network "project" positive pairs as close as possible, since they come from the same source image, but repulse the negative ones. This methodology is widely spread in SSL combined with Siamese networks [11][31][62]. The procedure to produce negative pairs changes between methods, but all of them benefit from a high number of negative samples.

He et al. [31] presented a contrastive SSL method called MoCo. In MoCo, the negative samples are stored in a queue, which is progressively updated during training. In each step, a distortion view of each image in the current batch is added to the queue and the ones of the oldest batch in the queue are removed. Then, a new distortion view of each image in the current batch is computed and their embeddings are contrasted with the queue images' embeddings. Therefore, each sample is contrasted with one positive sample (another distorted view of the same image) and N negative samples (distorted views

from different images), where $N + 1$ is the length of the queue. The benefit of MoCo is that thanks to the queue, the number of negative samples is independent of the batch size. Furthermore, the Siamese networks do not share weights, since the branch used for the samples in the queue is a momentum encoder.

Chen et al. [11] designed another contrastive method named SimCLR. SimCLR shows that a queue is not needed to store contrastive samples. In contrast to MoCo, positive and negative samples are taken only from the current batch. From a batch of N images, 2 random augmented views are created from each one, resulting in $2N$ samples. The augmented view that comes from the same image is treated as a positive sample and the other $2(N-1)$ augmented examples as negatives. As the number of negative samples is directly proportional to the batch size, a large batch size is required in order for this method to work correctly.

Another way to prevent collapse is clustering. SwAV [10] is a SSL method that clusters the data representation enforcing that different augmentations of the same image are grouped together. Then, it contrasts the cluster assignment of multiple image views, instead of comparing their features. The clustering is done online by assigning a code to the feature representation of a sample. The possible codes are in a set of K prototype vectors, which is created in a such way that all the examples in a batch are equally partitioned by the prototypes. This equipartition is what prevents trivial constant solutions and it is created using the iterative Sinkhorn-Knopp algorithm [14].

A different approximation is defined in BYOL [29]. In BYOL, the branches of the Siamese network are called online network and target network, respectively. The online network uses an image view to predict the target network output of another image view of the same image. Unlike contrastive methods, BYOL does not use negative samples. The online branch has a predictor after the encoder and the target network is a momentum encoder, similar to SimCLR.

Chen and He [12] introduced a new method called SimSiam, where the model simply maximizes the similarity between two views of an image, without using negative pairs. None of the previous strategies for preventing collapse is used. Instead, to avoid non-desired solutions, an asymmetry is introduced between the branches of the Siamese network and the stop-gradient operation is applied to one branch. Both Siamese branches have a weight-sharing encoder, but on one the encoder is followed by a predictor and the weight update is only done in this branch.

Zbontar et al. [64] presented a recent new approach, called Barlow Twins. In this method, no asymmetry exists between Siamese branches and positive and negative sample representations are not contrasted. Instead, the method consists in creating two different distorted batches from an original batch, where each distorted batch contains the original batch's images with some distortions applied. Then, each network is fed with one distorted batch and the cross-correlation matrix (along features) between the outputs is computed. The objective is to approximate as much as possible the cross-correlation ma-

trix to the identity matrix. This provokes different distorted views of one image to have similar representations and that there is the minimum redundancy between the components of representation vectors.

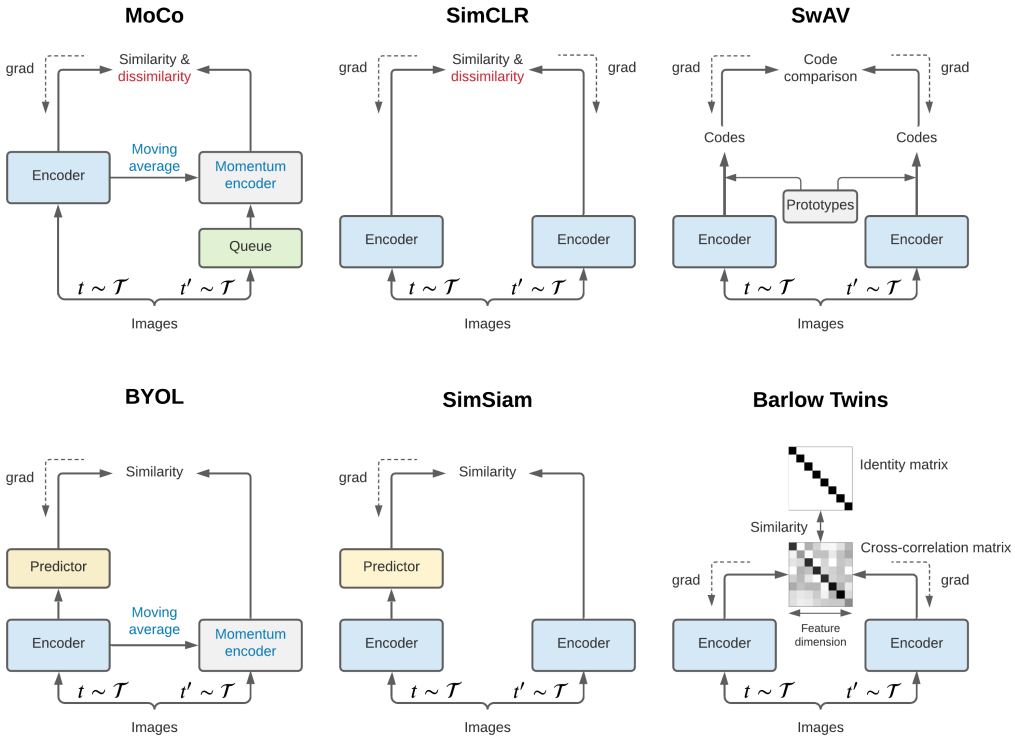


Figure 2.3: Architecture of novel SSL models for image representation.

Method	Top-1 acc. (%)	Top-5 acc. (%)
Supervised [32]	76.5	-
MoCo [31]	60.6	-
MoCo v2 [31]	71.1	90.1
SimCLR [11]	69.3	89.0
SwAV [10]	75.3	-
BYOL [29]	74.3	91.6
SimSiam [12]	71.3	-
Barlow Twins [64]	73.2	91.0

Table 2.1: Top-1 and top-5 accuracy of different SSL models under the linear evaluation protocol on ImageNet [15]. All models use a ResNet-50 [32] encoder. All the results are extracted from the original papers.

In Table 2.1 a benchmark of the presented methods under the linear evaluation protocol on ImageNet [15] is shown. The linear evaluation protocol consists of training a linear classifier on top of fixed representations of an encoder pre-trained with the SSL method. In the Table, the same encoder (ResNet-50 [32]) is used for all methods.

Deep neural networks have shown outstanding results in many computer vision tasks [32][47]. These results have generally been achieved under Supervised Learning protocols, when learning from big labeled datasets, like ImageNet[15]. However, Supervised Learning is meeting its own bottleneck. The amount of manual labeled data needed to continue outperforming the state-of-the-art methods is too high. The process of manual labeling is not trivial; it is very expensive, often difficult, and, in some scenarios, almost impossible. This barrier has motivated the research in SSL, the paradigm learning that instead of relying on manual labeled data, free inherent labels from the data are used to solve well-design pretext tasks. These labels are used as supervision to train a model (usually called encoder) for effective data representation. Finally, the entire encoder (or a part of it) can be leveraged to train a neural network for a downstream task in a supervision fashion, using far fewer labeled data than the conventional method. This learning paradigm is evolving very fast and in a large number of computer vision tasks, it is almost reaching the performance of traditional supervised methods [12][29][64].

Thanks to the great possibilities that SSL offers, many methods following this paradigm have been published in the recent literature [10][29][64]. As proposed in [46], Self-supervised methods can be organized according to their objectives in three main types with subcategories, as shown in Fig. 2.4.

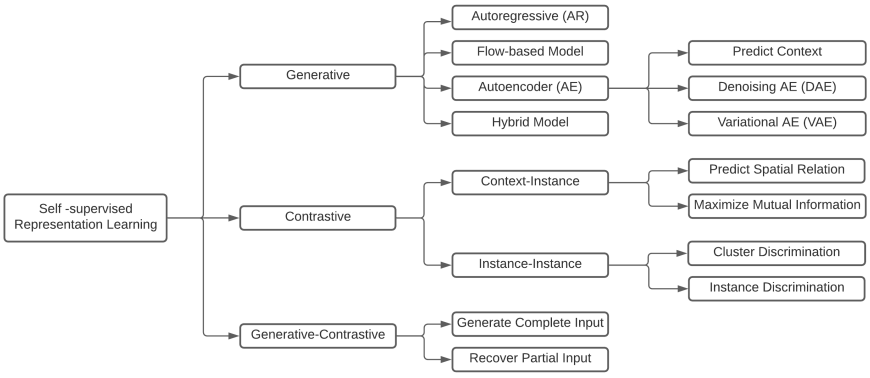


Figure 2.4: Organization of Self-supervised methods. Inspired by Liu et al. [46].

Generative SSL methods define pretext tasks that aim to reconstruct parts of the original input while learning effective latent representation for the data. Their architecture is based on training an encoder, which encodes an input x to a latent vector z, and a decoder, whose purpose is to make a reconstruction of x from z. Once the training is finished, the

encoder is often re-used for downstream tasks. Due to their good ability on recovering the original data distribution without assumptions, almost all the generation tasks for images, text, and audio involve a pre-training phase using generative SSL.

Contrastive SSL methods use contrastive pretext tasks that compare the similarity of inputs' representations and aim to get similar latent representations for similar inputs and dissimilar latent representations for non-similar inputs. Their basic architecture consists of training an encoder, that encodes the input x into a latent vector z, and a discriminator, which is often an MLP with 2 or 3 layers. However, a lot of modifications of this basic architecture can be found in the literature [12][31]. The encoder is usually re-used to solve visual tasks, such as image classification or object detection.

Generative-Contrastive methods define pretext tasks to generate fake samples from the input data and the objective is to distinguish them from real samples. These methods arose from the idea of generative SSL, but instead of relying on sample reconstruction, they learn to reconstruct the original data distribution by minimizing the distributional divergence. Their architecture is based on training an encoder and a decoder, which generate a fake sample $\hat{x}$ from an input x, and a discriminator, which is usually a big CNN (like a ResNet [32]), to distinguish the fake sample $\hat{x}$ from real samples.

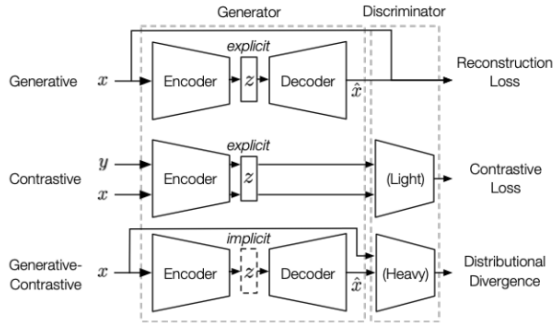


Figure 2.5: Architectures of Generative, Contrastive and Generative-Contrastive methods from top to bottom respectively. Inspired by [46].

In this project, where we use SSL to then solve the food recgonition downstream task, we focus on contrastive SSL. Particularly, on two of the state-of-the-art methods for that purpose exposed above, which are SimSiam and Barlow Twins.

2.3 Uncertainty Modeling

In the DL field, uncertainty refers to the confidence of a model's predictions and it comes from what the model has or has not been able to learn from the input data during the training stage. Over the last years, uncertainty has been a recurrent topic among re-

searchers, who have been working on analyzing and quantifying it [7][22][43]. Different studies [33][37] present the uncertainty in two different types: aleatoric uncertainty, which includes the noise inherent in the data, and epistemic uncertainty, which captures the lack of knowledge of the model. The first one is irreducible, but the second one is reducible and can be controlled by providing enough data. There are different approaches to model both types of uncertainties. The most important ones are based on Bayesian Neural Networks (BNN) [7][22], an ensemble of Neural Networks [43] and test-time augmentation [58].

The thesis of Gal [20], the thesis of Kendall [38] and the survey of Goan and Fokes [25] give a good introduction to BNN [16][49]. However, in practice, modeling exact BNN is intractable due to the huge number of parameters needed and some analytical constraints. Against this limitation, approximation techniques have been applied in the literature, being variational Bayesian methods [28][7][21] the most widespread. Amongst these approximations, the most popular technique to model uncertainty is the Monte Carlo Dropout (MC Dropout) [22]. The MC Dropout method was initially proposed by Gal and Ghahramani [22]. They claim that training a network with dropout layers and keeping dropout at inference time to take Monte Carlo samples of the prediction is a good approximation to BNN. This technique has been applied to many recent publications [36][51][2].

MC Dropout is widely spread in medical computer-assisted applications to model uncertainty since it is crucial to know the reliability of a model's prediction in safety-critical applications. Jungo et al. [36] applied the MC Dropout method to study the uncertainty in brain tumor cavity segmentation. The results showed the potential of modeling the model's uncertainty to validate the performance and introduce expert reviews when needed. Milanés-Hermosilla et al. [51] implemented the MC Dropout method to Motor Imagery (MI)-based Brain-Computer Interfaces (BCIs) field. They apply MC Dropout to Motor Imagery deep neural networks to quantify uncertainty and, from its analysis, enhance the models to obtain more reliable systems for real scenarios.

Regarding the food recognition task, Aguilar et al. [2] analyzed the uncertainty of a food classifier through the MC Dropout technique. From global and sample-level uncertainty studies, they take decisions and propose a new data augmentation approach to improve the performance of the model.

Owing to the increasing interest in quantifying and understanding uncertainty in a neural network's prediction, some recent studies reviewed and analyzed the different approaches used in the literature to model uncertainty. Abdar et al. [1] review recent advances in uncertainty quantification methods used in DL and discuss their most important applications in Reinforcement Learning, image processing, computer vision and medical image analysis. They also explain ongoing research challenges and explore possible future directions associated with the field. In [23], Gawlikowski et al. published a very extensive survey of uncertainty in deep neural networks. They provide an introduction and overview to uncertainty estimation in neural networks, explain the last advances in the field, present current challenges and analyze potential research opportunities.

Chapter 3

Methodology

In this chapter, we expose the terminology, the concepts and the theoretical bases of the project in detail. In Section 3.1, we introduce the background, giving an overview of convolutional neural networks. In Section 3.2, we expound the SSL fundamentals, focusing on contrastive SSL, the subtype of DL used in the project, and we describe the existent models used in the experimental part. Finally, in Section 3.3, we match the concept of uncertainty with neural networks, we give practical notions to model uncertainty and provide the most widely used metrics in literature to quantify the uncertainty.

3.1 Background

3.1.1 A brief overview of Convolutional Neural Networks

Artificial neural networks, or neural networks, are ML algorithms based on the behaviour of the human brain. They are the core of DL and are used to solve common problems in the field of Artificial Intelligence and Computer Vision, such as object recognition, object detection or object segmentation.

Deep feedforward networks, also known as feedforward neural networks or multilayer perceptrons (MLP), are a kind of neural networks that do not contain loops between their connections. The goal of a feedforward network is to approximate as much as possible some function $f^*(x)$ to resolve a task [26]. To obtain this purpose, the network defines a mapping $f(x;\theta)$, where θ is the set of parameters. Then, during a training phase, the parameters θ are tuned in search of the best approximation $f(x;\theta) \approx f^*(x)$. As an example, the target function on the image classification task is the function $f^*(x) = class(x)$, that maps the image x to its class y.

Since feedforward networks do not contain cycles, the function $f(x;\theta)$ can be modeled by a finite composition of functions:

$$f(x;\theta) = f_n(f_{n-1}((...f_2(f_1(x;\ \theta_1);\ \theta_2));\ \theta_{n-1});\ \theta_n),$$

where $\theta = (\theta_1, ..., \theta_n)$ and $n \in \mathbb{N}$.

Every function f_i is what is known as a layer of the network. The first layer f_1 is called input layer, the last layer f_n is called output layer and the layers f_i, $i \in \{2, ..., n-1\}$, are called hidden layers. The depth of the network is said to be n, according to the number of layers that the network has.

Convolutional neural networks (CNN) are a type of feedforward networks mainly used for processing structured arrays of data, such as images. They are the state-of-the-art for most visual tasks, namely image classification and image segmentation. They achieve such good performance by using a particular type of layer, called convolutional layer. CNNs combine convolutional layers with other well-known layers, such as pooling layers or fully-connected layers to configure the entire network. In Fig. 3.1, we show an example of the architecture of one CCN, called AlexNet [41].

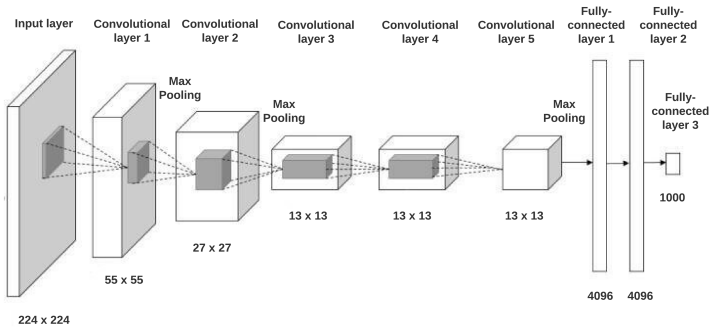


Figure 3.1: Architecture of AlexNet [41]. Example of one of the first CNNs introduced in the literature.

3.2 SSL fundamentals

From the SSL division exposed in Section 2.2, in this project we focus on contrastive SSL, in particular on instance discrimination SSL methods (Fig. 2.4), since they have an enormous potential for the food recognition downstream task, which is proved in the literature [12][64]. Therefore, in the next sections we dive into this type of SSL.

3.2.1 Contrastive SSL

From the joint distribution $P(X, Y)$ of input data X and target Y, it was believed that the only way to learn useful data representation was modeling $P(X|Y = y)$. However, as mentioned in Section 2.2, contrastive SSL methods focus on learning data representation by contrasting (or comparing) the similarity between inputs' representations, which

means that they try to model $P(Y|X = x)$. Interesting recent works on contrastive learning [31][11] showed and confirmed the potential of this approach.

