

Self- and Semi-Supervised Learning for Vehicle Make and Model Recognition in Images

Ricardo Teixeira

UMinho | 2024

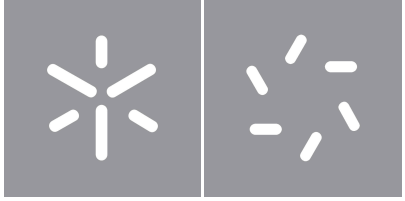


Universidade do Minho
Escola de
Ciências

Ricardo Nuno Alves Teixeira

Self- and Semi-Supervised Learning for Vehicle Make and Model Recognition in Images

October 2024



University of Minho
School of Sciences

Ricardo Nuno Alves Teixeira

Self and Semi-supervised Learning for Vehicle Make and Model Recognition in Images

Master's Dissertation in Mathematics and Computation

Area of Specialization in Computation

Dissertation supervised by
Professora Doutora Maria Fernanda Pires da Costa
Professor Doutor Emanuel Demétrio Mendes Gouveia

december 2024

Copyright and Terms of Use for Third Party Work

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

License granted to users of this work:



CC BY-NC-SA

https://creativecommons.org/licenses/by-nc-sa/4.0/ *[Esta licença permite que outros remisturem, adaptem e criem a partir do seu trabalho para fins não comerciais, desde que lhe atribuam a si o devido crédito e que licenciem as novas criações ao abrigo de termos idênticos.]*

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisors Professor Fernanda Costa and Professor Emanuel Gouveia for their invaluable guidance, unwavering support, and mentorship throughout this dissertation, allowing me to explore the fields of self-supervision and transformers, becoming my main interests in machine learning. Their expertise and insights have been instrumental in shaping this work.

My heartfelt appreciation goes to my family, particularly my mother Helena, for their endless patience, understanding, and emotional support throughout this journey. Also, a special thanks to all my friends for their encouragement and belief in me. Their support throughout my journey was invaluable and allowed me to pursue my academic interests, starting with my degree in Computer Science which gave me the mathematical and computational basis to explore the vast area of machine learning during my Master's in Mathematics and Computation.

I would like to acknowledge DTx-CoLAB for partnering with University of Minho and providing the context, facilities and resources for the development of this dissertation.

This work was produced with the support of HPC-UÉ Oblivion and Vision and it was funded by FCT I.P. under the project Advanced Computing Project 2022.27818.CPCA.A0 .

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, Braga, december 2024

Ricardo Nuno Alves Teixeira

Abstract

Recent advances in computer vision have demonstrated remarkable success in general object recognition tasks. However, fine-grained visual classification remains challenging due to subtle inter-class variations and the requirement for extensive labeled datasets. While self-supervised learning has shown promise in learning rich representations from unlabeled data, its application to specialized domains like fine-grained recognition has been less explored.

This study explores various self-supervised learning architectures, such as a contrastive learning method and a teacher-student framework , namely SimCLR and DINO respectively, to evaluate their effectiveness in fine-grained classification tasks. Specifically, it focuses on the application of self and semi-supervised learning techniques to vehicle make and model recognition in image datasets with few labels, providing a comprehensive analysis of their performance for this specialized classification challenge. We propose a multi-stage framework that leverages a backbone pretrained on the ImageNet dataset, followed by self-supervised fine-tuning on the Stanford Cars dataset, finalizing with few-shot classification with a linear classifier, also on Stanford Cars, to evaluate representation quality. Our experiments reveal significant challenges in domain transfer from general to specialized visual domains. We observe catastrophic forgetting when adapting models from broad object recognition to fine-grained automotive features, particularly pronounced with limited data availability.

This work highlights critical research directions for improving domain adaptation in self-supervised learning, particularly for fine-grained applications with data constraints.

Keywords Self-Supervised Learning, Semi-Supervised Learning, Vision Transformers, Contrastive learning, Self-distillation, Few-shot classification, Fine-grained classification, Dataset with few labeled data

Resumo

Os recentes avanços na visão por computador têm demonstrado um sucesso notável em tarefas gerais de reconhecimento de objectos. No entanto, a classificação visual de granularidade fina continua a ser um desafio devido a variações subtis entre classes e à necessidade de extensas bases de dados anotadas. Embora a aprendizagem auto-supervisionada se tenha revelado promissora na aprendizagem de representações ricas a partir de dados não anotados, a sua aplicação a domínios especializados, como o reconhecimento de objectos de granularidade fina, tem sido menos explorada.