The pretext task generator $\mathcal{P}$ of contrastive SSL methods, which is the process that obtains implicit labels from the input data, operates as follows:

Given a training dataset $D_{tr} = \{x_i\}_{i=1}^{N}$, $\mathcal{P}$ generates a new pseudo-labeled dataset

$$\overline{D}_{tr} = \mathcal{P}(D_{tr}) = \left\{ \left((x_{i_1}^1, ..., x_{i_n}^1), z_i^1 \right), ..., \left((x_{i_1}^j, ..., x_{i_n}^j), z_i^j \right) \right\}_{i=1}^{N}, \tag{3.1}$$

where $n, j \in \mathbb{N}$ and $\forall i \in \{1, ..., N\}$ $\forall k \in \{1, ..., n\}$ $\forall l \in \{1, ..., j\}$, $x_{i_k}^l$ are modified data points of any input example and z_i^l is the associated label to the group $(x_{i_1}^l, ..., x_{i_n}^l)$.

The pretext task generator is generally applied at the start of each training epoch. Then, the contrastive SSL model is trained with the new auto-labeled dataset $\overline{D}_{tr}$, relying on generated labels. As we can see in formulation 3.1, labels are not individual, but a set of modified data points that have a label. This is because the label keeps some relation between the data points in the set. This label will be used when contrasting the outputs of the data points in the loss function, as we will see later in detail. To make the process $\mathcal{P}$ more tangible and see how the formulation 3.1 acts in real cases, we provide two examples in the next sections. We give a short overview with an example of the first type of contrastive learning, **context-instance contrast**, and we go into more detail with the second type, **instance-instance contrast**, since it is the core of inspiration of our model.

Context-Instance Contrast

Context-instance contrastive methods aim to model the relationship between the representation of local features of a sample and the global representation. An example of context-instance contrastive method was proposed by Doersch et al. [18]. In this paper, the aim is to learn good image latent representations that enable to recognize objects and their parts. To this purpose, the pretext task designed for the Self-supervised method is to predict the relative position of patches within an image.

The pretext task generator $\mathcal{P}$ of this method acts in the following way: given the image training dataset $D_{tr} = \{x_i\}_{i=1}^{N}$, for each image x_i a first patch x_{i_1} is sampled, without any reference to the image. After that, knowing the position of the first patch, a second patch x_{i_2} is randomly sampled from the eight possible neighboring locations (*Top Left (1), Top Center (2), Top Right (3), Left (4), Right (5), Bottom Left (6), Bottom Center (7), Bottom Right (8)*). Finally, the label z_i assigned to (x_{i_1}, x_{i_2}) is the relative position of x_{i_2} with respective to x_{i_1}, as shown in Fig. 3.2.

Therefore, the pseudo-labeled dataset described in formulation 3.1 in this case is simplified with $j = 1$ and $n = 1$ and has the following structure:

$$\overline{D}_{tr} = \mathcal{P}(D_{tr}) = \{((x_{i_1}, x_{i_2}), z_i))\}_{i=1}^{N},$$

where x_{i_1} and x_{i_2} are random patches of the image x_i and z_i is the relative position of x_{i_2} with respect to x_{i_1}.

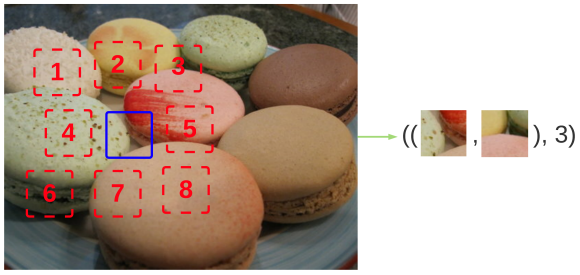


Figure 3.2: Process of the pretext task generator $\mathcal{P}$ in [18] applied to one image of the Food-101 dataset [8].

Instance-Instance Contrast

Instance-instance contrastive methods aim to model the relationship between sample-level representations. More formally, the pretext task of a contrastive SSL method is to learn data representations generated by a model $f(x;\theta)$ such that for any data point x

$$\Phi(f(x;\theta), f(x^+;\theta)) >> \Phi(f(x;\theta), f(x^-;\theta)),$$

where x^+ is any data point similar to x, x^- is any data point dissimilar to x and $\Phi(a,b)$ is a function that measures the similarity between a and b, called similarity function.

To achieve that goal, data augmentation takes an essential role in the pretext task generator to create pairs of similar and dissimilar data points. The basic framework of the pretext task generator $\mathcal{P}$ is described below.

There is a process $\mathcal{T}$ that given an input x returns a distorted/augmented view of x. Then, given the training dataset $D_{tr} = \{x_i\}_{i=1}^N$, for each image x_i two augmented views are generated, $x_i^a \sim \mathcal{T}(x_i)$ (called the anchor) and $x_i^+ \sim \mathcal{T}(x_i)$ (called positive sample). Furthermore, K augmented views from other K inputs, x_{i_k}, in the dataset are computed, $x_{i_k}^- \sim \mathcal{T}(x_{i_k})$ (called negative pairs). Finally, the pair (x_i^a, x_i^+) is labeled with 1 and pairs $(x_i^a, x_{i_1}^-)$, ..., $(x_i^a, x_{i_K}^-)$ are labeled with 0, where 1 means that both views in the pair are sampled from the same input and 0 means that they come from different inputs.

Therefore, the pseudo-labeled dataset described in formulation 3.1 can be re-written as:

$$\overline{D}_{tr} = \mathcal{P}(D_{tr}) = \{((x_i^a, x_i^+), 1), ((x_i^a, x_{i_1}^-), 0), ..., ((x_i^a, x_{i_K}^-), 0)\}_{i=1}^N,$$

where $K \in \mathbb{N}$, x_i^a and x_i^+ are distorted views of input x_i and $x_{i_k}^-$ are distorted views of inputs x_{i_k} with $x_{i_k} \neq x_i \; \forall i \in \{1, ..., N\} \; \forall k \in \{1, ..., K\}$. In Fig. 3.3 this procedure is exemplified for a group of images.

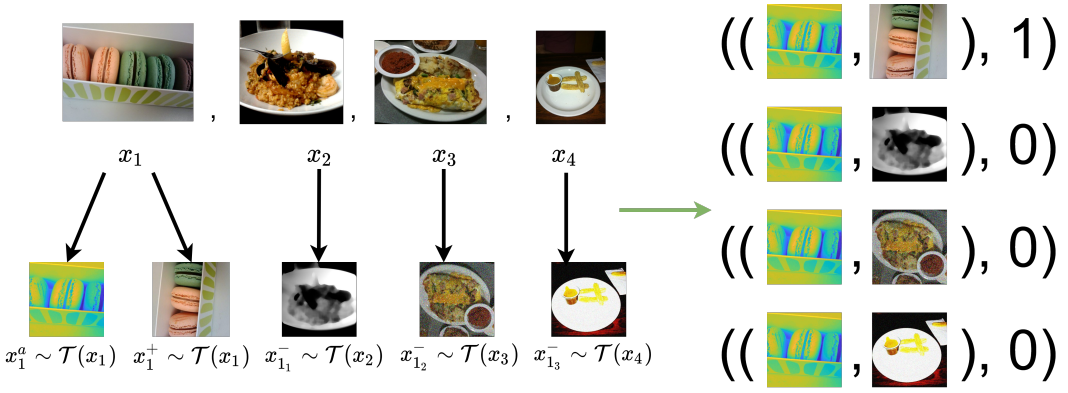


Figure 3.3: Process of the pretext task generator $\mathcal{P}$ in instance-instance contrastive methods applied to a group of images of the Food-101 dataset [8].

After applying $\mathcal{P}$, the representations of the augmented views are obtained: $z^a = f(x_i^a;\theta)$, $z^+ = f(x_i^+;\theta)$ and $z_{i_k}^- = f(x_{i_k}^-;\theta)$ $\forall k \in \{1,...,K\}$. Then, the system is trained to return similar representations for positive pairs and dissimilar representations for negatives, according to a similarity function Φ used in the loss function. A common contrastive loss function extended in the literature is inspired by the InfoNCE Loss [56], and can be written as:

$$\mathcal{L}_{con} = -\mathbb{E}\left[\log \frac{\Phi(z^a, z^+)}{\Phi(z^a, z^+) + \sum_{k=1}^{K} \Phi(z^a, z_{i_k}^-)}\right]. \tag{3.2}$$

Even though this is the basic framework, in literature many methods introduce variations in search of better performance and more efficient models. We can find variations in the family of transformations $\mathcal{T}$ used, in the similarity function Φ, how they sample the anchors, the positive and the negative samples and whether they use the same encoder for the anchor and contrastive samples [31][11].

Contrastive SSL methods have some problems when used for practical purposes. It has been shown that in order to learn useful representations, a very large number of negative samples need to be used in the loss function [11]. Therefore, new approaches (called non-contrastive methods in some contexts) focus on removing negative samples without losing effectiveness. As no negative samples are used to discriminate, such methods use regularisation techniques to avoid all outputs collapsing to a constant representation. One of these techniques is clustering, which is used in BYOL [29], but there are many other powerful novel techniques. In the next section, we describe the two state-of-the-art models explored in the experimental part, SimSiam [12] and Barlow Twins [64]. Both models can be considered non-contrastive SSL, because they do not contrast negative samples, but they introduce novel normalization mechanisms to avoid collapse.

3.2.2 SSL Models

In the literature, there are a large number of SSL models. In this project, where we are dealing with the food recognition problem, we explore two SSL methods introduced by Facebook AI Research (FAIR) group, called SimSiam [12] and Barlow Twins [64]. Before describing both methods, let us have a look at a common architecture used in contrastive SSL: Siamese networks.

Siamese Networks

Siamese networks were introduced in the late 20th century by Bromley et al. [9]. A Siamese network is formed by twin networks which receive different inputs, but are joint by an energy function (or loss function) at the top. This energy function computes some metric between the representations of each branch, which is often the similarity between the representations. The parameters of the branches are shared, or at least tied, and the network is usually symmetric, so the computation of the energy function remains invariant regardless of which branch is used for each one of the inputs [40]. This architecture is illustrated in Fig. 3.4.

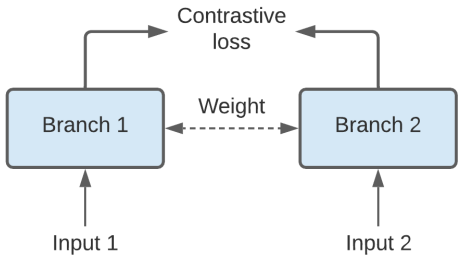


Figure 3.4: Basic architecture of a Siamese network.

SimSiam

SimSiam is a novel non-contrastive SSL method (contrastive SSL method that does not use negative pairs) published on November, 2020 by Chen and He [12]. As other contrastive SSL methods, SimSiam uses the structure of Siamese networks, but with a peculiar trait: there is an asymmetry between the Siameses. The model is formed by an encoder f, which in one branch is followed by a predictor h.

The encoder f consists of a backbone followed by a projection MLP head. In the original study, the backbone is a ResNet-50 [32] without the final classification layer and the projection MLP has 3 fully-connected (*fc*) layers. Each of the 3 layers has batch normalization applied, including the last one. In the first two layers (hidden layers), ReLU is applied

as activation function, unlike the last layer, which does not use any activation function. All the three layers have 2048 output units.

The predictor h, also referred as prediction MLP, has 2 fc layers. In this case, the first one has batch normalization applied and uses ReLU as activation function. Instead, neither batch normalization nor ReLU are applied in the second layer (output layer). The predictor h acts as a bottleneck structure, since the input and output dimensions of h are both 2048, but the hidden layer's output dimension is 512. In the original paper, it is reported that setting the hidden layer's output dimension to $1/4$ of the output dimension makes the structure more robust and stable. Putting it all together, the architecture of SimSiam is the one illustrated in Fig. 3.5.

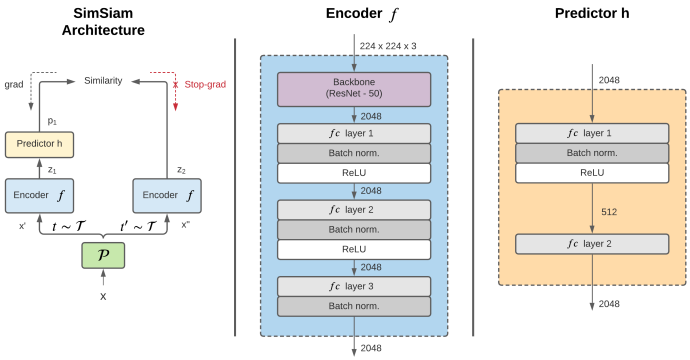


Figure 3.5: SimSiam architecture.

The outputs of the encoder and the predictor are used to compute the loss function and the backbone is used for downstream tasks. The pretext task generator $\mathcal{P}$ works in the following way: for each input image x in the training dataset D_{tr}, two augmented views x', x'' are generated using the transformation process $\mathcal{T}_{SimSiam}$ described below. Then, the same encoder f is applied to both images to get $z_1 = f(x')$ and $z_2 = f(x'')$. Next, and here is where the asymmetry appears, the predictor h is only applied on z_1, giving $p_1 = h(z_1)$. Finally, the similarity between p_1 and z_2 is computed (Fig. 3.5). The objective is to train the encoder and the predictor so that p_1 and z_2 are as much similar as possible for each image x.

The transformation process $\mathcal{T}_{SimSiam}$ consists in the application of the following techniques, in order: random resize cropping with scale in $[0.2, 1.0]$, random horizontal flipping, color jittering with brighness, contrast, saturation and hue strength of 0.4, 0.4, 0.4 and 0.1 respectively, grayscale conversion with a probability of 0.2 and blurring augmentation with a Gaussian kernel with std in $[0.2, 1.0]$.

The similarity function used to measure the sameness between the representations

$p_1 = h(f(x'))$ and $z_2 = f(x'')$ is the cosine similarity, which is:

$$C_s(p_1, z_2) = \frac{p_1 \cdot z_2}{\|p_1\|_2 \, \|z_2\|_2},$$

where $\cdot$ denotes the dot product between the vectors and $\|\cdot\|_2$ denotes the l_2-norm. Cosine similarity ranges from -1 (which means that the vectors are exactly opposite) to 1 (which means total correlation between the vectors).

In fact, in practice, during training the Siamese branches are also fed with the inputs exchanged. It means that $p_2 = h(z_2) = h(f(x''))$ and $z_1 = f(x')$ are also considered to learn representations. Then, the target function to maximize for each image x in the dataset is the symmetrized function:

$$F(x) = \frac{1}{2} C_s(p_1, z_2) + \frac{1}{2} C_s(p_2, z_1), \tag{3.3}$$

where $z_1 = f(x')$, $z_2 = f(x'')$, $p_1 = h(z_1)$, $p_2 = h(z_2)$ and $x', x'' \sim \mathcal{T}_{SimSiam}(x)$.

The training is done using batches of images in the dataset and a loss function over each batch is used to optimize the goal. In this case, the loss function used is just the average of $-F(x)$ over all the images in the batch. Formally, if $B = \{x_1, ..., x_N\}$ is a batch of N images, the loss function used in SimSiam is:

$$\mathcal{L}_{SimSiam} = -\frac{1}{N} \sum_{i=1}^{N} F(x_i). \tag{3.4}$$

This loss function is nothing else than the mean of negative cosine similarities, so it ranges from -1 to 1. Low values mean that distorted views of the same image have very similar representations according to cosine similarity. Nevertheless, from a theoretical point of view, this loss function may lead to problems. Imagine that f and h learn to map every input to a constant vector $\vec{v}$. Then, for any batch of images, we would get that $\mathcal{L} = -1$, since for each image x, $F(x) = 1$. In this case, the minimum value of the loss function would be reached what might mean that the objective has been optimized. Nevertheless, this solution for f and h is not desired, because they are sending all images to the same representations, which means that also dissimilar views have the same representation. This problem is known as "collapse".

While developing the method, the authors experienced that the collapsing problem was present, but they introduced an original solution that empirically proved to solve it. This solution, called *stop-gradient*, is applied to some variables that contribute to the loss function and when applying it, no gradient is backpropagated along the variable.

This method is applied to the variables z_1 and z_2 used in the computation of the loss function. Therefore, the function $F(x)$ defined in Equation 3.3 used to compute the loss function in Equation 3.4, in practice, is used as follows for all input image x in the batch:

$$F(x) = \frac{1}{2} C_s(p_1, \textit{stop-gradient}(z_2)) + \frac{1}{2} C_s(p_2, \textit{stop-gradient}(z_1)). \tag{3.5}$$

This modification implies that z_1 and z_2 are treated as constants, which means that the backpropagation method is only carried out through the variables p_1 and p_2, as shown in Fig. 3.5. The encoder f will not receive gradients from z_1 and z_2, but it will receive them from p_1 and p_2 to update the weights.

Barlow Twins

Zbontar et al. [64] published the Barlow Twins SSL on May, 2021. It is a contrastive SSL method which instead of contrasting the data representations using a variant of the InfoNCE Loss (Equation 3.2), it proposes a loss function based on Horace Barlow's redundancy-reduction principle [6] applied to a Siamese network. The model consists of an encoder and a projector, abbreviated as f.

The encoder is a ResNet-50 [32] with the last fully-connected layer cut and is followed by a projector. The projector is an MLP with three fully-connected layers, each one with 8192 output units. The first two layers (hidden layers) have batch normalization applied and are followed by a ReLU activation layer. The outputs of the projector are used in the loss function and the encoder is then used for downstream tasks. The Fig. 3.6 illustrates the architecture of Barlow Twins.

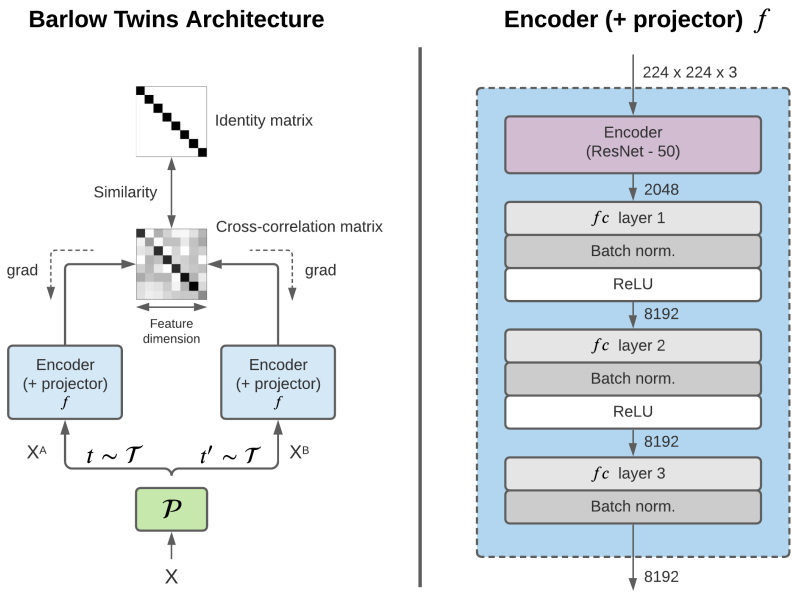


Figure 3.6: Barlow Twins architecture.

To understand how the redundancy reduction is applied and how the method works, we will not interpret that the model is fed with only one image x, but with a batch X

of images, $X = \{x_i\}_{i=1}^N$. First of all, the pretext task generator $\mathcal{P}$ creates two augmented views, x_i^A and x_i^B, for each image x_i in the batch X, using the transformation process $\mathcal{T}_{BT}$ (which is exactly equal to the one defined in Section 4.3), producing two batches of distorted images, $X^A = \{x_i^A\}_{i=1}^N$ and $X^B = \{x_i^B\}_{i=1}^N$. Then, each twin branch f of the network is fed with one distorted batch, obtaining $Z^A = f(X^A)$ and $Z^B = f(X^B)$. Finally, Z^A and Z^B are used to compute the loss function.

We can assume that:

$$Z^A = \begin{bmatrix} z_{11}^A & z_{12}^A & \cdots & z_{1D}^A \\ z_{21}^A & z_{22}^A & \cdots & z_{2D}^A \\ \vdots & \vdots & \ddots & \vdots \\ z_{N1}^A & z_{N2}^A & \cdots & z_{ND}^A \end{bmatrix}, \; Z^B = \begin{bmatrix} z_{11}^B & z_{12}^B & \cdots & z_{1D}^B \\ z_{21}^B & z_{22}^B & \cdots & z_{2D}^B \\ \vdots & \vdots & \ddots & \vdots \\ z_{N1}^B & z_{N2}^B & \cdots & z_{ND}^B \end{bmatrix},$$

where $\forall i \in \{1, ..., N\}$, $(z_{i1}^A, z_{i2}^A, ..., z_{iD}^A) = f(x_i^A)$ and $(z_{i1}^B, z_{i2}^B, ..., z_{iD}^B) = f(x_i^B)$ are the representations of the two distorted views of the image x_i and D is the dimension of the representations. In the original implementation, as said before, $D = 8192$.

Next, let us consider the loss function used during the training phase. To simplify notations, we assume that Z^A and Z^B are mean-centered for columns, that is to say that each feature of the representations has mean 0 over the batch. The first step is the computation of the cross-correlation matrix C of the outputs along the batch dimension, where:

$$C = (c_{ij})_{1 \leq i,j \leq D}, \; c_{ij} = \frac{\sum_{b=1}^N z_{bi}^A z_{bj}^B}{\sqrt{\sum_{b=1}^N (z_{bi}^A)^2} \sqrt{\sum_{b=1}^N (z_{bj}^B)^2}} \; \forall i, j \in \{1, ..., D\}.$$

We can see that each element c_{ij} of C is the cosine similarity between the column i of Z^A and the column j of Z^B. Therefore, each element of C ranges between -1 (perfect anti-correlation) and 1 (perfect correlation). Then, the loss function of Barlow Twins is defined as:

$$\mathcal{L}_{BT} = \sum_{i=1}^D (1 - c_{ii})^2 + \lambda \sum_{i=1}^D \sum_{j=1, j \neq i}^D c_{ij}^2, \tag{3.6}$$

where $\lambda \in \mathbb{R}_{>0}$ is a constant that balances the importance between the first and the second term of the loss function. The first term of the loss is called *invariance term* and the second one *redundancy reduction term*.

Barlow Twins' loss function can be compared with the prototypical loss function used in contrastive SSL, InfoNCE Loss (Equation 3.2). The *invariance term* acts as the contrast of positive pairs and the *redundancy reduction term* plays a similar role to the contrast of negative samples. Nevertheless, Barlow Twins directly compares features instead of image representations.

If we analyze the loss $\mathcal{L}_{BT}$ in detail, we can see that its purpose is to approximate the cross-correlation matrix C as much as possible to the identity matrix, since diagonal

elements are forced to go to 1 (*invariance term*) and off-diagonal elements are forced to go to 0 (*redundancy reduction term*).

The *invariance term* aims to make the representations invariant to the distortions applied. Each element c_{ii} of C is the similarity between the vectors $(z^A_{1i}, z^A_{2i}, ..., z^A_{Ni})$ and $(z^B_{1i}, z^B_{2i}, ..., z^B_{Ni})$. Therefore, it contrasts the i^{th} representation feature of each positive pair (x^A_i, x^B_i). Trying to approximate c_{ii} to 1 can be understood as making the i^{th} feature of the embeddings invariant to distortions.

In contrast, the *redundancy reduction term* aims to decorrelate as much as possible the features of the embeddings, so as to reduce the redundancy between features. Each element $c_{ij}, i \neq j$ of C is the similarity between the vectors $(z^A_{1i}, z^A_{2i}, ..., z^A_{Ni})$ and $(z^B_{1j}, z^B_{2j}, ..., z^B_{Nj})$. Therefore, for each positive pair (x^A_i, x^B_i), it contrasts the i^{th} representation feature of image x^A_i with the j^{th} representation feature of image x^B_i. Trying to approximate c_{ij} to 0 can be understood as decorrelating as much as possible the i^{th} and j^{th} features for distortions.

3.3 Uncertainty Estimation and Modeling

Over the last years, the use of neural networks in science and real world applications has experienced an exponential growth due to the exceptional results they have shown for many problems in the field of AI. Though not exclusively, they are often used for predicting, and predictions are associated with uncertainty. Model assumptions and the inductive learning method used (learning from training data and generalize) lead to hypothetical and uncertain predictions.

However, in practice, when conventional neural networks return a prediction, they are not capable of modeling the reliability of the outcome. Imagine a neural network trained to classify food images into four classes: $\{Apple, Bread, Pizza, Sushi\}$. When predicting the class of an image, it computes the probability of each class and the class with highest probability becomes the prediction. The model will always return a prediction, even if the image is not from any class or it is not clear to which class it belongs. However, if the neural network could quantify the uncertainty of its predictions, it would probably detect that some predictions are not reliable.

Even though the probability over the classes could be interpreted as an uncertainty estimation, it has been shown that it often overestimates its confidence, even on data divergent from the dataset used for training. From this point, the need arises to estimate the uncertainty of models' predictions, to give more flexibility on taking human actions in safety-critical domains.

3.3.1 Types of Uncertainty

Depending on the source, uncertainty can be divided into two types: aleatoric and epistemic [39][33]. **Aleatoric uncertainty** refers to the notion of how inherent noise in the data can affect to the outcome of a model. **Epistemic uncertainty** refers to the lack of knowledge of the model, that is to say, what the model has not been able to learn from the data.

Aleatoric uncertainty is irreducible, as it is related to the data itself. On the other hand, epistemic uncertainty can be reduced by providing the model with additional information. To encourage this claim, let us see both concepts in simple real world applications. Firstly, consider the problem of predicting the coin flipping. The process to generate training data in this problem has a stochastic component that cannot be reduced. Now, consider the problem that consists to answer to the following question: "What does *'poma'* mean in Catalan: apple or pear?" There are also two possible answers and the uncertainty of the answer can be the same than in the first example. However, in this case, the uncertainty can be reduced if we have enough information, such as a dictionary. In the first case, the uncertainty comes from the data, thus, it is aleatoric uncertainty. In the second case, uncertainty comes from lack of knowledge, so, it is epistemic uncertainty.

In DL, it is important to understand how uncertain a model is with its predictions. To this purpose, both types of uncertainty have to be considered, as all of them affect to the final predictions. Therefore, aleatoric and epistemic uncertainty are used to induce **predictive uncertainty**, defined as the confidence of a prediction.

3.3.2 Bayesian Deep Learning

Bayesian DL is a field that mixes DL and Bayesian probability theory. The idea of Bayesian DL is to add a prior distribution over the parameters of a neural network rather than using deterministic weights, so that the outcomes are also distributions instead of deterministic predictions. In this way, uncertainty can be estimated. Such neural networks are called Bayesian Neural Networks (BNN) [16][49].

Next, we will give a mathematical perspective of BNNs. Let $D_{tr} = \{X, Y\} = \{(x_i, y_i)\}_{i=1}^{N}$ be a training dataset, where x_i are the input data and y_i their corresponding labels $\forall i \in \{1, ..., N\}$. The aim of a deterministic neural network is to optimize a function $f^*(x)$ by tuning the parameters θ of the network. Bayesian neural networks, however, define a prior distribution over the model parameters, $p(\theta)$, and a model likelihood, $p(y|x, \theta)$. As an example, in a classification task with k classes, the softmax likelihood is usually used as the model likelihood:

$$p(y = i|x, \theta) = \frac{\exp(f_i(x; \theta))}{\sum_{j=1}^{k} \exp(f_j(x; \theta))}, \ \forall \ i \in \{1, ..., k\}.$$

The posterior distribution is the distribution of the model's parameters given the training dataset. By assuming independence between the parameters θ and the training dataset

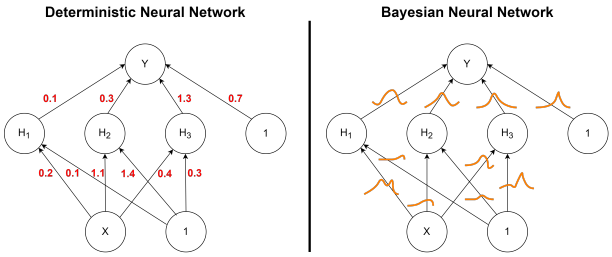


Figure 3.7: Comparison between a deterministic Neural Network (left) and a Bayesian Neural Network (right). The Bayesian Neural Network adds a prior distribution over the parameters.

$\{X, Y\}$, and applying Bayes' Theorem, it can be written as:

$$p(\theta|X,Y) = \frac{p(Y|X,\theta)p(\theta)}{p(Y|X)},$$

where $p(Y|X,\theta)$ is the model likelihood, $p(\theta)$ is the prior distribution and $p(Y|X)$ is the model evidence (the probability of the data).

Assuming that the posterior distribution is known, at inference time the prediction y^* for a test example x^* is done considering all the possible weights, weighted by their probability. Therefore, the predictive distribution is:

$$p(y^*|x^*,X,Y) = \int p(y^*|x^*,\theta,X,Y)p(\theta|X,Y)d\theta = \int p(y^*|x^*,\theta)p(\theta|X,Y)d\theta,$$

where $p(y^*|x^*,\theta)$ is the model likelihood and $p(\theta|X,Y)$ is the posterior distribution. Note that in the second equality, we assumed that the prediction y^* is independent of the training data (X,Y) given parameters θ.

The problem here is that the posterior $p(\theta|X,Y)$ cannot usually be evaluated analytically. Thus, the predictive distribution cannot be determined. To overcome this drawback, a large number of alternatives have been proposed in the literature, most of them based on variational inference. These methods focus on finding an approximating of the posterior distribution by a variational distribution $q_\omega(\theta)$, parametrised by ω that minimizes the Kullback-Leibler (KL) divergence [42]:

$$q^*_\omega(\theta) = \underset{q_\omega(\theta)}{argmin}\ KL\Big(q_\omega(\theta)\ ||\ p(\theta|X,Y)\Big). \tag{3.7}$$

Then, the predictive distribution can be approximated by:

$$p(y^*|x^*,X,Y) = \int p(y^*|x^*,\theta)p(\theta|X,Y)d\theta \approx \int p(y^*|x^*,\theta)q^*_\omega(\theta)d\theta. \tag{3.8}$$

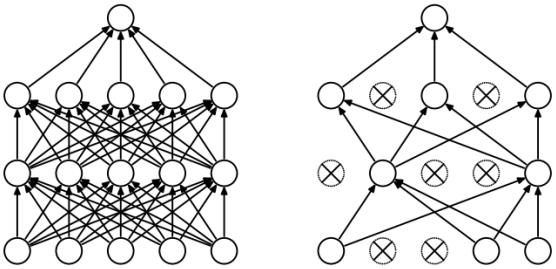
Gal and Ghahramani [22] published a variational inference stochastic approach to BNN using dropout [54] simple to implement that allows to model uncertainty in a straightforward manner. This method is known as Monte Carlo Dropout and is the most widespread

technique in the literature to approximate BNNs and model uncertainty in DL. This is the technique that will be used in the experimental section to model the uncertainty of our models, so it is described in detail in the next section.

3.3.3 Monte Carlo Dropout as a Bayesian approximation

Dropout

Dropout was introduced by Srivastava et al. [54] as a regularization technique to deal with the overfitting problem in neural networks. Deep neural networks with a large number of parameters tend to co-adapt the parameters too much on the training data, giving generalization problems at inference time. When dropout is applied to a layer of the network, some units of the layer are randomly dropped out during training, which prevents parameters from adjusting too much on the training data. The Dropout technique has a hyper-parameter called dropout rate, which is the fraction of units to randomly drop out in a forward pass.



(a) Standard Neural Net (b) After applying dropout.

Figure 3.8: Application of dropout technique to a neural network. **Left**: a standard neural network with 2 hidden layers. **Right**: application of dropout to the first 3 layers of the network on the left. Inspired by [54].

Monte Carlo Dropout

The Monte Carlo Dropout (MC Dropout) technique was introduced by Gal and Ghahramani [22]. It suggests using dropout for another purpose rather than preventing overfitting, which is to compute uncertainty. In their paper, they showed that training a network applying dropout after each layer of weights is equivalent to minimize the Kullback-Leibler divergence in Equation (3.7). Moreover, they extended the dropout application, keeping it at inference time to be able to compute uncertainty.

The workflow of MC Dropout is the following: Firstly, a neural network is trained with the dropout technique. Then, at inference time (or evaluation) dropout is still applied. For a single test input x^*, T network's outputs are taken. While using dropout, a different subset of the neural network's weights θ_t will be used to compute each output, giving T different outputs $p(y^*|x^*, \theta_t)$. The different weight subsets used for each

computation represent a stochastic sample of the variational distribution $q_\omega(\theta)$, which is the approximation of the posterior distribution $p(\theta|X,Y)$ that minimizes the Kullback-Leibler divergence. Therefore, as each output is computed through a stochastic sample of $q_\omega(\theta)$, they can be considered as samples of the predictive distribution $p(y^*|x^*,X,Y)$. From these observations, the predictive distribution approximated in Equation (3.8) can be re-approximated as:

$$p(y^*|x^*,X,Y) \approx \int p(y^*|x^*,\theta) q^*_\omega(\theta) d\theta \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|x^*,\theta_t),$$

where T is the number of forward passes through the network and θ_t is the subsample of weights used in the forward pass t, $\theta_t \sim q^*_\omega(\theta)$.

With this approximation of the predictive distribution $p(y^*|x^*,X,Y)$, which can be easily modeled in practice, it is possible to approximately quantify different uncertainty measures, as we will discuss in the next section.

3.3.4 Uncertainty Quantification

In this section, we propose and discuss three uncertainty measures: variance, predictive entropy and mutual information. These three uncertainty measures quantify different types of uncertainty in DL, so all of them are interesting to model. For all of them, we provide a description and explicit formulae for practical application, using a Monte Carlo Dropout technique. As MC Dropout provides an estimation of the predictive distribution $p(y^*|x^*,X,Y)$, the quantification of the uncertainty measures will also be approximated. Next, we provide some assumptions and notations that will be used in the measures descriptions.

We assume that the predictive distribution is over C classes (for instance, to deal with a classification problem) and that we use T MC Dropout samples at inference. To simplify the notation, for an input x^*, we denote the sampled distributions $p(y^*|x^*,\theta_t)$ as $p_t[x^*]$, and the probability of the c^{th} class as $p_{tc}[x^*]$. We also denote the mean predictive distribution

$$p(y^*|x^*,X,Y) = \frac{1}{T}\sum_{t=1}^{T} p(y^*|x^*,\theta_t)$$

as $\hat{p}[x^*]$, and the mean probability of the c^{th} class as $\hat{p}_c[x^*]$.

We assume that the model has been trained with a training dataset $D_{tr} = \{X,Y\} = \{(x_i,y_i)\}_{i=1}^N$, where x_i are the input data and y_i their corresponding labels $\forall i \in \{1,...,N\}$.

Finally, we denote an arbitrary dataset as $D = \{X^*,Y^*\} = \{(x^*_i,y^*_i)\}_{i=1}^M$, where x^*_i are the input data and y^*_i their corresponding labels $\forall i \in \{1,...,M\}$.

Variance

Variance is an uncertainty measure that estimates how different several network's outcomes for a single input x^* are. Therefore, this measure describes how much stable is the model. If variance is high, it means that the same output can give distant outputs depending on the subset of network's weights used. Thus, the network is not stable, because little changes produce very different predictions.

Given an input x^*, the variance measure can be computed as the mean variance across the classes:

$$var[y^*|x^*, X, Y] = \frac{1}{C}\sum_{c=1}^{C}\frac{1}{T}\sum_{t=1}^{T}(p_{tc}[x^*] - \hat{p}_c[x^*])^2.$$

For simplicity, we will denote $var[y|x^*, X, Y]$ as $var(x^*)$. With this notation, the variance of a dataset D can be defined as the mean variance across the samples in the dataset:

$$var(D) = \frac{1}{M}\sum_{i=1}^{M} var(x_i).$$

Predictive Entropy

This uncertainty measure has its foundations in information theory. It estimates how much information contains the model's predictive distribution. The predictive entropy achieves the maximum value when all classes are predicted with an equal probability, while it goes to the minimum value when one class has a probability of 1 and the other classes of 0. Therefore, it measures how certain the network is about its predictions.