Este trabalho explora várias arquitecturas auto-supervisionadas, como uma abordagem de aprendizagem contrastiva e uma estrutura professor-aluno, SimCLR e DINO respetivamente, analisando a sua eficácia para tarefas de classificação de granularidade fina. Especificamente, foca-se na aplicação de métodos auto-supervisionados e semi-supervisionados em classificação de marcas e modelos de veículos em imagens, assim como numa análise compreensiva da sua performance neste desafio de classificação especializado. Propomos uma estrutura de várias etapas que utiliza um modelo pré-treinado na base de dados ImageNet, seguido de um *fine-tuning* auto-supervisionado na base de dados Stanford Cars, finalizando com uma classificação com poucos exemplos usando um classificador linear, também no Stanford Cars, para avaliar a qualidade das representações. As nossas experiências revelam desafios significativos na transferência de domínios visuais gerais para domínios visuais especializados. Observamos um esquecimento catastrófico ao adaptar modelos de reconhecimento de objectos gerais a caraterísticas automóveis subtis, particularmente pronunciado pela disponibilidade de dados limitada.

Este trabalho destaca direcções claras de investigação para melhorar a adaptação ao domínio na aprendizagem auto-supervisionada, em particular para aplicações de granularidade fina com restrições de dados.

Palavras-chave Aprendizagem Auto-Supervisionada, Aprendizagem Semi-Supervisionada, Transformers de Visão, Classificação com poucos exemplos, Aprendizagem Contrastiva, Auto-distilação, Classificação com poucos exemplos, Classificação de granularidade fina, Conjunto de dados com poucos dados anotados

Contents

1 Introduction **1**

1.1 Context and Motivation . . . 1

1.2 Outline . . . 2

2 Computer Vision **4**

2.1 Convolutional Neural Networks . . . 5

2.1.1 Convolutional layers . . . 5

2.1.2 Padding and Striding . . . 7

2.1.3 Pooling layers . . . 8

2.1.4 Final architecture . . . 9

2.1.5 Noteworthy Features . . . 9

2.2 ResNets . . . 10

2.2.1 Residual Learning . . . 10

2.2.2 Residual blocks and overall architecture . . . 11

2.2.3 Advantages . . . 12

2.3 Transformer . . . 12

2.3.1 Architecture . . . 14

2.3.2 Input Embedding . . . 14

2.3.3 Attention . . . 16

2.3.4 Multi-Head Attention . . . 16

2.3.5 Applications of Attention . . . 17

2.3.6 Point-Wise Feed-Forward Network . . . 18

2.3.7 Advantages . . . 18

2.4 Vision Transformer (ViT) . . . 18

2.4.1 Patch Embeddings . . . 19

2.4.2 Class Token . . . 19
2.4.3 Positional Encoding . . . 20
2.4.4 Advantages . . . 20

3 Self-Supervised Learning 21

3.1 Types of self-supervised learning methods . . . 22
3.1.1 Autoassociative or Self-predictive Learning . . . 22
3.1.2 Joint embedding contrastive SSL . . . 22
3.1.3 Joint embedding non-contrastive SSL . . . 23
3.2 Collapse . . . 23
3.3 SimCLR: Contrastive SSL framework . . . 24
3.3.1 Method . . . 25
3.3.2 Data Augmentation . . . 25
3.3.3 Backbone . . . 26
3.3.4 Projector . . . 26
3.3.5 Loss . . . 26
3.3.6 Batch size . . . 27
3.3.7 Avoiding collapse . . . 27
3.4 DINO: Non-contrastive SSL framework . . . 28
3.4.1 Method . . . 28
3.4.2 Data augmentation . . . 29
3.4.3 Teacher and Student Networks . . . 29
3.4.4 Teacher and Student weight updates . . . 30
3.4.5 Avoiding collapse . . . 30

4 Semi-Supervised Learning 32

4.1 Types of Semi-supervised learning methods . . . 32
4.1.1 Deep generative methods . . . 32
4.1.2 Consistency regularization methods . . . 33
4.1.3 Graph-based methods . . . 33
4.1.4 Pseudo-labelling methods . . . 34
4.2 Transfer Learning from Self-Supervised Models . . . 34
4.2.1 Advantages . . . 35

4.2.2 Challenges . . . 35
4.2.3 Strategies . . . 36
4.3 Fine-tuning self-supervised models with few examples . . . 36
4.3.1 Advantages . . . 37
4.3.2 Challenges . . . 37
4.3.3 Strategies . . . 38

5 Technical concepts and metrics **39**
5.1 Cosine annealing schedule . . . 39
5.2 One epoch learning rate range test . . . 40
5.3 Collapse Analysis . . . 42
5.4 K-nearest neighbours classifier . . . 43
5.4.1 KNN retrieval . . . 44

6 Numerical Experiments **45**
6.1 Pretrained model (baseline) . . . 45
6.2 Fine-tuning using DINO . . . 46
6.2.1 Default parameters . . . 47
6.2.2 Hyperparameter tuning: Learning rate . . . 47
6.2.3 Hyperparameter tuning: Center Momentum . . . 51
6.2.4 Scaling . . . 52
6.2.5 Partial pretrained model . . . 55
6.3 Training from scratch . . . 57
6.4 Embedding space analysis . . . 59
6.4.1 Collapse analysis . . . 59
6.4.2 Neighbour retrieval . . . 61
6.5 DINO vs SimCLR . . . 70
6.6 Few-shot classification . . . 71