Originally, the entropy of a discrete random variable z is defined as [53]:

$$H(z) = -\sum_{k=1}^{K} P(z = k) \cdot \log(P(z = k)),$$

where K is the number of possible values for z and $P(z = k)$ is the probability of z being of class k.

In our case, the predictive entropy is computed through the mean predictive distribution. Given an input x^*, it is approximated as:

$$\tilde{H}[y^*|x^*, X, Y] = -\sum_{c=1}^{C} \hat{p}_c[x^*] \cdot \log(\hat{p}_c[x^*]).$$

With this approximation, the predictive entropy of a dataset D can be defined as the mean predictive entropy across the samples in the dataset:

$$\tilde{H}(D) = \frac{1}{M}\sum_{i=1}^{M} \tilde{H}[y^*|x_i^*, X, Y].$$

Mutual Information

Mutual Information (MI) models the mutual information between the model's posterior distribution $p(\theta|X,Y)$ and the predictive distribution $p(y^*|x^*,X,Y)$. Test inputs with a high mutual information are inputs on which the model is uncertain on average. It is important to quantify this measure, because in the model there can be parameters that erroneously give predictions with high confidence.

Given an input x^*, the mutual information between the posterior distribution and the predictive distribution is defined as:

$$MI[y^*,\theta|x^*,X,Y] = H[y^*|x^*,X,Y] - \mathbb{E}_{p(\theta|X,Y)}\Big[H[y^*|x^*,\theta]\Big].$$

Using an MC Dropout technique, it can be approximated through the approximation of the predictive entropy as follows:

$$\tilde{MI}[y^*,\theta|x^*,X,Y] = -\sum_{c=1}^{C} \hat{p}_c[x^*] \cdot \log(\hat{p}_c[x^*]) + \frac{1}{T}\sum_{t=1}^{T}\sum_{c=1}^{C} p_{tc}[x^*] \cdot \log(p_{tc}[x^*]).$$

Just like with the previous uncertainty measures, we can compute an approximation of the mutual information of a dataset D by averaging the mutual information across the samples in the dataset:

$$\tilde{MI}(D) = \frac{1}{M}\tilde{MI}[y^*,\theta|x^*,X,Y].$$

Chapter 4

Our proposal: OptSSL - A New Method for SSL

In this chapter, we propose a new method for SSL inspired in the architecture of SimSiam, but with a new loss function and different data augmentation techniques. In Section 4.1, we explain in what way OptSSL is an extension of SimSiam and what similarities and differences exist between them. In Section 4.2, we propose a novel loss function for the learning representation task using SSL. Finally, in Section 4.3 we expose the data augmentation techniques used in our proposed method to obtain the augmented views of the images in the dataset.

4.1 OptSSL: an extension of the SimSiam

OptSSL's architecture is inspired by SimSiam's one (see Section 3.2.2). As SimSiam, OptSSL uses an asymmetric Siamese network structure. Both branches have an encoder f, but in one branch it is followed by a predictor h. We keep the same configuration for both encoder and predictor and use the stop-gradient operation. However, we use a new loss function that empirically proves to give better results than SimSiam's one and introduce new data augmentation techniques to generate the distorted views of the images. Fig. 4.1 illustrates the architecture of our model.

4.2 OptLoss Loss function - a Novel Loss Function for SSL

The loss function of our method, called optLoss, contrasts positive and negative samples. In some way, it can be understood as an extension of SimSiam's loss function adding also negative sample contrasting. To define the loss function, let us assume that the model is fed with a batch X of images, $X = \{x_i\}_{i=1}^{N}$. The pretext task generator $\mathcal{P}$ creates two augmented views, x_i^A and x_i^B, for each image x_i in the batch X, using the data augmentation process $\mathcal{T}_{Opt}$ defined in Section 4.3, producing two batches of distorted images, $X^A = \{x_i^A\}_{i=1}^{N}$ and $X^B = \{x_i^B\}_{i=1}^{N}$. Then, the branches of the network are fed with the

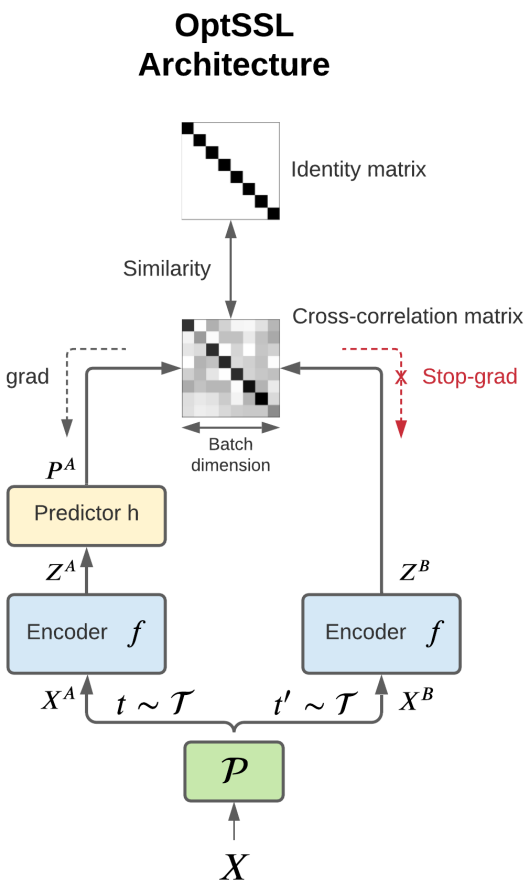


Figure 4.1: OptSSL architecture.

distorted batches, obtaining $P^A = h(f(X^A))$ and $Z^B = f(X^B)$. Furthermore, the network is also fed with the inputs exchanged, which gives $P^B = h(f(X^B))$ and $Z^A = f(X^A)$.

We can assume that:

$$Z^A = \begin{bmatrix} z^A_{11} & z^A_{12} & \cdots & z^A_{1D} \\ z^A_{21} & z^A_{22} & \cdots & z^A_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ z^A_{N1} & z^A_{N2} & \cdots & z^A_{ND} \end{bmatrix}, \; Z^B = \begin{bmatrix} z^B_{11} & z^B_{12} & \cdots & z^B_{1D} \\ z^B_{21} & z^B_{22} & \cdots & z^B_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ z^B_{N1} & z^B_{N2} & \cdots & z^B_{ND} \end{bmatrix},$$

$$P^A = \begin{bmatrix} p^A_{11} & p^A_{12} & \cdots & p^A_{1D} \\ p^A_{21} & p^A_{22} & \cdots & p^A_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ p^A_{N1} & p^A_{N2} & \cdots & p^A_{ND} \end{bmatrix}, \; P^B = \begin{bmatrix} p^B_{11} & p^B_{12} & \cdots & p^B_{1D} \\ p^B_{21} & p^B_{22} & \cdots & p^B_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ p^B_{N1} & p^B_{N2} & \cdots & p^B_{ND} \end{bmatrix},$$

where $\forall i \in \{1, ..., N\}$, $(z^A_{i1}, z^A_{i2}, ..., z^A_{iD}) = f(x^A_i)$, $(z^B_{i1}, z^B_{i2}, ..., z^B_{iD}) = f(x^B_i)$, $(p^A_{i1}, p^A_{i2}, ..., p^A_{iD}) = h(f(x^A_i))$, $(p^B_{i1}, p^B_{i2}, ..., p^B_{iD}) = h(f(x^B_i))$ are the representations of the two distorted views of the image x_i and D is the dimension of the representations. In our original implementation, we used the same value for D as in SimSiam, $D = 2048$.

The loss function aims to contrast these representations, to learn in such a way that the distorted views of the same image have similar representations and distorted views of different images have dissimilar representations. For this purpose, the cosine similarity between each row of Z^A and each row of P^B is computed, as well as the cosine similarity between each row of Z^B and each row of P^B. As a result, we get two cross-correlation matrices, $C^{(1)}$ and $C^{(2)}$, computed along the dimension of the features. $C^{(1)}$ and $C^{(2)}$ are:

$$C^{(1)} = (c^{(1)}_{ij})_{1 \leq i,j \leq N}, \; c^{(1)}_{ij} = \frac{\sum_{d=1}^{D} z^A_{id} p^B_{jd}}{\sqrt{\sum_{d=1}^{D} (z^A_{id})^2} \sqrt{\sum_{d=1}^{D} (p^B_{jd})^2}} \; \forall i, j \in \{1, ..., N\},$$

$$C^{(2)} = (c^{(2)}_{ij})_{1 \leq i,j \leq N}, \; c^{(2)}_{ij} = \frac{\sum_{d=1}^{D} z^B_{id} p^A_{jd}}{\sqrt{\sum_{d=1}^{D} (z^B_{id})^2} \sqrt{\sum_{d=1}^{D} (p^A_{jd})^2}} \; \forall i, j \in \{1, ..., N\}. \tag{4.1}$$

We can see that each element $c^{(1)}_{ij}$ of $C^{(1)}$ is the cosine similarity between the row i of Z^A and the row j of P^B, and each element $c^{(2)}_{ij}$ of $C^{(2)}$ is the cosine similarity between the row i of Z^B and the row j of P^A.

From this two cross-correlation matrices, the OptSSL's loss function, called OptLoss, is defined as:

$$\mathcal{L}_{OptSSL} = \sqrt{\frac{1}{2}\Big(\frac{1}{N}\sum_{i=1}^{N}(1 - c^{(1)}_{ii})^2 + \frac{1}{N}\sum_{i=1}^{N}(1 - c^{(2)}_{ii})^2\Big)} + \lambda\sqrt{\frac{1}{2}\Big(\frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1, j\neq i}^{N}(c^{(1)}_{ij})^2 + \frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1, j\neq i}^{N}(c^{(2)}_{ij})^2\Big)}, \tag{4.2}$$

where $\lambda \in \mathbb{R}_{>0}$ is a constant that balances the importance between the first and the second term of the loss function.

Next, in order to motivate and explain each part of this new loss function proposal, we provide a discussion, going from its elemental parts into its final definition.

Discussion on our OptLoss

As discussed above, the i^{th} diagonal element of the cross-correlation matrix $C^{(1)}$ (Equation 4.2) is the cosine similarity between the row i of Z^A and the row i of P^B. In other words, it is the cosine similarity between the predictor representation of the distorted view x_i^A and the encoder representation of the distorted view x_i^B. Therefore, as x_i^A and x_i^B are positive samples, we want both representations to be as much similar as possible, or what is the same, that $c_{ii}^{(1)}$ is as much closer as possible to 1. Analogously, we also want each diagonal element of $C^{(2)}$ to be as much closer as possible to 1. We force the diagonal elements of $C^{(1)}$ and $C^{(2)}$ to go to 1 with the following terms in the loss function:

$$(1 - c_{ii}^{(k)})^2,\ k \in \{1,2\},\ i \in \{1,...,N\}.$$

Note that these terms range between 0 (perfect correlation) and 4 (perfect anti-correlation), as the cosine similarity ranges between -1 and 1. Finally, we average these scores and compute the square root of the result, getting the first term of $\mathcal{L}_{Opt}$:

$$\sqrt{\frac{1}{2}\Big(\frac{1}{N}\sum_{i=1}^{N}(1 - c_{ii}^{(1)})^2 + \frac{1}{N}\sum_{i=1}^{N}(1 - c_{ii}^{(2)})^2\Big)}.$$

We compute the average to make this first term invariant to the batch dimension and the square root is used to range this first term between 0 (perfect correlation for all pair of images) and 2 (anti-correlation for all pair of images).

Otherwise, each element $c_{ij}^{(1)}, i \neq j$, of the cross-correlation matrix $C^{(1)}$ (Equation 4.2) is the cosine similarity between the row i of Z^A and the row j of P^B. In other words, it is the cosine similarity between the predictor representation of the distorted view, x_i^A and the encoder representation of the distorted view, x_j^B. Therefore, as x_i^A and x_i^B are negative samples, because they come from different images, we want both representations to be as less correlated as possible, or what is the same, that $c_{ij}^{(1)}$ is as much closer as possible to 0. Analogously, we also want each element $c_{ij}^{(2)}, i \neq j$ of $C^{(2)}$ to be as much closer as possible to 0. We force the off-diagonal elements of $C^{(1)}$ and $C^{(2)}$ to go to 0 with the following terms in the loss function:

$$(c_{ij}^{(k)})^2,\ k \in \{1,2\},\ i,j \in \{1,...,N\},\ i \neq j.$$

Note that these terms range between 0 (no correlation) and 1 (perfect correlation or anti-correlation), as the cosine similarity ranges between -1 and 1. Finally, we average these

scores and compute the square root of the result, getting the second term of $\mathcal{L}_{Opt}$:

$$\sqrt{\frac{1}{2}\Big(\frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}(c_{ij}^{(1)})^2+\frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}(c_{ij}^{(2)})^2\Big)}.$$

Just like with the first term, we compute the average to make this second term invariant to the batch dimension and the square root is used to have consistency with the first term. This second term ranges between 0 (no correlation for any negative pair of distorted views) and 1 (perfect anti-correlation or correlation for all pairs of negative distorted views). From the formulation of $\mathcal{L}_{Opt}$ (Equation 4.2), we see that it ranges from 0 and $2+1\cdot\lambda$, being 0 the desired output.

4.3 Data Augmentation Techniques

OptSSL uses the same data augmentation techniques as other state-of-the-art SSL methods, like BYOL [29] and Barlow Twins [64]. Concretely, the image augmentations applied (in order) are:

- Random resize cropping: a random patch of the original image is selected, with side lengths between 8% and 100% of that of the original image and an aspect ratio between $3/4$ and $4/3$. Finally, the path is resized to a resolution of 224×224 pixels using bicubic interpolation.

- Random horizontal flipping: the patch from the previous step is randomly horizontal flipped with some probability p.

- Color jittering: the brightness (br), contrast (con), saturation (sat) and hue of the patch are randomly changed with some probability p.

- Grayscale conversion: the patch is converted to grayscale with some probability p.

- Gaussian blurring: the patch is randomly blurred with some probability p using a Gaussian kernel with a standard deviation in $[0.1, 2.0]$.

- Solarization: all the pixel values above a threshold of 0.5 (supposing that pixels range between 0 and 1) are complemented with some probability p.

The pretext task generator $\mathcal{P}$ of OptSSL uses a distortion process $\mathcal{T}_{Opt} = (\mathcal{T}_1, \mathcal{T}_2)$, where $\mathcal{T}_1$ and $\mathcal{T}_2$ apply the listed augmentations sequentially with a predetermined probability for each type. Then, for each image x in a batch X, the two augmented views are computed as $x_i^A \sim \mathcal{T}_1(x_i)$ and $x_i^B \sim \mathcal{T}_2(x_i)$. The probabilities used are the followings:

Augmentation technique	Probability in $\mathcal{T}_1$	Probability in $\mathcal{T}_2$
Random resized cropping	1.0	1.0
Random horizontal flipping	0.5	0.5
Color jittering (br, con, sat, hue)	0.8, (0.4, 0.4, 0.2, 0.1)	0.8, (0.4, 0.4, 0.2, 0.1)
Grayscale conversion	0.2	0.2
Gaussian blurring	1.0	0.1
Solarization	0.0	0.2

Table 4.1: Probabilities used for the generating views process.

In Fig. 4.2 an example of the behaviour of each of these augmentation techniques is shown.

Figure 4.2: Application example of the different augmentation techniques applied in the data augmentation process of OptSSL in an image of the Food-101 dataset [8].

Chapter 5

Validation

In this chapter, we present the experiments carried out for the project, and an analysis of their results. In Section 5.1, we introduce the environment and in Section 5.2, we show the datasets used for the experiments. Next, in Section 5.3, we discuss the evaluation metrics and finally, in Section 5.4, we present in detail the implemented experiments and an analysis of the results.

5.1 Environment

All the code for the experiments was implemented using Python. More technically, we used the PyTorch framework combined with PyTorch Lightning. To run the experiments, we used Colaboratory (`https://colab.research.google.com`) and two RTX2080Ti GPUs from the CVMLUB Group (`www.ub.edu/cvub`) of UB.

Pytorch is an open source ML framework based on the Torch library, widely spread and frequently used for Computer Vision and Natural Language Processing applications. We decided to use PyTorch over other libraries, such as Keras, because the explored SSL models are officially implemented using this framework. Thus, it was easier to keep the framework to explore and introduce modifications on the code. Nevertheless, in the first steps of the project, when introducing to the DL field, we also used Keras for some practice experiments.

PyTorch Lightning is a PyTorch research library that provides a high-level interface for PyTorch. It tries to simplify the coding process for DL experiments and helps to make the code easier to be understood. We used this library for the models defined from scratch and for transfer learning tasks.

Colaboratory is a Jupyter Notebook service provided by Google Research that allows to write and execute Python notebooks in a private virtual machine. With this product, we can run our code on GPUs freely. However, the virtual machine sessions finish when there is inactivity or when the maximum lifetime is exceeded. For this reason, we upgrated our

plan to Google Colab Pro, so as to have access to faster GPUs, increase the lifetime of the sessions and have more RAM memory.

5.2 Dataset

In this project, we worked on the Food-101 dataset [8]. This dataset contains 101000 food images divided in 101 classes, with 1000 images per food class. The images have different resolutions, with a maximum side length of 512 pixels. The dataset is split into a training and a test set. The training set contains 75750 images (750 images per class) and the test one contains the remaining 25250 images (250 images per class). The labels of all test images have been manually reviewed, but the training dataset still contains some amount of noise meaning that some images are wrongly labeled. Following the procedure of other works [45][50], we use the test split both for validation and testing.

Figure 5.1: Images from the FOOD-101 dataset [8].

5.3 Evaluation Metrics

In our experiments, we can distinguish two types of training. The first one is based on SSL and a model is trained to learn effective image representations in a latent space. The second one is based on SL and a classifier is trained for the image recognition task. In fact, both training are executed sequentially for each experiment. We first carry out a **pre-training** phase, where a model is trained with the SSL paradigm. Then, we leverage a part of this model to build a classifier and perform a **classifier training** phase under the SL protocol, for the downstream task of image recognition. Owing to the difference in the architecture and the purpose of the models in both types of training, we use different metrics and techniques at inference time to evaluate them.