7 Conclusions and Future Work **73**
7.1 Conclusions . . . 73
7.2 Future work . . . 74

List of Figures

1 1D convolution example . . . 6
2 2D convolution (the blue region is the receptive field of the red neuron) . . . 7
3 Example of (2×2) maxpooling with stride $= 2$ . . . 9
4 Architecture of the LeNet . . . 9
5 Residual block . . . 11
6 Architecture of the ResNet-34 compared to VGG-19 and a plain CNN with 34 layers . . . 13
7 Architecture of the Transformer . . . 15
8 Attention mechanism . . . 17
9 Architecture of the Vision Transformer . . . 19
10 Patterns of collapse . . . 24
11 SimCLR architecture . . . 25
12 DINO architecture . . . 29
13 Example of learning rate during training using cosine annealing schedule without warm restarts . . . 40
14 Comparing the effects of learning rate on optimizer steps during gradient descent . . . 41
15 Comparing the effects of learning rate on loss values during training . . . 41
16 Example of collapse analysis of an embedding space . . . 43
17 Results from training with default parameters for 100 epochs . . . 47
18 One epoch learning rate range test. Starting Lr values - Orange: 5×10^{-9}, Dark Blue: 5×10^{-8}, Red: 5×10^{-7}, Light Blue: 5×10^{-6}, Pink: 5×10^{-5} . . . 48
19 One epoch learning rate range test with SGD. Starting Lr values - Orange: 5×10^{-9}, Dark Blue: 5×10^{-8}, Red: 5×10^{-7}, Light Blue: 5×10^{-6} . . . 49
20 Loss across one epoch with Learning rate set to 0 . . . 50

21 One epoch learning rate range test. Starting Lr values - Red: 5×10^{-4}, Blue: 5×10^{-5}, Pink: 5×10^{-6}, Green: 5×10^{-7} 50
22 Center momentum test: Values - Dark Blue: 0.97, Red: 0.999, Pink: 0.9997, Light Blue: 0.9999 52
23 600 epochs with fixed learning rate, fixed teacher momentum and default center momentum (0.9) 53
24 600 epochs with fixed learning rate, fixed teacher momentum and fixed center momentum (0.9999) 54
25 600 epochs with fixed Learning rate, fixed teacher momentum and center momentum schedule 55
26 1000 epochs with reset layers, fixed Learning rate and fixed teacher momentum 56
27 1000 epochs training from scratch with fixed Learning rate and fixed teacher momentum 57
28 Collapse analysis of the output after teacher backbone at several points during training. . 60
29 Projector structure 61
30 Collapse analysis of the output after projector at several points during training. 62
31 Collapse analysis of the output after intermediate projector layer at several points during training. 63
32 KNN neighbour retrieval from pretrained model. 64
33 KNN neighbour retrieval from 600 epochs with fixed Learning rate and fixed teacher momentum. 65
34 KNN neighbour retrieval from 600 epochs with fixed Learning rate, fixed teacher momentum and fixed center momentum. 66
35 KNN neighbour retrieval from 600 epochs with fixed Learning rate, fixed teacher momentum and center momentum schedule. 67
36 KNN neighbour retrieval from 1000 epochs with reset layers, fixed Learning rate and fixed teacher momentum. 68
37 KNN neighbour retrieval from 1000 epochs from scratch with fixed Learning rate and fixed teacher momentum. 69
38 1000 epochs training Dino (red) vs SimCLR (blue) 71

List of Tables

1 Summary of the experiments 58
2 Few-shot classification results 72

Chapter 1

Introduction

1.1 Context and Motivation

Modern toll collection systems are undergoing a significant transformation, moving away from traditional manual methods toward automated solutions that can identify vehicles and process payments without requiring vehicles to stop. In solutions for automatic toll fee charge, one of the strategies is the automatic recognition of license plates in vehicle pictures in order to identify who should be charged the fee. However several factors can impact the reliability of the identification of the driver/car owner, such as environmental conditions or damage to the license plate impacting the accuracy of the license plate recognition system. These limitations carry financial implications such as revenue loss from missed identifications or false positives leading to incorrect charges and costs from legal dispute resolutions. One way to reduce the number of incorrect charges due to errors in license plate identification is to combine it with other available context information, such as vehicle make and model. Crossing this auxiliary information with the records associated with a certain predicted license plate allows us to be more confident in the predictions of the license plate detection system, resulting in higher billing accuracy and enabling the detection of plate swapping/prevention of fraud.