5.3.1 Evaluation Metrics for the Classifiers

The models trained in the classifier training phase, called classifiers, aim to classify images into a discrete set of categories. In particular, as we work on the Food-101 dataset, into 101 food categories. To evaluate their performance, following the methodology used

in the SSL papers [12][64] for the image recognition downstream task, we use the validation dataset to compute the overall accuracy (Acc). Roughly speaking, using a validation dataset, the overall accuracy is the ratio between the well-classified images and the total number of images in the dataset, being 1 the highest value (when all the images are correctly classified), and 0 the worst value (when all image are wrongly classified). However, sometimes it is given in percentage. Hence, it is defined as follows:

Given the validation dataset $D_{val} = \{X, Y\} = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i are the validation images and $y_i \in \{1, ..., C\}$ are their corresponding classes $\forall i \in \{1, ..., N\}$, the **overall accuracy** on the validation dataset D_{val} is:

$$Acc = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\hat{y}_i = y_i} \text{ or } Acc = \left(\frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\hat{y}_i = y_i} \right) \cdot 100\ \%,$$

where $\hat{y}_i$ is the predicted class for the image x_i and $\mathbb{1}$ is the indicator function.

As Food-101 dataset is balanced, meaning that the number of images per class is equitable, the overall accuracy is a suitable metric to evaluate the models' performance.

Apart from the overall accuracy, we also use uncertainty metrics to evaluate the confidence of the models' predictions. To that purpose, we use an MC Dropout approach (Section 3.3.3) and we compute the uncertainty approximations in terms of the predictions variance (*var*), entropy ($\tilde{H}$) and mutual information $\tilde{M}I$ metrics defined in Section 3.3.4.

5.3.2 Evaluation Metrics for the SSL Models

As mentioned, the objective in the pre-training phase is to learn image representations in a latent space so as to later transfer this learning for a downstream task. As the downstream task is image classification, we expect that the learnt representations are useful for such objective, so we evaluate how good they are using a classification method based on the representations. As proposed in the SimSiam's paper [12], we use a k-Nearest Neighbors (k-NN) classifier based on the implementation proposed by Wu et al. [59]. To make the prediction of a validation image $\hat{x}$, the k-NN classifier carries out the following steps:

Given the training dataset $D_{tr} = \{X, Y\} = \{(x_i, y_i)\}_{i=1}^{N}$ where x_i are the training images and $y_i \in \{1, ..., C\}$ are their corresponding classes $\forall i \in \{1, ..., N\}$,

1. Compute the feature representation $z_i = f(x_i; \theta)$ for each image x_i in the training dataset and store it in a memory bank, where f is the SSL model mapping and θ its parameters.
2. Compute the feature representation $\hat{z} = f(\hat{x}; \theta)$ of image $\hat{x}$.
3. Compute the cosine similarities $c_i = C_s(z_i, \hat{z})\ \forall i \in \{1, ..., N\}$.
4. Select the the top k nearest neighbors to $\hat{x}$ according to the cosine similarities computed in step 3 and store them in a set $\mathcal{N}_k$, where k is a defined constant.

5. Compute the vector $w = (w_1, ..., w_C)$, where $w_i = \sum_{c_j \in \mathcal{N}_k} \alpha_j \cdot \mathbb{1}_{y_j=i}$ $\forall i \in \{1, ..., C\}$, $\alpha_j = \exp(c_j/\tau)$ and τ is a constant.

6. The predicted class for image $\hat{x}$ is $\hat{y} = argmin\ (w)$.

To perform the evaluation, we predict the class for each image in the validation dataset, using the output of the backbone that will be re-used for the downstream task, and we compute the overall accuracy (Acc) as defined in Section 5.3.1. In the implementation, we use $k = 200$ and $\tau = 10$. It is important to remark that the training and validation image labels are only used in this evaluation, but never used for training, since we are in the SSL paradigm.

5.4 Implementation Setting

In this section, we describe the main architecture and the training setting of the baseline SSL models and the classifiers explored in the experiments.

5.4.1 Baseline models

For the classifier, without loss of generality, we used the ResNet-50 [32] as base CNN architecture, since it has been accepted as an optimal network to solve many different image recognition problems. In order to apply an MC Dropout approach to quantify uncertainty, we readjusted its architecture replacing the output layer by a dropout layer (with a dropout rate of 0.4) and adding an output layer of N neurons with softmax activation, where N is the number of available classes to be predicted. In our experiments, as we used the Food-101 dataset, we set $N = 101$. In the description and the analysis of the experiments, we call this baseline architecture B-ResNet-50.

Regarding SSL models used in the pre-training phase, all of them include a ResNet-50 with the output layer cut as a backbone and other layers on top of it. After the pre-training phase, the backbone is re-used, transferring its learning, for the classifier.

In Fig. 5.2, it is shown the architecture of the baseline models architecture and how the transfer learning process between them is done.

5.4.2 Setting of the OptSSL Model

OptSSL has the architecture explained in Chapter 4, with the default output units for each layer. Before training, the ResNet-50 used as a backbone is pre-trained on ImageNet [15]. The projection MLP head and the predictor follow the default PyTorch initialization, with which the weights of each fc layer are initialized by a uniform distribution $\mathcal{U}(\sqrt{-k}, \sqrt{k})$ ($k = 1/\text{input units}$). The input shape of the model is $224 \times 224 \times 3$. To perform the training we use a batch size of 32. We use the SGD optimizer with a weight decay of 0.0001, a momentum of 0.9 and a learning rate of $lr \times \text{batch size}/256$, starting with $lr = 0.05$. The learning rate is reduced until 0 with a cosine decay schedule.

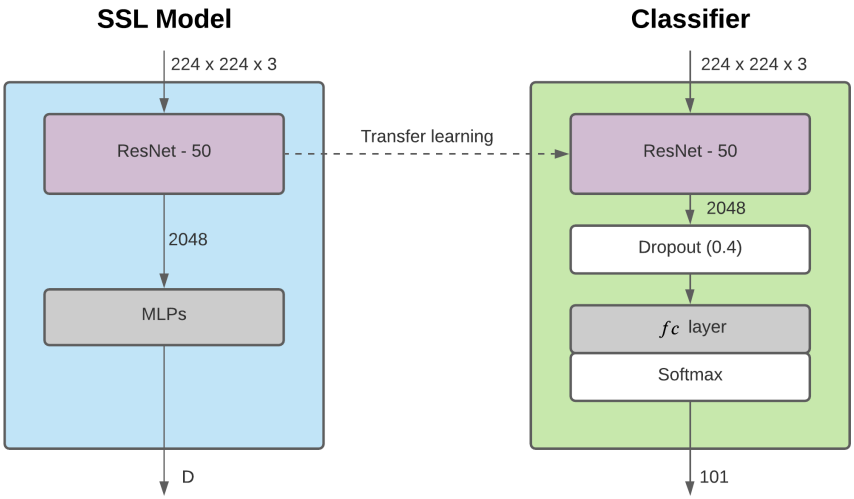


Figure 5.2: **Left**: SSL models architecture. **Right**: classifier architecture (B-ResNet-50). After the pre-training phase, the ResNet-50 from the SSL model is re-used to train the classifier.

5.4.3 Setting of the SimSiam Model

For the SimSiam configuration, we use the default architecture described in Section 3.2.2, with the described output units for each layer. The ResNet-50 used as a backbone is initialized with weights pre-trained on ImageNet and the projection MLP head, as well as the predictor, follow the default PyTorch initialization. The settings to perform the training are the default ones described in SimSiam's paper, except the batch size. In particular, we use the SGD optimizer with a weight decay of 0.0001, a momentum of 0.9 and a learning rate of $lr \times$ batch size$/256$, starting with $lr = 0.05$. The lr is decreased with a cosine decay schedule until 0. The batch size used is 32 and the input shape is $224 \times 224 \times 3$.

5.4.4 Setting of the Barlow Twins Model

We use the default architecture of Barlow Twins described in Section 3.2.2, with the detailed output units for each layer. We initialize the ResNet-50 encoder with pre-trained weights on ImageNet and we start the projector with the default PyTorch initialization. The input shape of the model is $224 \times 224 \times 3$. For the training, we use a batch size of 32 and the rest of the configurations are the same as in the original paper. We use the LARS optimizer [63] with a weight decay of $1 \cdot 10^{-6}$, a momentum of 0.9 and a learning rate of $lr \times$ batch size$/256$, starting with $lr = 0.2$ for the weights, and $lr = 0.0048$ for the biases and batch normalization parameters. We perform a learning rate warm-up for a period of 10 epochs and then we reduce the learning rate with a cosine decay schedule by a factor of 1000. The biases and batch normalization parameters are excluded from LARS

adaption and weight decay. The λ value used in the loss function is 0.0051, the same as in the original implementation.

5.4.5 Setting of the Final Classifier

All the classifiers used in the experiments have the B-ResNet-50 architecture (see 5.4.1). The initialization of the ResNet-50 changes between the experiments. The baseline classifier takes the ResNet-50. However, when using a SSL pre-training phase, the initialization of the ResNet-50 comes from this pre-training phase. The output layer is always initialized with the default PyTorch initialization. The training settings to perform the final classifier training phase the following:

- Input shape: $224 \times 224 \times 3$.
- Data augmentation techniques: a first random transformation is applied using the class *transforms.RandAugment* from the package *torchvision*, with the default parameters. Next, a random resized cropping with scale in $[0.08, 1]$ and ratio in $[3/4, 4/3]$ is applied. Finally, a random horizontal flipping is applied with a probability of 0.5.
- Batch size: 64.
- Optimizer: SGD with an initial learning rate of 0.001, weight decay of 0.0005 and momentum of 0.9.
- Schedule: cosine annealed warm restart [48] learning schedule with an initial number of epochs per cycle of 10 with an increasing factor of 2 after each cycle.
- Loss function: Cross-Entropy Loss.

5.5 Results and Analysis

As said in the Introduction Chapter (1.3), our goal is to explore the relevance of SSL methods to improve DL algorithms validating them on the domain of food recognition. To provide an organized report, we firstly give a discussion about different SSL approximations and variations in Section 5.5.1, and then we divide the results obtained in two sections. In Section 5.5.2, we present the results with respect to the SSL pre-training phase. We compare OptSSL with other state-of-the-art models and we show that it is optimal one for learning data representation in the food domain. In Section 5.5.3, we show the potential of using SSL for the food recognition downstream task, we analyze different techniques to face the overfitting problem, and we provide a comparison of the performance of a classifier using different SSL pre-trained models.

5.5.1 Discussion about different SSL Approximations and Variations

The different SSL methods explored in our study, which are SimSiam, Barlow Twins, and OptSSL, share the same objective: learn effective images representation in a latent space. However, there exist some differences between their learning processes to achieve

that aim. This divergence basically comes from two aspects: the data augmentation techniques applied to generate distorted views of images, and the loss function used to contrast the images' representation. To understand and prove that OptSSL is the optimal SSL method for learning data representation in the food domain, we design intermediate methods between the explored ones to analyze the benefits of each part of the OptSSL architecture with respect to the others'.

First of all, let us focus on the differences between OptSSL and SimSiam. Regarding the loss function, both leverage on cosine similarities between image representations in their formulations. However, SimSiam's loss function (Equation 3.4) only contrasts representations of positive pairs of images, while OptSSL's loss function (Equation 4.2) contrasts representations between positive and negative pairs of images.

To later study which of these two techniques is better in our domain, we define an extended SimSiam's loss function that also considers the comparison between negative pairs of images. Using the same notation as in Section 4.2, we can give an equivalent expression to the SimSiam's loss function formulation in Equation 3.4, which is:

$$\mathcal{L}_{SimSiam} = -\frac{1}{2}\Big(\frac{1}{N}\sum_{i=1}^{N} c_{ii}^{(1)} + \frac{1}{N}\sum_{i=1}^{N} c_{ii}^{(2)}\Big).$$

To include the contrast of negative pairs of images, we also consider the off-diagonal elements of the cross-correlation matrices $C^{(1)}$ and $C^{(2)}$ in $\mathcal{L}_{SimSiam}$, which gives an extended loss function that we call $\mathcal{L}_{SimSiam}^{(v1)}$:

$$\mathcal{L}_{SimSiam}^{(v1)} = \mathcal{L}_{SimSiam} + \lambda\frac{1}{2}\Big(\frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}|c_{ij}^{(1)}| + \frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}|c_{ij}^{(2)}|\Big) = \tag{5.1}$$

$$= -\frac{1}{2}\Big(\frac{1}{N}\sum_{i=1}^{N} c_{ii}^{(1)} + \frac{1}{N}\sum_{i=1}^{N} c_{ii}^{(2)}\Big) +$$

$$\lambda\frac{1}{2}\Big(\frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}|c_{ij}^{(1)}| + \frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}|c_{ij}^{(2)}|\Big),$$

where $\lambda \in \mathbb{R}_{>0}$ is a constant to balance the importance between the terms.

We can see in Equation 5.1, the purpose of the second term of $\mathcal{L}_{SimSiam}^{(v1)}$ is to force the cosine similarity between the representations of negative samples to go to zero, so the representations of negative pairs of images should not be correlated. For the ablation studies on OptSSL, we define an intermediate SSL model called ***SimSiam (v1)***, which is exactly equal to SimSiam but uses the loss function $\mathcal{L}_{SimSiam}^{(v1)}$ instead of $\mathcal{L}_{SimSiam}$.

Regarding the data augmentation processes, there are differences between the sequence $\mathcal{T}_{OptSSL}$ defined in OptSSL 4.3 and the sequence $\mathcal{T}_{SimSiam}$ defined in SimSiam 3.2.2. To later study the effect of data augmentation process on SSL methods, we define two

transitional models, which are: ***SimSiam (v2)*** and ***OptSSL (v2)***. *SimSiam (v2)* is exactly equal to SimSiam with the only difference that the data augmentation process used is $\mathcal{T}_{OptSSL}$ instead of $\mathcal{T}_{SimSiam}$. Similarly, *OptSSL (v2)* is a modified version of OptSSL which the only difference that the data augmentation process used is $\mathcal{T}_{SimSiam}$.

Comparing the methods OptSSL and SimSiam against Barlow Twins, we see a considerable difference with respect to the loss functions. While SimSiam and OptSSL directly contrast image representations between different images, Barlow Twins is based on contrasting features, trying to minimize their redundancy. To do study of the potential of each approaches in the next section, we construct a loss function which gathers them. To construct this loss function, called $\mathcal{L}^{(v3)}_{OptSSL}$, we start from $\mathcal{L}_{OptSSL}$ (Equation 4.2) and we add the contrasting of representations along features. Following the notation in Section 4.2, we compute the cross-correlation matrix between Z^A and P^B (along the batch dimension), which we call $C^{(3)}$, and the cross-correlation matrix between Z^B and P^A (along the batch dimension), which we call $C^{(4)}$. Leveraging on this two matrices, the loss function $\mathcal{L}^{(v3)}_{OptSSL}$ is computed as:

$$\begin{aligned}\mathcal{L}^{(v3)}_{OptSSL} = &\sqrt{\frac{1}{2}\Big(\frac{1}{N}\sum_{i=1}^{N}(1-c^{(1)}_{ii})^2 + \frac{1}{N}\sum_{i=1}^{N}(1-c^{(2)}_{ii})^2\Big)} + \\ &+ \lambda\sqrt{\frac{1}{2}\Big(\frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}(c^{(1)}_{ij})^2 + \frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}(c^{(2)}_{ij})^2\Big)} + \\ &+ \sqrt{\frac{1}{2}\Big(\frac{1}{D}\sum_{i=1}^{D}(1-c^{(3)}_{ii})^2 + \frac{1}{D}\sum_{i=1}^{D}(1-c^{(4)}_{ii})^2\Big)} + \\ &+ \lambda\sqrt{\frac{1}{2}\Big(\frac{1}{D(D-1)}\sum_{i=1}^{D}\sum_{j=1,j\neq i}^{D}(c^{(3)}_{ij})^2 + \frac{1}{D(D-1)}\sum_{i=1}^{D}\sum_{j=1,j\neq i}^{D}(c^{(4)}_{ij})^2\Big)},\end{aligned} \tag{5.2}$$

where D is the dimension of the representations. The first two terms of the loss function are the same terms than in $\mathcal{L}_{Opt}$. The third and the fourth terms plays the same role as the *invariance term* and the *redundancy reduction term* in Barlow Twins' loss function 3.6, respectively.

From $\mathcal{L}^{(v3)}_{OptSSL}$ (Equation 5.2), we define a variation of OptSSL model, called ***OptSSL (v3)***, with the same configuration of OptSSL except the loss function, which is $\mathcal{L}^{(v3)}_{OptSSL}$.

Finally, and with the aim of studding the OptSSL's loss function $\mathcal{L}_{OptSSL}$, we consider a not-normalized version of $\mathcal{L}_{OptSSL}$, called $\mathcal{L}^{(v1)}_{OptSSL}$. The normalization process used in $\mathcal{L}_{OptSSL}$, consists in averaging the atomic values of each term in the loss function and then

computing its square root (4.2). Therefore, the not-averaged $\mathcal{L}^{(v1)}_{OptSSL}$ loss function is:

$$\mathcal{L}^{(v1)}_{Opt} = \Big(\sum_{i=1}^{N} (1 - c^{(1)}_{ii})^2 + \sum_{i=1}^{N} (1 - c^{(2)}_{ii})^2 \Big) + \lambda \Big(\sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} (c^{(1)}_{ij})^2 + \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} (c^{(2)}_{ij})^2 \Big). \quad (5.3)$$

From $\mathcal{L}^{(v1)}_{Opt}$ (5.3), we design an intermediate model from OptSSL, called ***OptSSL (v1)***, which relies on this loss function. This model is later used to study whether the normalization in the OptSSL's loss function is important.

5.5.2 Results of the Data Representation using SSL

In this section, we compare the performance of our OptSSL model with respect to other state-of-the-art SSL models and analyze the source of its benefits with several ablation experiments.

In Table 5.1, we present the results of the different SSL models compared according to the validation accuracy of a k-NN classifier on the Food-101 dataset. Note that the parameter λ of the OptSSL loss function, as defined in Equation (4.2), determines the importance (weight) between the diagonal (typical for SimSiam) and off-diagonal elements (available in the BarlowTwins but not in SimSiam method) of the cross-correlation matrices. Finding the optimal values is an issue of empirical tests. We tested with different values of λ, where values of 1 mean equal weight of diagonal and off-diagonal elements and values of 0 mean that no off-diagonal elements are considered in the loss function. Models followed by (vX) are the customized models defined in Section 5.5.1 used for ablations.