The classification of vehicle make and model in color images can be carried out using deep learning algorithms, which in recent years have proven themselves capable of solving this problem as well as other problems with the same fine-grained nature (distinguishing between classes that differ from each other in very subtle ways) even in the presence of significant variations in pose and lighting. However, the application of deep learning algorithms in a real scenario presents its own set of challenges, with the main one being the limitation of labeled datasets, as manual labeling is both costly and time-intensive. It also demands regular re-training of the deployed models with many new labeled examples in order to accommodate the recognition of new vehicle models. These limitations motivate the use of self-supervised learning and semi-supervised learning techniques that leverage large amounts of available unlabeled data

while requiring no labeled data or only small amounts of labeled data respectively.

Self-supervised learning techniques offer a promising solution by generating supervisory signals from the data itself and leverage natural relationships within the data, enabling feature learning from vehicle images and multi-view consistency from different cameras, eliminating the need for labeled examples. Similarly, semi-supervised learning methods provide an efficient approach by combining small amounts of labeled data with large quantities of unlabeled data. This is accomplished through various implementation strategies including the use of deep generative models, consistency regularization and pseudo-labeling approaches.

This dissertation aims to contribute to the solution of the problems described by enhancing the understanding of these topics and then exploring a path that combines both approaches, minimizing the use of labeled data, in order to classify the make and model of vehicles in images from a public dataset. Thus, the goal is to rely on pretrained models, improving their representations with self-supervised fine-tuning, in order to classify images from fine-grained classes with the smallest amount of annotated data possible. The application to vehicle make and model will be used only as a case study of this generalist approach, and no attempt will be made to actually verify the performance of the license plate recognition system with the addition of make and model classification.

This dissertation presents a concept of a framework that begins with a model pretrained on a large general image dataset using self-supervision. In the first stage, this pretrained model will be fine-tuned to the particular domain using the same self-supervised method as the one from pretraining. The final stage consists of a supervised training of a linear classifier on top of frozen features, using few labeled examples per class, leveraging the pretrained representations fine-tuned to the target domain, without relying on an extensive and costly labeled dataset.

The work of this dissertation included an attempt to repurpose the self-supervised learning frameworks, originally intended for pretraining, to a fine-tuning stage. It also includes a comparison of this adaptation using different self-supervised approaches.

1.2 Outline

This dissertation focuses on exploring self-supervised and semi-supervised techniques and their application to solve tasks in the computer vision domain. We aim to provide an understanding of these topics based on formal mathematical definitions supported by intuitive explanations.

The outline of this dissertation can be summarized as follows:

- **Computer Vision (chapter 2):**
  - Brief context of the computer vision domain's history and developments.
  - Detailed analysis of prominent deep learning architectures in computer vision that will serve as the backbone for the frameworks used later.
- **Self-supervised learning (chapter 3):**
  - Overview of the paradigm and survey of groups of frameworks.
  - Presentation and detailed analysis of two self-supervised frameworks representative of distinct groups.
- **Semi-supervised learning (chapter 4):**
  - Overview of the paradigm and survey on families of methods.
  - Description of techniques based on pretraining.
- **Technical concepts and metrics (chapter 5):**
  - Definition of metrics and techniques used in the experiments.
- **Experimental Results (chapter 6):**
  - Description of the framework developed.
  - Analysis of the performance of the framework developed.

Chapter 2

Computer Vision

According to IBM, "Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs — and take actions or make recommendations based on that information". Furthermore, "if AI enables computers to think, computer vision enables them to see, observe and understand" (IBM).

Therefore, the goal of Computer Vision is to train machines to perform tasks on visual inputs, such as image classification or object detection, akin to human kind. To achieve this goal, in the same way that humans use years of visual inputs as context to solve this type of problems, Computer Vision requires a huge amount of data, as well as computing power to process it. Fortunately, these requirements are getting easier to fulfill as models and hardware have evolved dramatically in recent years and data is readily available on the internet. As of today, these advancements have allowed Computer Vision models to achieve better performance than humans in some tasks like tumor identification in patients ahead of time, allowing for a swifter and more efficient treatment.

Computer Vision started in 1957 at the American National Institute for Standards and Technology, where a team led by Russell Kirsch made the first digital scan of an image. Two years later, in 1959, two neurophysiologists, David Hubel and Torsten Wiesel decided to run an experiment in which they showed a cat an array of images and measured the activation of the neurons in response with electrodes (Hubel and Wiesel, 1959), reaching the conclusion that visual processing starts with simple features like lines or edges. Later in 1982, the neuroscientist David Marr built on this work by establishing that vision works hierarchically.

In 1963, Lawrence Roberts, who is considered to be the godfather of Computer Vision, introduced in his PhD thesis a method for getting information about a 3D object from a 2D image.

Almost 20 years latter, in 1980, computer scientist Kunihiko Fukushima introduced the Neocognitron (Fukushima, 1980), a neural network with convolutional layers, the precursor of modern Convolutional Neural Networks (CNNs) and, in 1989, computer scientist Yann LeCun expanded on Fukushima's work by

successfully applying backpropagation to a variant of the Neocognitron (LeCun et al., 1989).