Experiment	Model	λ	Epochs	Top 1 acc. (%)	Top 5 acc. (%)
1	Barlow Twins	0.0051	100	48.51	77.98
2	SimSiam	-	100	50.42	79.18
3	SimSiam	-	150	54.33	81.62
4	SimSiam (v1)	$1 \cdot 10^{-6}$	150	54.91	81.85
5	SimSiam (v1)	0.5	150	57.47	83.71
6	SimSiam (v1)	1.0	150	57.88	83.92
7	SimSiam (v2)	-	150	61.76	86.48
8	OptSSL (v1)	0.0	150	37.61	67.59
9	OptSSL (v1)	1.0	150	53.02	80.52
10	OptSSL (v2)	0.0	150	58.48	84.53
11	OptSSL (v2)	0.5	150	59.00	84.91
12	OptSSL (v2)	0.75	150	58.82	84.81
13	OptSSL (v2)	1.0	150	58.94	84.78
14	OptSSL (v3)	0.5	150	62.06	86.43
15	OptSSL	0.5	150	**63.52**	**87.48**

Table 5.1: Evaluation of the different SSL models on the Food-101 dataset according to the accuracy of a k-NN classifier. (vX) denotes a customization of the original SSL model, which are defined in Section 5.5.1

To analyze the potential of our OptSSL model, next we give comparisons between intermediate experiments found in Table 5.1 to extract conclusions on different stages of the process. At the end, we give general conclusions to justify the performance of OptSSL. In the discussion of the results, we name the experiments using their first column value in Table 5.1. Furthermore, to provide more information, in Appendix A we show the loss training curves and the monitoring of the k-NN validation accuracy during training for all the experiments in Table 5.1.

Results of the comparison of image-based vs. feature-based contrastive SSL

First of all, we focus on comparing Barlow Twins vs. SimSiam methods. A part from the architecture and configuration setting variations between them, the most important difference is that SimSiam learns by directly contrasting the representations of different images, while the Barlow Twins method does the contrasting along the features, trying to minimize their redundancy. The comparison between SimSiam and BarlowTwins is done through the experiments 1 and 2 from Table 5.1, where both methods are trained for 100 epochs. Before looking at their final accuracy, we first analyze their loss training curves (Fig. 5.3) and the monitoring of the k-NN accuracy (Fig. 5.4) during the training, to check their behaviour.

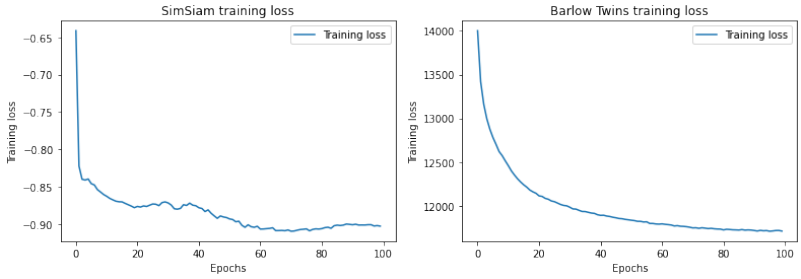


Figure 5.3: Training loss curves of SimSiam (left) and Barlow Twins (right) methods monitored for 100 training epochs on the Food-101 dataset.

Fig. 5.3 shows the training loss curves of SimSiam and Barlow Twins methods for 100 training epochs. We can note that both loss curves are not comparable because the loss functions of the methods do not have the same range and contrast different aspects. However, the graphics show that, in both approaches, the shape of the loss curves is converging as expected, so the methods are behaving in a robust way. Finally, to check if the methods progressively learn more efficient data representations, we check the k-NN accuracy graphics. Fig. 5.4 allows to monitor the validation accuracy (in %) of a k-NN classifier during training. The monitoring is performed every 20 epochs. As we can see there, the k-NN validation accuracy progressively improves, what means that indeed, both methods progressively learn better the data representation. After this checking, we observe see that, as reflected in Table 5.1, SimSiam reaches a top 1 accuracy 1.91% higher than Barlow Twins after the 100 epochs of training. This fact leads us to assume that, even though both

methods are capable to learn representations, for the food image representation task it is better to directly contrast image representations in the loss function, as SimSiam does.

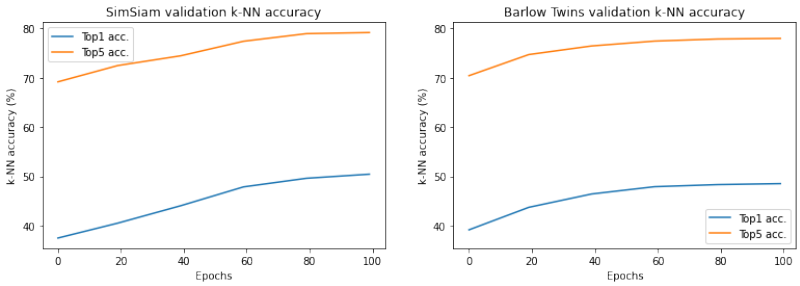


Figure 5.4: Monitoring of the validation accuracy (in %) of a k-NN classifier during the training of SimSiam (left) and Barlow Twins (right) methods. The monitoring is performed every 20 epochs.

Results of the comparison of positive-samples contrasting vs. positive and negative-samples contrasting

Acknowledging that an image-based contrastive approach works better in the food recognition domain, we analyze the effect of comparing only the representation of positive pairs of images or including also the comparison of negative pairs of images. As far as we know, in the SimSiam's loss function, only positive pairs of images are contrasted. In experiments 4, 5 and 6 from Table 5.1, we use a customized version of SimSiam (called *SimSiam (v1)*), where the loss function also contrast the representation of negative pairs of images, as described in Section 5.5.1.

To compare the effect of including negative pairs in the loss function, we can compare the experiment 3 with the experiments 4, 5 and 6 from Table 5.1. The experiment 4 is a sanity check to assure that the extended loss function does not break the original SimSiam. The λ value used in $\mathcal{L}^{(v1)}_{SimSiam}$ is $1 \cdot 10^{-6}$, so that $\mathcal{L}^{(v1)}_{SimSiam} \approx \mathcal{L}_{SimSiam}$. We can see in Table 5.1 that the top 1 accuracy of experiment 4 from Table 5.1 is 0.58% higher than the one of experiment 3, showing not only that the extended loss function does not break the original SimSiam, but that the addition of negative pairs contrast can play an important role, since for a very small λ the accuracy improves a bit.

The experiments 5 and 6 show the behaviour of *SimSiam (v1)* with λ values of 0.5 and 1.0, respectively. The experiment 5 improves the top 1 accuracy of the original SimSiam (experiment 3) by 3.14% and experiment 6 improves it by 3.55%. Therefore, we can conclude, as empirically tested, that the addition of negative samples comparison helps the model to learn more powerful data representations.

Results of the comparison of SimSiam's Loss Function vs. OptSSL's Loss Function

OptSSL's Loss Function has two terms (see Section 4.2). The first term plays the same role as the SimSiam's loss function (positive pairs contrasting) and the second one acts like the second term of the *SimSiam (v1)*'s loss function (negative pairs contrasting). The comparisons of experiment 10 to experiment 3, and experiments 11 and 13 to experiments 5 and 6 (all of them from Table 5.1) allow us to study the effectiveness of the OptSSL's loss function with respect to the SimSiam's and *SimSiam (v1)*'s ones. *OptSSL (v2)* model, defined in Section 5.5.1, is a modified version of OptSSL, where the only change is that the data augmentation techniques used are the same as in SimSiam. Therefore, the only difference between SimSiam, *SimSiam (v1)* and *OptSSL (v2)* is in their loss functions. For the experiment 10, $\lambda = 0.0$ so it allows to compare the first term of OptSSL's loss function with SimSiam's loss function. As reported in Table 5.1, the experiment 10 beats the experiment 3 top 1 accuracy by 4.15%, which means that the treatment of the cosine similarities between positive samples done in OptSSL's loss function is better, and helps more to learn representations, than the one in SimSiam.

Contrasting the experiments 11 and 13 with the experiments 5 and 6 is equivalent to compare the OptSSL's loss function with the *SimSiam (v1)*'s one, keeping the same weight (λ) for negative samples. Both experiments (11 and 13) show an improvement higher than 1% with respect to the experiments 5 and 6, respectively. Therefore, it indicates that the whole OptSSL's loss function is also more useful than the extended SimSiam's one. For completeness, the experiment 12 runs *OptSSL (v2)* with $\lambda = 0.75$. However, from the experiments 10, 11, 12 and 13 we deduce that the best λ value for OptSSL's loss function, in the set of tested values, in our domain is 0.5, since it is the one that allows to achieve a better top 1 accuracy.

Results of the Normalization in OptSSL's Loss Function

The atomic values of each term in OptSSL's loss function are averaged along the number of values and followed by a square root (see 4.2). We empirically show the importance of this normalization comparing the experiments 8 and 9 from Table 5.1. *OptSSL (v1)* is a variant of SSLOpt where there is no normalization in the loss function and the data augmentation techniques are the same as in *OptSSL (v2)*. The experiment 8 ($\lambda = 0.0$) shows a top 1 accuracy drop of 20.87% with respect to the experiment 10 and the experiment 9 ($\lambda = 1.0$) shows a top 1 accuracy drop of 5.92% with respect to the experiment 13. Therefore, this results confirm that normalization takes a real important role in the OptSSL's loss function and that it is crucial for the OptSSL success.

Results of the Effect of the Data Augmentation Techniques in OptSSL

OptSSL uses the data augmentation techniques defined in Section 4.3 to obtain distorted views from the original images in the training dataset. We did a study to check if the choice of the image augmentation process is important, comparing the performance of OptSSL using the data augmentation techniques in SimSiam. As described in Section 5.5.1, *OptSSL (v2)* is a variant of OptSSL with SimSiam's data augmentation techniques.

Similarly, *SimSiam (v2)* is a variation of SimSiam using OptSSL's data augmentation techniques. Comparing experiments 15 and 11 from Table 5.1, we see that OptSSL top 1 accuracy drops by 4.52% when using SimSiam's DA techniques. The same phenomenon is also observed in SimSiam, since when using OptSSL's DA techniques (experiment 7) the top 1 accuracy increases by 7.43% than using original SimSiam's DA techinques (experiment 3). Therefore, we conclude that the data augmentation techniques used in OptSSL are optimal in our domain.

Results of adding feature contrasting to OptSSL

This last study explores whether adding feature contrasting to OptSSL can help improve data representation. *OptSSL (v3)* is an extension of OptSSL, where feature contrasting is also added to the loss function (see Section 5.5.1). The experiment 14 shows that when extending $\mathcal{L}_{Opt}$ to $\mathcal{L}_{Opt}^{(v3)}$ in OptSSL, the top 1 accuracy drops by 1.46%, showing again, as proved before, that feature contrasting is not as useful as image representation contrasting for the food data representation task.

With all the results reported and discusses above, we conclude that OptSSL is optimal SSL model for data representation in the food image domain, and that every component of its loss function is important for its success.

5.5.3 Results of the Food Recognition using SSL and Classification

In this section, we show the potential of leveraging a pre-trained SSL model for the food recognition task. To structure the results, we divide them into three parts. We firstly show that a well SSL pre-trained model can help improve a baseline classifier. Then, we make a discussion on the classifier training process to find the best optimization. Finally, we compare the usefulness of the best SSL models found in the previous section for the food recognition downstream task.

Results of the comparison of the Baseline Classifier vs. using SSL pre-trained Models

In this section, we compare the performance of the baseline classifier B-ResNet-50 when using a SSL pre-training phase for food classification problem. The transfer learning between the pre-trained SSL models and the classifier are done as explained in Section 5.4.1. The results of Table 5.2 are obtained training the classifiers with the following setting (called *Setting 1*):

- Input shape: $224 \times 224 \times 3$.
- Data augmentation techniques: random resized cropping with scale in $[0.08, 1]$ and ratio in $[3/4, 4/3]$, and random horizontal flipping with a probability of 0.5.
- Batch size: 64.
- Optimizer: SGD with an initial learning rate of 0.001, weight decay of 0.0005, and momentum of 0.9.

- Schedule: cosine decay for 100 epochs.
- Early stopping: early stopping with a patience of 10 epochs for the validation loss.
- Loss function: Cross-Entropy Loss.

The Table 5.2 shows a comparison of the performance of different trained classifiers, using the metrics described in Section 5.3.1.

Architecture	Pre-training phase	Acc. (%)	Variance	Entropy	Mutual Inf.
B-ResNet-50	None (baseline)	85.23	$3.17 \cdot 10^{-3}$	0.45	$2.29 \cdot 10^{-2}$
B-ResNet-50	SimSiam (100 epochs)	87.35	$1.48 \cdot 10^{-3}$	0.38	$1.10 \cdot 10^{-2}$
B-ResNet-50	Barlow Twins (100 epochs)	84.25	$5.41 \cdot 10^{-3}$	0.42	$3.65 \cdot 10^{-2}$

Table 5.2: Evaluation of different classifiers. In the first row, no pre-training phase is used. In the second row, a pre-training phase using the SimSiam method is carried out. In the third row, a pre-training phase using the Barlow Twins method is performed.

The results of Table 5.2 show that using a SSL pre-training phase with SimSiam helps improve the accuracy of the baseline classifier by 2.12%. Furthermore, it also helps reduce the uncertainty of the model, according to the experienced decrease in the uncertainty metrics. In summary, SimSiam helps to get a most robust classifer. However, the table also shows that a SSL pre-training phase with Barlow Twins does not help improve the performance. In fact, the accuracy decreases by 0.98%. From these results, we confirm again that image-based SSL is more effective in the food image domain. To analyze the differences during the classifier training phase of the three experiments, we show the comparison of the loss curves in Fig. 5.5 and the comparison of the evolution of the accuracy in Fig. 5.6.

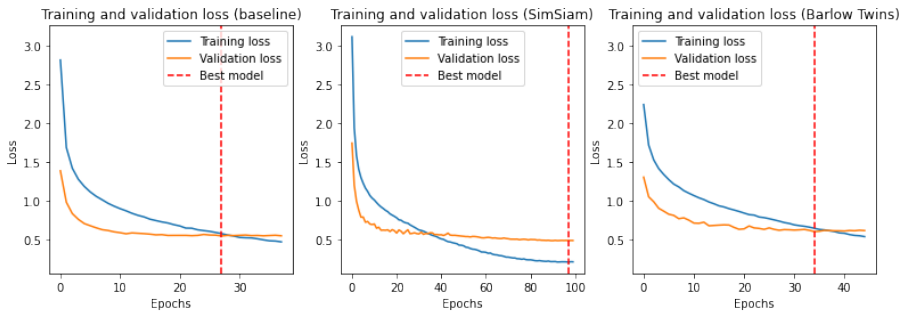


Figure 5.5: Loss curves of classifiers training. **Left**: baseline. **Center**: using a pre-training phase with SimSiam. **Right**: using a pre-training phase with Barlow Twins.

Fig. 5.5 shows the loss curves of the classifier training without and with SSL pre-training. There, we see that all the loss curves have a similar shape. However, one can

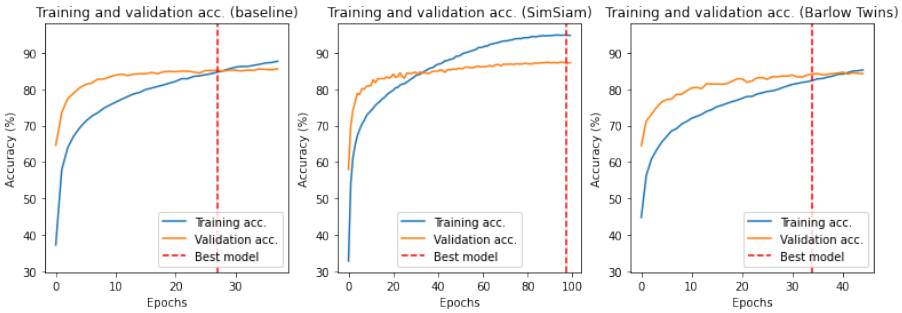


Figure 5.6: Accuracy curves of classifiers training. **Left**: baseline. **Center**: using a pre-training phase with SimSiam. **Right**: using a pre-training phase with Barlow Twins.

observe that the epoch where the validation loss stops improving (the best model achieved for the specific configuration) is very different in the three experiments. While the baseline and the classifier with a pre-training phase with Barlow Twins stop improving approximately at epoch 30, the classifier with a pre-training phase with SimSiam improves until epoch 98. This fact leads us to think that the SimSiam pre-training phase helps the classifier to start closer to a better local minimum of the loss function. Therefore, the classifier can learn for more epochs resulting in a better performance. Nevertheless, we see that in this case the difference between the training and the validation loss of the best model is much higher than in the other two experiments, which means that the model ends up overfitting. We can also see the overfitting problem of this method in Fig. 5.6, which shows the accuracy curves of the classifiers training. We see that the difference between the training and the validation accuracy is much higher. Here, we noticed that the SimSiam has a bigger problem with overfitting compared to the Barlow Twins and the baseline.

However, in spite of the overfitting problem, we see that the use of a pre-training phase with SimSiam (image-based contrasting SSL) improves the accuracy of the baseline classifier by 2.12%, showing the potential of SSL.

Results of the searching for the Best Classifier Training Setting

To deal with the overfitting problem faced when training the classifier with a pre-training phase with image-based contrastive SSL, we propose several training settings and compare their effectiveness. We tested 5 training setting on the same classifier, which is a classifier with a pre-training phase with SimSiam for 150 epochs.

1. The training setting *Setting 1* is the one used and described in the above section.

2. *Setting 2* is equal to *Setting 1*, but changing the data augmentation techniques, which are:

- Data augmentation techniques: a first transformation is applied using the class *transforms.RandAugment* from the package *torchvision*, with the default parameters. Next, a random resized cropping with scale in $[0.08, 1]$ and ratio in $[3/4, 4/3]$ is applied. Finally, a random horizontal flipping is applied with a probability of 0.5.