The biggest jump in Computer Vision and Machine Learning came to be in 2012, when a team from the University of Toronto led by Alex Krizhevsky won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) with a CNN, the AlexNet (Krizhevsky et al., 2012), achieving 15.3% top-5 error in the image classification task of that competition, 10.8 percentage points lower than the runner up. After this breakthrough, new state-of-art models based on CNNs and other neural networks managed to reach single digit error rates and, in 2017, 29 of 38 competing teams in In ILSVRC had lower than 5% error rates.

2.1 Convolutional Neural Networks

2.1.1 Convolutional layers

The convolutional layers are the cornerstone of CNNs, playing the role of extracting spatial features from the input data. The foundation of convolution layers is the convolution operation. In mathemathics, the convolution is an operation that combines two different functions, producing a third. This operation is useful in neural networks when replacing the functions with finite-length vectors (functions defined in a finite set of points).

Let $x = (x_1, x_2, ..., x_n)^T$ be an input vector and $w = (w_1, w_2, ..., w_k)^T$ a vector of weights called a kernel (or filter), with $k \leq n$. The cross-correlation of x and w ($x \star w$) is a vector of length $(n - k + 1)$ where the i-th element is defined as (adapted from (Zaki et al., 2020)):

$$(x \star w)_i = \sum_{j=1}^{k} x_{i+j-1} \cdot w_j \tag{2.1}$$

This operation can be interpreted visually as a sliding window of size k along the input vector where we compute the dot product of the input window at each iteration and the kernel.

If we reverse the order of the weights ($w = (w_k, ..., w_2, w_1)$), we have a convolution instead of cross-correlation. Although the operation used in convolution layers is the cross-correlation, in literature the term "convolution" is usually used instead (Murphy, 2022). We will also refer to cross-correlation with this term from now on.

By padding the input vector x with values in the beginning and the end of the sequence, we can now manipulate the size of the output vector. For example, when considering a weight vector of length 3, by prepending and appending one extra value to x (typically 0) we obtain a convolution operation where the output size is the same as the input size (figure 1).

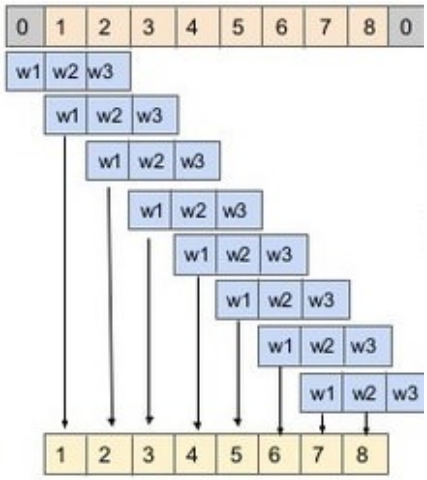


Figure 1: 1D convolution example

For image processing, the convolution operation is particularly relevant when extended to matrix input (2D convolution). In this case, the input is a two-dimensional pixel matrix $X$and the kernel W, also a two-dimensional matrix, with $k \leq n$:

$$X_{n\times n} = \begin{pmatrix} x_{1,1} & \dots & x_{1,n} \\ \dots & \dots & \dots \\ x_{n,1} & \dots & x_{n,n} \end{pmatrix} \quad W_{k\times k} = \begin{pmatrix} w_{1,1} & \dots & w_{1,k} \\ \dots & \dots & \dots \\ w_{k,1} & \dots & w_{k,k} \end{pmatrix} \tag{2.2}$$

The output (without any padding) is a $(n-k+1)\times(n-k+1)$ matrix called activation map where the element in the (i,j) position can be understood as the activation of a corresponding neuron and is defined as (adapted from (Zaki et al., 2020)):

$$(X*W)_{i,j} = \sum_{a=1}^{k}\sum_{b=1}^{k} x_{i+a-1,j+b-1}\cdot w_{a,b} \tag{2.3}$$

We assumed a square input for simplicity but in image processing, X usually has different dimensions, denoted as H for height and W for width.

Each neuron in the convolutional layer has a local receptive field, which refers to the portion of the input image that is used in the dot product with the kernel to compute the activation of that neuron (figure 2). This operation is equivalent to a fully connected layer where neurons share weights (Venkatesan and Li, 2018) because the same kernel of weights is used across the whole image, and where most weights are set to 0, because they connect to input pixels outside the local receptive field.

In order to process RGB images, we need to extend the operation to the 3D convolution. In this case both input and kernel matrices have a third dimension m and r respectively, with $r \leq m$, corresponding to the channel dimension and the output is a $(n-k+1)\times(n-k+1)\times(m-r+1)$ tensor. However,

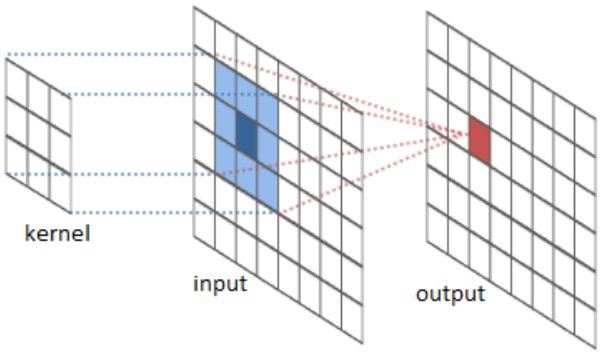


Figure 2: 2D convolution (the blue region is the receptive field of the red neuron)

with CNNs, $r = m$. Since there is no freedom to move in the third dimension, the activation map is a $(n - k + 1) \times (n - k + 1)$ 2D matrix where the element in the (i, j) position is defined as (adapted from (Zaki et al., 2020)):

$$(X * W)_{i,j} = \sum_{a=1}^{k} \sum_{b=1}^{k} \sum_{c=1}^{r=m} x_{i+a-1,j+b-1,c} \cdot w_{a,b,c} \tag{2.4}$$

When considering multiple kernels, we perform the convolution between the input image and all kernels, producing one activation map for each kernel. Stacking the activation maps for all kernels forms the full output volume of the convolutional layer.

2.1.2 Padding and Striding

As seen in both 2D and 3D cases of convolution, the size of the activation maps always decreases with each layer limiting the number of layers and the depth of the network. Also, in a convolution of a kernel and an image, the pixels on the border of the image are featured in fewer receptive fields, leading to a loss of information on the border of the image along the network. A solution to both of these problems is to pad each tensor in the height and width dimensions in each channel by some default value (usually 0). The convention is to pad by adding the same number of rows at the top and the bottom, and the same number of columns on the left and on the right. Padding extends the boundaries of the input, ensuring every pixel receives equal treatment during convolution and allows us to preserve the spatial dimensions of feature maps throughout the network in order to stack as many convolutional layers as needed (Zaki et al., 2020).

Every pixel in the activation map is generated by a linear transformation of the receptive field. As such, neighboring activation map pixels will be similar due to partial overlapping of the corresponding receptive fields. To reduce this redundancy and speed up computation, we can introduce a stride s in the

convolution in order to skip certain receptive fields. The 3D convolution is now defined as (adapted from (Zaki et al., 2020)):

$$(X*W)_{i,j} = \sum_{a=1}^{k}\sum_{b=1}^{k}\sum_{c=1}^{r=m} x_{(i\times s)+a-s,(j\times s)+b-s,c} \cdot w_{a,b,c} \tag{2.5}$$

Larger strides produce smaller outputs, and less receptive field overlapping while smaller strides produce larger outputs and more receptive field overlapping, with the default value corresponding to no stride being $s = 1$ (Murphy, 2022).

The dimension of each activation map of a convolutional layer is defined by 2.6.

$$\left(\frac{H_{in} - 2 \times p_H - H_{kernel}}{s_H} + 1\right) \times \left(\frac{W_{in} - 2 \times p_W - W_{kernel}}{s_W} + 1\right) \times N_{kernels} \tag{2.6}$$

where $(H_{in} \times W_{in} \times c)$ is the input dimensions, $(H_{kernel} \times W_{kernel} \times c)$ is the kernel dimensions, p_H and p_W are the vertical and horizontal padding respectively, s_H and s_W are the vertical and horizontal stride respectively, and $N_{kernels}$ is the number of kernels.

2.1.3 Pooling layers

Pooling is a form of non-linear down-sampling and serves to reduce the spatial size of the activation maps, reducing the number of parameters in the network and the amount of computation power required. By pooling we are in general reducing the data entropy (randomness withing the dateset) and it allows us to impose some invariances among features spatially (Venkatesan and Li, 2018). While there are several forms of pooling, max-pooling is preferred in CNNs as it picks out the strongest neuron activation, which represents the best match to the feature that the kernels are 'looking for' in the convolutional layer, allowing the network to keep most of the information. In a max-pooling layer, we perform a similar operation to convolution (2.4), replacing the sum operators with the maximum operator (or other operators such as average in avg-pooling layers), as shown in 2.7. Also, in all pooling layers, the weights of the kernel are set to 1 in order to aggregate the input values without manipulating any information.

$$max_pool(X,W)_{i,j} = \max_{\substack{a=1,2,\ldots,k \\ b=1,2,\ldots,k \\ c=1,2,\ldots,k}} \{x_{(i\times s)+a-s,(j\times s)+b-s,c} \cdot w_{a,b,c}\} \quad (Zakiet\ al.,\ 2020) \tag{2.7}$$