3. *Setting 3* is equal to *Setting 2*, but changing the initial learning rate and the schedule, which are:
   - Initial learning rate: 0.01.
   - Schedule: cosine annealed warm restart with a number of epochs per cycle of 10.
4. *Setting 4* is equal to *Setting 3*, but changing the initial learning rate and the number of epochs per cycle of the schedule, which are:
   - Initial learning rate: 0.001.
   - Number of epochs per cycle for the schedule: 40.
5. Finally, *Setting 5* is the final classifier training setting exposed in Section 5.4.5. To put it in context, it is equal to *Setting 4*, but using an initial number of epochs per cycle of 10 for the schedule and increasing it by a factor of 2 after each restart.

The results of the experiments to compare these settings are presented in Table 5.3. As we can see, the best training setting is *Setting 5*, with which the model reaches an accuracy of 88.60%. Regarding overfitting, we also see that it is the second less overfitted model after the one from *Setting 3*. The difference between the validation and the training loss with *Setting 5* is -0.01.

Arch.	Pre-training phase	Training setting	Train. loss	Val. loss	Acc. (%)
B-ResNet-50	SimSiam (150 epochs)	*Setting 1*	**0.22**	0.47	87.99
B-ResNet-50	SimSiam (150 epochs)	*Setting 2*	0.33	0.45	88.44
B-ResNet-50	SimSiam (150 epochs)	*Setting 3*	0.63	0.46	87.52
B-ResNet-50	SimSiam (150 epochs)	*Setting 4*	0.41	**0.43**	88.30
B-ResNet-50	SimSiam (150 epochs)	*Setting 5*	0.44	**0.43**	**88.60**

Table 5.3: Evaluation of different training setting for our classifier.

Fig. 5.7 shows the comparison of the loss curves during training, for each setting. *Setting 2* is capable to reduce the overfitting from 0.25 to 0.12 with respect to *Setting 1*, meaning that data augmentation helps reduce overfitting in our domain, as it usually does. Furthermore, we can see that *Setting 3*, *Setting 4* and *Setting 5* result in the least overfitted models. Therefore, we can conclude that the cosine annealed warm restart schedule also helps face overfitting. From the different settings, we take the last, *Setting*

5, as the best one, since it is the one that gives the most accurate model in regard to the accuracy and the second least overfitted model.

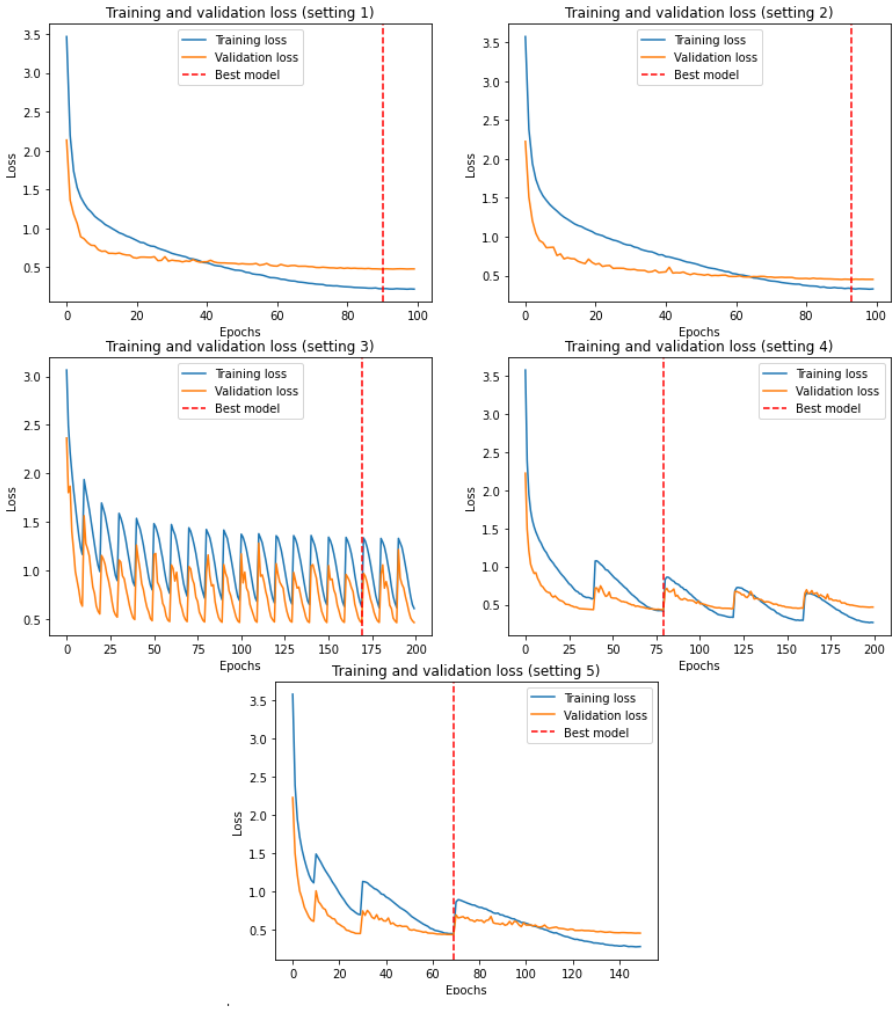


Figure 5.7: Loss curves of the classifiers training using the 5 different settings.

Results of the comparison of the SSL pre-trained Models for the Food Recognition Downstream Task

In this section, we compare the performance of our classifier when using the best SSL models presented in Table 5.1 in a pre-training phase. In particular, we take the models from the experiment 3 (SimSaim), the experiment 6 (*SimSiam (v1)*), the experiment 7 (*SimSiam (v2)*), the experiment 11 (*OptSSL (v2)*), the experiment 14 (*OptSSL (v3)*) and the experiment 15 (OptSSL). We use the setting of the final classifier training (5.4.5) to train the classifier in all the experiments. Table 5.4 shows the performance and the uncertainty

quantification of the different classifies using a pre-training phase, with each of the mentioned models, before training.

Architecture	Pre-training phase	Acc. (%)	Variance	Entropy	Mutual Inf.
B-ResNet-50	SimSiam	88.60	$1.51 \cdot 10^{-3}$	**0.39**	$1.08 \cdot 10^{-2}$
B-ResNet-50	SimSiam (v1)	**88.63**	$\mathbf{1.50 \cdot 10^{-3}}$	**0.39**	$\mathbf{1.07 \cdot 10^{-2}}$
B-ResNet-50	SimSiam (v2)	88.51	$1.52 \cdot 10^{-3}$	0.40	$1.09 \cdot 10^{-2}$
B-ResNet-50	OptSSL (v2)	88.55	$1.55 \cdot 10^{-3}$	**0.39**	$1.11 \cdot 10^{-2}$
B-ResNet-50	OptSSL (v3)	**88.63**	$\mathbf{1.50 \cdot 10^{-3}}$	**0.39**	$\mathbf{1.07 \cdot 10^{-2}}$
B-ResNet-50	OptSSL	88.53	$1.55 \cdot 10^{-3}$	**0.39**	$1.10 \cdot 10^{-2}$

Table 5.4: Evaluation of our classifier using different pre-training phases.

From the results in Table 5.4, we can see that the best classifier is not the one that uses a pre-training phase with the best SSL model (OptSSL). Nevertheless, the difference between the worst classifier and the best one (reported in Table 5.4) with respect to the accuracy is only of 0.12%. Regarding the uncertainty metrics, we also see that they are relatively stable across the experiments. To understand why the best SSL model is not the one that gives the better classifier, we analyze the loss curves during training (shown in Fig. 5.8) and the training and accuracy curves (shown in Fig. 5.9).

From the graphics in Fig. 5.8 and 5.9, we can see that the classifiers pre-trained with the best SSL models start at a better point in the first epoch. For instance, if we have a look at the graphics corresponding to the classifier that uses OptSSL in the pre-training phase, which is the best SSL model, we can observe that in the first epoch it is the one that has lower training and validation loss and higher training and validation accuracy. However, during training, all the graphics from different classifiers converge to a same point and this initialization difference is not appreciated at the end. Therefore, the substantial difference in the performance of the different SSL models is not appreciated when using them for the food recognition downstream task with our classifier. Therefore, in this domain it is not so important which SSL method is used in a pre-training phase, but as discussed before, the use of a pre-training phase with SSL really helps to improve a baseline classifier.

Finally, we give examples of well-classified and wrongly-classified images by the best classifier obtained (row 4 of Table 5.4). Fig. 5.10 shows the 4 images in Food-101 validation dataset that are predicted with lowest uncertainty, according to entropy. All of them are well classified. Fig. 5.11 shows the 4 images in Food-101 validation dataset that are predicted with highest entropy uncertainty, according to entropy. All of them are wrongly classified.

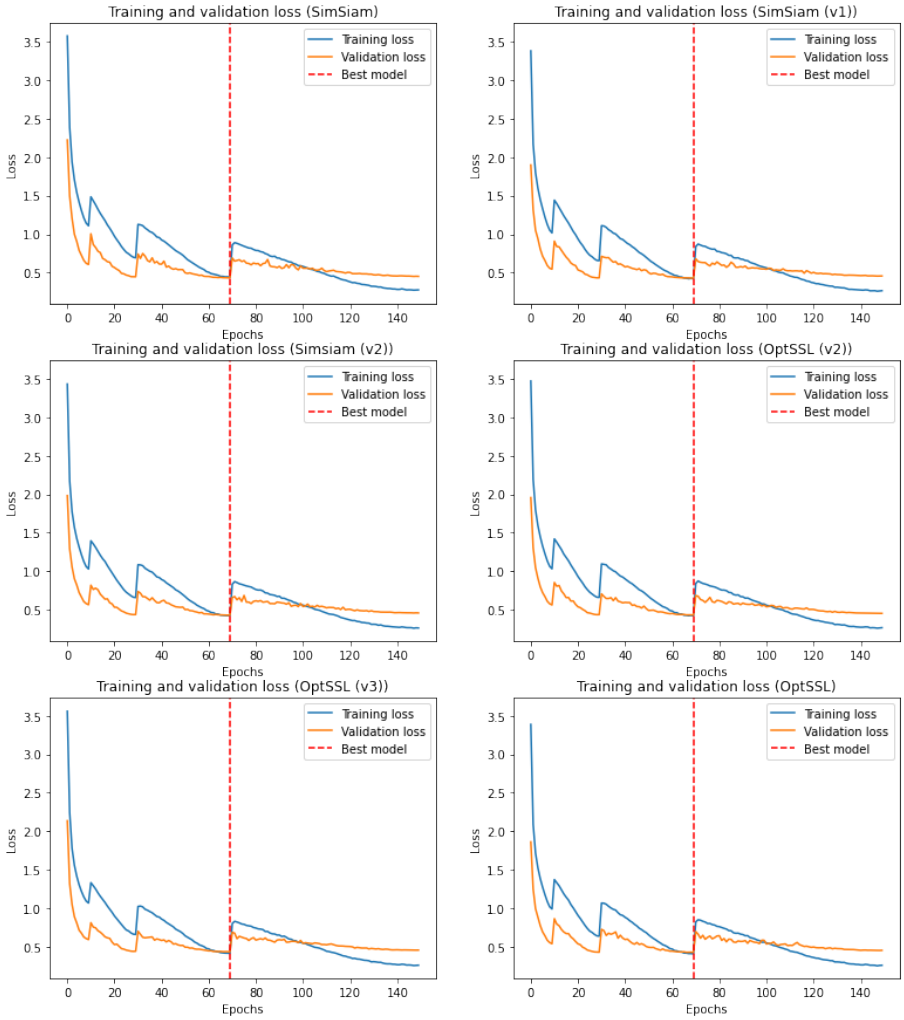


Figure 5.8: Loss curves of classifiers training using different SSL methods as the pre-training phase.

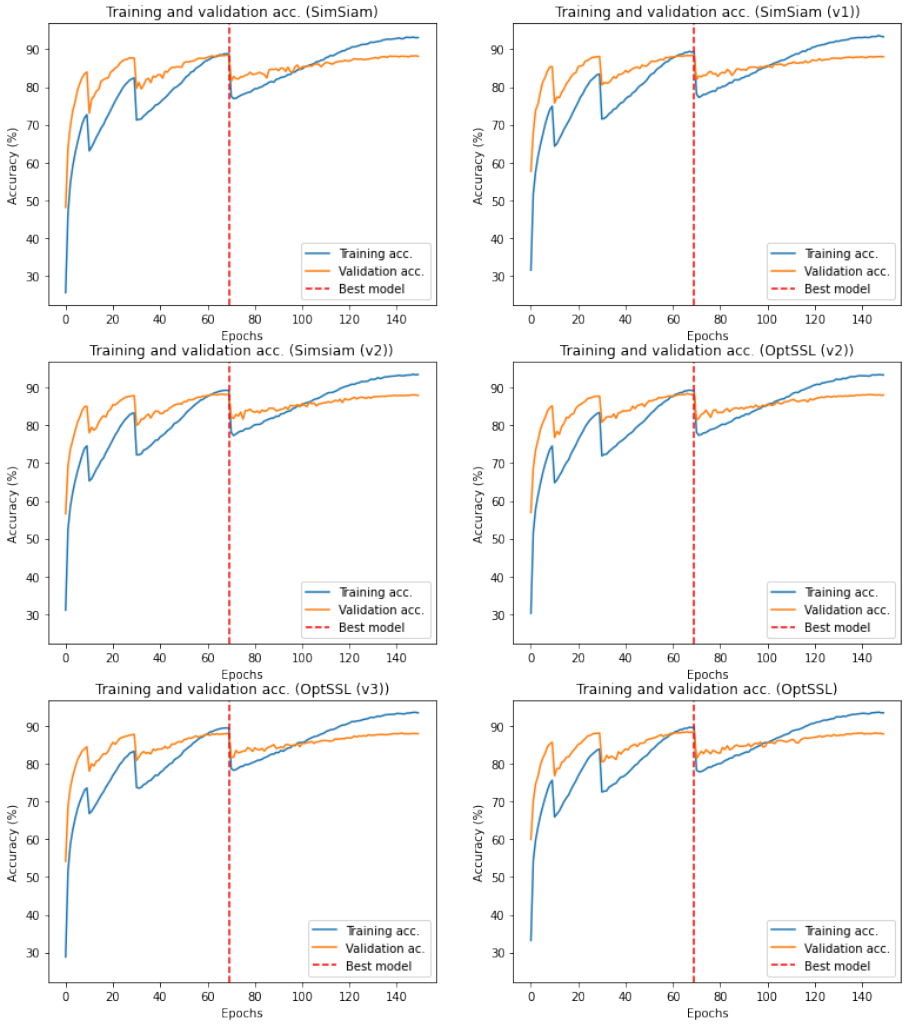


Figure 5.9: Accuracy curves of classifiers training using different SSL methods as the pre-training phase.

Figure 5.10: Images of the Food-101 validation dataset that are predicted with lowest uncertainty. All of them are correctly classified. **Top Left**: image of class "Falafel". **Top Right**: image of class "Cheese plate". **Bottom Left**: image of class "Pulled pork shandwitch". **Bottom Right**: image of class "Shrimp and Grits".

Figure 5.11: Images of the Food-101 validation dataset that are predicted with highest uncertainty. All of them are correctly classified. **Top Left**: image of class "Tiramisu". **Top Right**: image of class "Hot dog". **Bottom Left**: image of class "Chicken curry". **Bottom Right**: image of class "Chicken curry".

Chapter 6

Conclusions and Future Work

In this project, we explored the field of food image analysis and concretely we worked on the food recognition task. The main objective of the project was to explore whether state-of-the-art Self-supervised learning (SSL) techniques could help us improve the performance of a food classifier. However, while developing the project we reached several very interesting conclusions.

Regarding SSL, we analyzed the application of two very recent state-of-the-art models, SimSiam and Barlow Twins (published in 2020 and in 2021 respectively) on food images. After understanding these methods and considering many hypotheses, we were able to introduce modifications to their architectures to develop and present a new SSL model, called OptSSL. We empirically experienced the potential of this method applied on food images and its benefits with respect to other SSL approaches. While searching for the optimal SSL method for food images, we noticed the importance of contrasting both positive and negative samples and the importance of contrasting representations along images instead of features.

With respect to food classification, we proved the potential of using a SSL pre-training phase before training the classifier with the labeled dataset, showing that it can help get a better model in terms of accuracy and uncertainty.

While searching for the best food classifier, we also faced the overfitting problem, from which we extracted some conclusions. We experienced the importance of applying data augmentation techniques and the influence of selecting a good schedule for the learning rate.

Even though we consider that we carried out an extensive work and extracted several conclusions, we believe that there is still exploration to be done in this domain. A possible research line to continue our work is to optimize even more OptSSL by introducing new assumptions in its loss function. Our proposal is to try to move to a Semi-self-supervised Learning paradigm where image labels are considered in the loss function in an ingenious way. Instead of considering that positive pairs are only distorted views of

the same image, we suggest to consider that also distorted views of different images from a same class are positive pairs. It basically means that instead of trying to send all the off-diagonal elements of the cross-correlation matrices to 0 in the loss function, those elements that contrast distorted views of images from the same class are sent to 1, similar to the diagonal elements. In our opinion, this could be an interesting research line to explore.

Regarding the classifier training, we still think that the training process can be optimized and we suggest to investigate more on different training settings to see if SSL pre-training can be even more useful for the classifier.

Finally, another possibility is to try OptSSL in other contexts or study its possibilities for other downstream tasks, such as image detection or image segmentation.

Appendix A

SSL Training Graphics

In this appendix, we show the loss training curves (Fig. A.1) and the monitoring of the k-NN validation accuracy during training (Fig. A.2) of all the experiments presented in Table 5.1. In both figures, the graphics are ordered as follows: *Row 1*: Experiment 1 - Experiment 2 - Experiment 3; *Row 2*: Experiment 4 - Experiment 5 - Experiment 6; *Row 3*: Experiment 7 - Experiment 8 - Experiment 9; *Row 4*: Experiment 10 - Experiment 11 - Experiment 12; *Row 5*: Experiment 13 - Experiment 14 - Experiment 15.