A common implementation of max-pooling in CNNs is a layer with a 2×2 kernel and a stride of 2, which subsamples every activation map in the input by 2 in height and width (example in figure 3). In earlier CNN architectures, it was common practice to have a pooling layer following every convolutional

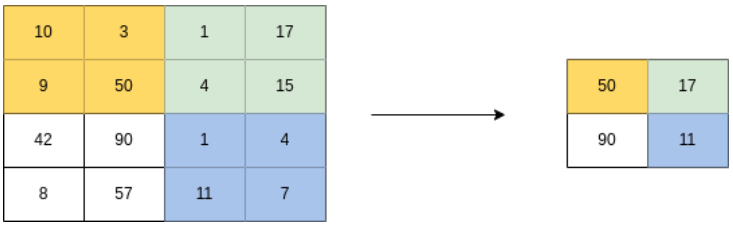


Figure 3: Example of (2×2) maxpooling with stride $= 2$

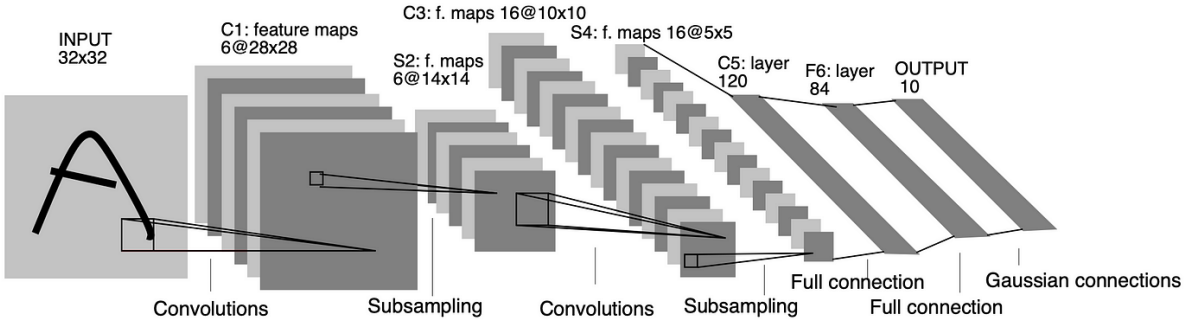


Figure 4: Architecture of the LeNet (LeCun et al., 1998)

layer, however due to concerns about loss of information, there is a recent trend towards using smaller filters, replace them with strided convolutions or even discarding them altogether.

2.1.4 Final architecture

The main block of a basic CNN consists of a convolutional layer followed by a pooling layer (usually a max-pooling layer) and finally a non-linear activation layer (ReLU, hyperbolic tangent, sigmoid function, etc). The final network is a stack of convolutional blocks with varied number of kernels and sizes which in turn feed the information gathered from the image to a small stack of fully connected layers for processing. If our network is intended to be used in a classification problem, the output of the last fully connected layer is fed to a softmax layer. An example of a CNN network is the LeNet-5 (LeCun et al., 1998) (figure 4)

2.1.5 Noteworthy Features

The CNN architecture provides several distinguishing features when compared to the Multi-Layered Perceptrons (MLPs) previously used for computer vision related tasks.

Each neuron of a convolutional layer is only connected to its receptive field and activates when a certain feature is identified by the kernels in that field. However, unlike MLPs, the weights (kernels) are shared across all neurons, which means that increasing the dimension of the inputs does not increase

memory usage nor operation complexity and only increases the number of operations, allowing the use of deeper networks even with high resolution images. This also means that the activation maps are invariant to shifts of the locations of the features making CNNs effective in tasks where the spatial arrangement of features is more critical than their absolute location. While pooling layers are used to downsample the spatial dimensions of the feature maps, reducing computational complexity, they also enhance translation invariance.

Another characteristic is the hierarchical learning of features, where each layer of the CNN captures increasingly higher level features, with lower layers capturing low level and local features like edges, and higher layers capturing high level and global features, usually with substantial meaning to humans.

These features have provided CNNs with mainstream usage in computer vision tasks, allowing them to achieve better generalization over MLPs

2.2 ResNets

As stated before, CNNs learn more complex features in the last layers of the model when compared with the first and middle layers. Intuitively, stacking more convolutional layers should allow the model to learn more complex features and perform better at any task. However, this turned out to not be the case, as increasing the depth of the model by concatenating more convolutional layers leads to a degradation problem where training accuracy saturates and starts to degrade rapidly (He et al., 2015).

Let us consider a relatively shallow architecture and a deeper version that adds more layers onto it. By constructing the deeper network as a copy of the shallow network and the remaining layers being the identity mapping, we can conclude that the deeper model should produce no higher training errors than its shallow version. However, the degradation problem suggests that the solvers have problems approximating the identity mapping by multiple layers. The solution presented in (He et al., 2015) is to adopt residual learning in the network.