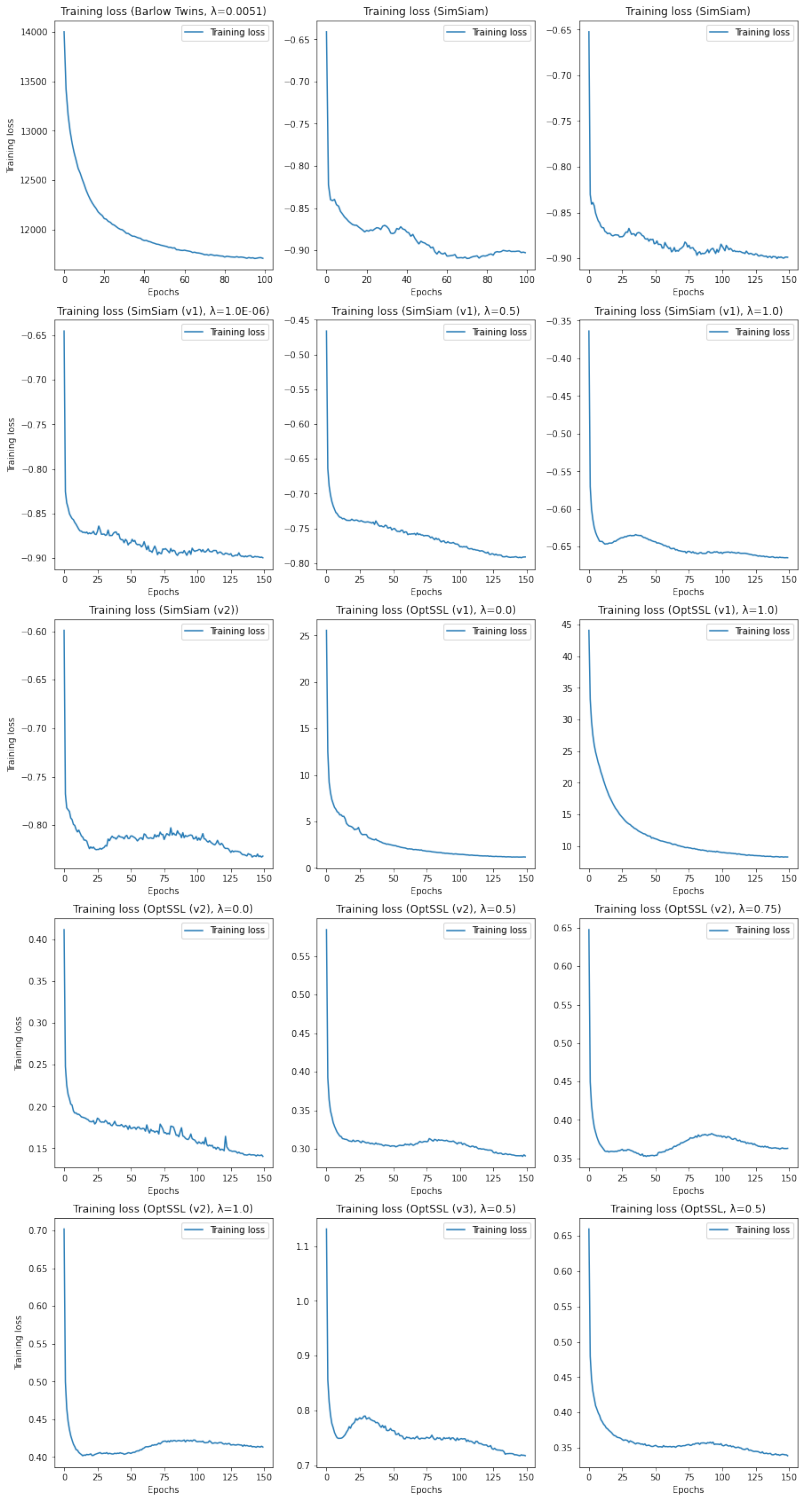


Figure A.1: Training loss curves of all the SSL models trained.

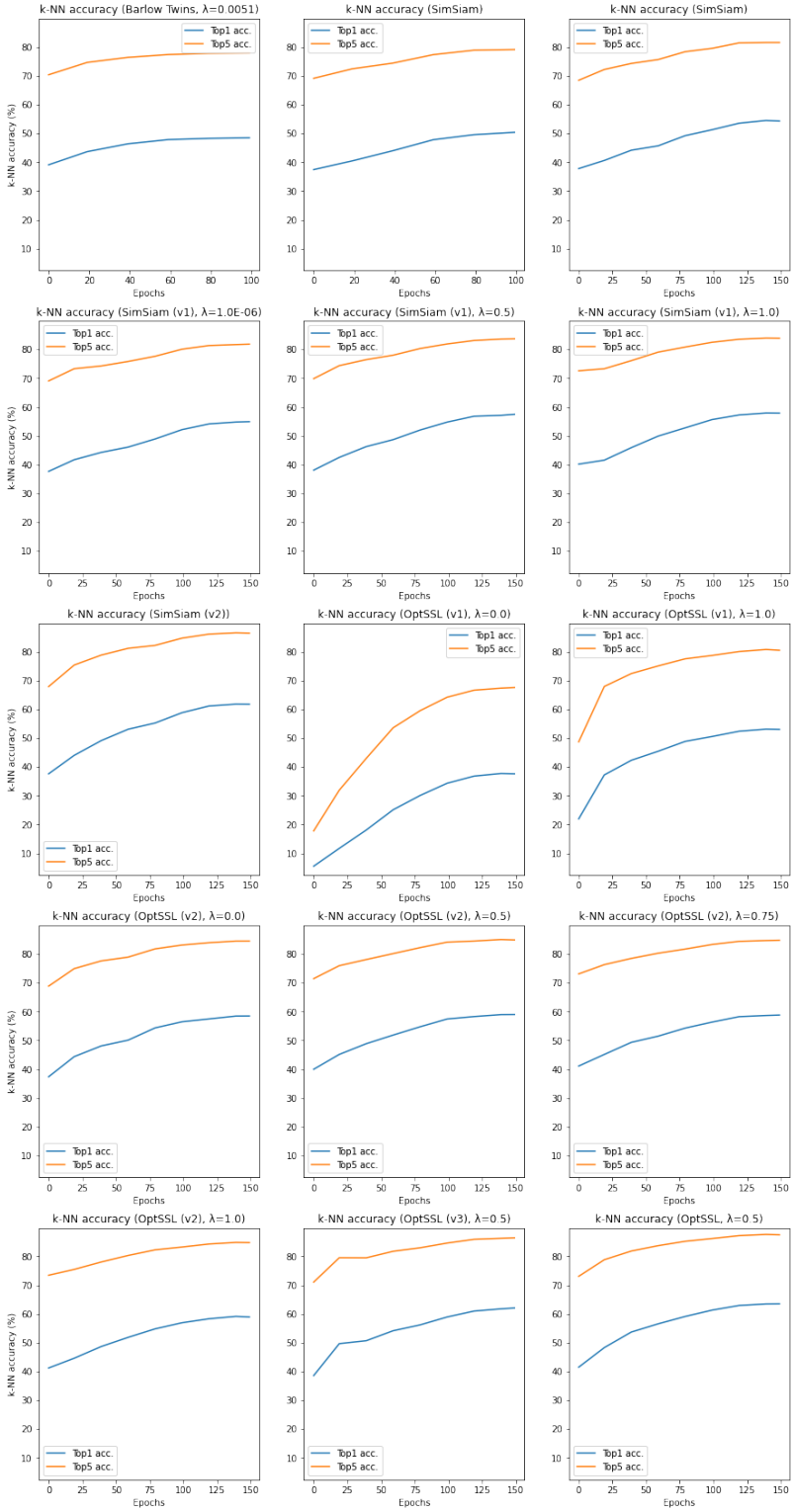


Figure A.2: Validation accuracy (in %) of a k-NN classifier during the training of all SSL models trained.

Bibliography

[1] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul W. Fieguth, Xiaochun Cao, Abbas Khosravi, U. Rajendra Acharya, Vladimir Makarenkov, and Saeid Nahavandi, *A review of uncertainty quantification in deep learning: Techniques, applications and challenges*, CoRR **abs/2011.06225** (2020).

[2] Eduardo Aguilar, Bhalaji Nagarajan, Rupali Khatun, Marc Bolaños, and Petia Radeva, *Uncertainty modeling and deep learning applied to food image analysis*, BIOSTEC, 2020.

[3] Anonymous, *Incident number 16*, AI Incident Database (2015).

[4] Sinem Aslan, Gianluigi Ciocca, Davide Mazzini, and Raimondo Schettini, *Benchmarking algorithms for food localization and semantic segmentation*, International Journal of Machine Learning and Cybernetics **11** (2020).

[5] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli, *wav2vec 2.0: A framework for self-supervised learning of speech representations*, CoRR **abs/2006.11477** (2020).

[6] Horace Barlow, *Possible principles underlying the transformations of sensory messages*, Sensory Communication **1** (1961).

[7] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra, *Weight uncertainty in neural networks*, 2015.

[8] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool, *Food-101 – mining discriminative components with random forests*, European Conference on Computer Vision, 2014.

[9] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah, *Signature verification using a "siamese" time delay neural network*, Proceedings of the 6th International Conference on Neural Information Processing Systems (San Francisco, CA, USA), NIPS'93, Morgan Kaufmann Publishers Inc., 1993, p. 737–744.

[10] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin, *Unsupervised learning of visual features by contrasting cluster assignments*, CoRR **abs/2006.09882** (2020).

[11] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton, *A simple framework for contrastive learning of visual representations*, CoRR **abs/2002.05709** (2020).

[12] Xinlei Chen and Kaiming He, *Exploring simple siamese representation learning*, CoRR **abs/2011.10566** (2020).

[13] Sumit Chopra, Raia Hadsell, and Yann LeCun, *Learning a similarity metric discriminatively, with application to face verification*, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 1, 2005, pp. 539–546 vol. 1.

[14] Marco Cuturi, *Sinkhorn distances: Lightspeed computation of optimal transport*, Advances in Neural Information Processing Systems (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), vol. 26, Curran Associates, Inc., 2013.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, *Imagenet: A large-scale hierarchical image database*, 2009 IEEE Conference on Computer Vision and Pattern Recognition, Ieee, 2009, pp. 248–255.

[16] J. S. Denker and Yann Lecun, *Transforming neural-net output levels to probability distributions*, Advances in Neural Information Processing Systems (NIPS 1990), Denver, CO, April 1991 (R. Lippmann, J. Moody, and D. Touretzky, eds.), vol. 3, Morgan Kaufmann, 1991 (English (US)).

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, *BERT: pre-training of deep bidirectional transformers for language understanding*, CoRR **abs/1810.04805** (2018).

[18] Carl Doersch, Abhinav Gupta, and Alexei A. Efros, *Unsupervised visual representation learning by context prediction*, CoRR **abs/1505.05192** (2015).

[19] Linus Ericsson, Henry Gouk, Chen Change Loy, and Timothy M. Hospedales, *Self-supervised representation learning: Introduction, advances and challenges*, CoRR **abs/2110.09327** (2021).

[20] Yarin Gal, *Uncertainty in deep learning*, Ph.D. thesis, University of Cambridge, 2016.

[21] Yarin Gal and Zoubin Ghahramani, *Bayesian convolutional neural networks with bernoulli approximate variational inference*, 2016.

[22] ———, *Dropout as a bayesian approximation: Representing model uncertainty in deep learning*, 2016.

[23] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna M. Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, Muhammad Shahzad, Wen Yang, Richard Bamler, and Xiao Xiang Zhu, *A survey of uncertainty in deep neural networks*, CoRR **abs/2107.03342** (2021).

[24] Mariana-Iuliana Georgescu, Antonio Barbalau, Radu Tudor Ionescu, Fahad Shahbaz Khan, Marius Popescu, and Mubarak Shah, *Anomaly detection in video via self-supervised and multi-task learning*, CoRR **abs/2011.07491** (2020).

[25] Ethan Goan and Clinton Fookes, *Bayesian neural networks: An introduction and survey*, Lecture Notes in Mathematics (2020), 45–87.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, pp. 164–167, MIT Press, 2016, `http://www.deeplearningbook.org`.

[27] Priya Goyal, Dhruv Mahajan, Abhinav Gupta, and Ishan Misra, *Scaling and benchmarking self-supervised visual representation learning*, CoRR **abs/1905.01235** (2019).

[28] Alex Graves, *Practical variational inference for neural networks*, Proceedings of the 24th International Conference on Neural Information Processing Systems (Red Hook, NY, USA), NIPS'11, Curran Associates Inc., 2011, p. 2348–2356.

[29] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko, *Bootstrap your own latent: A new approach to self-supervised learning*, CoRR **abs/2006.07733** (2020).

[30] Raia Hadsell, Sumit Chopra, and Yann LeCun, *Dimensionality reduction by learning an invariant mapping*, 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), vol. 2, 2006, pp. 1735–1742.

[31] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick, *Momentum contrast for unsupervised visual representation learning*, CoRR **abs/1911.05722** (2019).

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, CoRR **abs/1512.03385** (2015).

[33] Eyke Hüllermeier and Willem Waegeman, *Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction*, CoRR **abs/1910.09457** (2019).

[34] Azumino Inc, *Calorie mama food ai - food image recognition and calorie counter using deep learning*, (2017), `https://caloriemama.ai/`.

[35] Christian Janiesch, Patrick Zschech, and Kai Heinrich, *Machine learning and deep learning*, CoRR **abs/2104.05314** (2021).

[36] Alain Jungo, Raphael Meier, Ekin Ermis, Evelyn Herrmann, and Mauricio Reyes, *Uncertainty-driven sanity check: Application to postoperative brain tumor cavity segmentation*, CoRR **abs/1806.03106** (2018).

[37] Alex Kendall and Yarin Gal, *What uncertainties do we need in bayesian deep learning for computer vision?*, CoRR **abs/1703.04977** (2017).

[38] Alex Guy Kendall, *Geometry and uncertainty in deep learning for computer vision*, Ph.D. thesis, University of Cambridge, 2019.

[39] Armen Der Kiureghian and Ove Ditlevsen, *Aleatory or epistemic? does it matter?*, Structural Safety **31** (2009), no. 2, 105–112, Risk Acceptance and Risk Communication.

[40] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al., *Siamese neural networks for one-shot image recognition*, ICML deep learning workshop, vol. 2, Lille, 2015.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in Neural Information Processing Systems (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[42] Solomon Kullback and Richard A. Leibler, *On information and sufficiency*, The Annals of Mathematical Statistics **22** (1951), no. 1, 79–86.

[43] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell, *Simple and scalable predictive uncertainty estimation using deep ensembles*, 2017.

[44] Fei-Fei Li, Ranjay Krishna, and Danfei Xu, *Cs231n: Convolutional neural networks for visual recognition 2021*.

[45] Chang Liu, Yu Cao, Yan Luo, Guanling Chen, Vinod Vokkarane, and Yunsheng Ma, *Deepfood: Deep learning-based food image recognition for computer-aided dietary assessment*, CoRR **abs/1606.05675** (2016).

[46] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Zhaoyu Wang, Li Mian, Jing Zhang, and Jie Tang, *Self-supervised learning: Generative or contrastive*, CoRR **abs/2006.08218** (2020).

[47] Jonathan Long, Evan Shelhamer, and Trevor Darrell, *Fully convolutional networks for semantic segmentation*, CoRR **abs/1411.4038** (2014).

[48] Ilya Loshchilov and Frank Hutter, *SGDR: stochastic gradient descent with restarts*, CoRR **abs/1608.03983** (2016).

[49] David J. C. MacKay, *A practical bayesian framework for backpropagation networks*, Neural Comput. **4** (1992), no. 3, 448–472.

[50] Niki Martinel, Gian Luca Foresti, and Christian Micheloni, *Wide-slice residual networks for food recognition*, CoRR **abs/1612.06543** (2016).

[51] Daily Milanés-Hermosilla, Rafael Trujillo Codorniú, René López-Baracaldo, Roberto Sagaró-Zamora, Denis Delisle-Rodriguez, John Jairo Villarejo-Mayor, and José Ricardo Núñez-Álvarez, *Monte carlo dropout for uncertainty estimation and motor imagery classification*, Sensors **21** (2021), no. 21.

[52] Yassine Ouali, Céline Hudelot, and Myriam Tami, *An overview of deep semi-supervised learning*, CoRR **abs/2006.05278** (2020).

[53] Claude Elwood Shannon, *A mathematical theory of communication*, The Bell System Technical Journal **27** (1948), 379–423.

[54] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research **15** (2014), no. 56, 1929–1958.

[55] Jianing Sun, Katarzyna Radecka, and Zeljko Zilic, *Foodtracker: A real-time food detection mobile application by deep convolutional neural networks*, CoRR **abs/1909.05994** (2019).

[56] Aäron van den Oord, Yazhe Li, and Oriol Vinyals, *Representation learning with contrastive predictive coding*, CoRR **abs/1807.03748** (2018).

[57] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm, *Deep graph infomax*, 2018.

[58] Guotai Wang, Wenqi Li, Michael Aertsen, Jan Deprest, Sébastien Ourselin, and Tom Vercauteren, *Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks*, CoRR **abs/1807.07356** (2018).

[59] Zhirong Wu, Yuanjun Xiong, Stella X. Yu, and Dahua Lin, *Unsupervised feature learning via non-parametric instance-level discrimination*, CoRR **abs/1805.01978** (2018).

[60] Dejing Xu, Jun Xiao, Zhou Zhao, Jian Shao, Di Xie, and Yueting Zhuang, *Self-supervised spatiotemporal learning via video clip order prediction*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[61] Roman Yampolskiy, *Incident number 52*, AI Incident Database (2016).

[62] Mang Ye, Xu Zhang, Pong C. Yuen, and Shih-Fu Chang, *Unsupervised embedding learning via invariant and spreading instance feature*, CoRR **abs/1904.03436** (2019).

[63] Yang You, Igor Gitman, and Boris Ginsburg, *Scaling SGD batch size to 32k for imagenet training*, CoRR **abs/1708.03888** (2017).

[64] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny, *Barlow twins: Self-supervised learning via redundancy reduction*, CoRR **abs/2103.03230** (2021).

[65] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola, *Dive into deep learning*, arXiv preprint arXiv:2106.11342 (2021).