2.2.1 Residual Learning

Let us consider $H(x)$ as the mapping to be approximated by a few stacked layers and x the input to this stack. If we assume these layers can approximate $H(x)$ then we can also assume they can approximate the residual function $F(x) = H(x) - x$. By letting these layers approximate $F(x)$ instead and add x to the output, we approximate our original mapping $H(x) = F(x) + x$, however it should be easier for the layers to learn $F(x)$. In the case of the optimal solution being the identity mapping $H(x) = x$, the stack

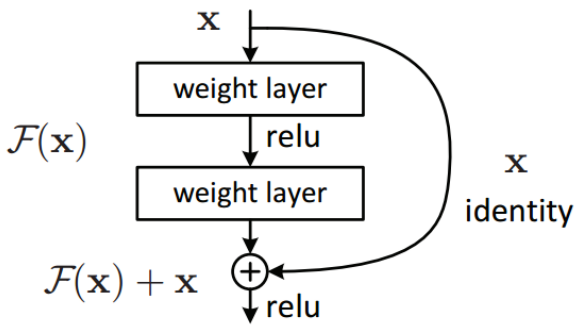


Figure 5: Residual block (He et al., 2015)

of layers only has to push its weights close to zero ($F(x) = 0$) to approximate $H(x)$ correctly, which is easier than to fit an such mapping by a stack of nonlinear layers.

2.2.2 Residual blocks and overall architecture

The implementation of residual learning in ResNets comes in the form of blocks with two convolutional layers and a shortcut connection from the input to the output of the layers. The output of the layers and the input coming from the shortcut connection are then added element-wise (Figure 5).

However, this addition is not always possible due to dimension reduction along the network, making the input and the output dimensions not match. At several points in the network, the stride of the convolution is changed to 2, halving the height and width of the output. In order to keep the computational power required consistent along the network, the number of kernels is doubled. In these cases, the dimension of the input and output have a dimensional mismatch in height, width and channels. In order to match the number of channels, the authors propose two options to change these specific shortcut connections. The first is to zero-pad the input, adding zero entries to the additional dimensions. Although no extra weights are added, we are wasting computational power on meaningless information. The second option is to use 1×1 convolutions to match dimensions. This option adds extra weights to the network in negligible amounts, but comes with the benefits that only useful information is used. Both options are performed with stride 2 in order to match the height and width dimensions along with the channels. The second option is recommended due to the benefits outweighing the drawbacks.

Inspired by the VGG-19 architecture (Simonyan and Zisserman, 2015), one of the most used networks at the time, the authors designed a plain network with 34 layers. It shares the same kernel sizes and the philosophy of doubling the number of kernels when the height and width dimensions are halved to keep a steady computation requirement across the board. Due to the difference in the frequency of doubling the kernel numbers as well as the number of layers in the final fully connected portion, this new design

requires way less floating point operations (FLOPs), decreasing from 19.5 billion FLOPs to 3.6 billion FLOPs. The final ResNet design is completed by adding shortcut connections every couple layers, as shown in figure 6, where dotted line represent shortcut connections where dimension matching occurs and solid line represents standard identity shortcut connections.

2.2.3 Advantages

With the addition of residual learning in the form of shortcut connections to CNNs, ResNets mitigate vanishing gradient and degradation problems, allowing the training of deeper networks with hundreds of layers (He et al., 2015). Deeper networks in turn facilitate the learning of intricate hierarchical features that would not be captured in shallow networks, allowing them to outperform other classic networks in more complex problems. The learning of residual functions means that the network effectively adds or modifies features rather than completely overwriting them. This mechanism allows for increased model capacity, enhancing the ability of the network to learn complex relationships in the data.

Due to these advantages, ResNets have demonstrated exceptional performance in image classification benchmarks and major competitions, such as the ImageNet Large Scale Visual Recognition Challenge (Russakovsky et al., 2015), becoming the uncontested choice for supervised learning and for the backbone of several self-supervised learning and reinforcement learning frameworks.

2.3 Transformer

The state-of-the-art approaches to sequence modeling involved the use of Recurrent Neural Networks (RNNs), namely LSTMs (Long Short-Term Memory) (Hochreiter and Schmidhuber, 1997) and GRUs (Gated Recurrent Units) (Cho et al., 2014), which process the input sequentially, one time-step at a time. At each step, hidden states are updated as a function of the hidden states in the previous step and the input for the current time-step (or words in the case of Natural Language Processing). However, there were two problems with these networks that the community was trying to improve on. First, the short-term memory problem or the inability to capture relations between two tokens in longer sequences. As we process the sequence and the information is loaded into the hidden states, information from the beginning of the sequence loses relevance. The second problem is that the sequential nature of RNNs does not allow for parallelization, resulting in networks that are heavier to train, especially for longer input sequences. Although LSTMs and GRUs were designed to alleviate the first problem, there was still a limit to how long the sequences could be to prevent information loss.

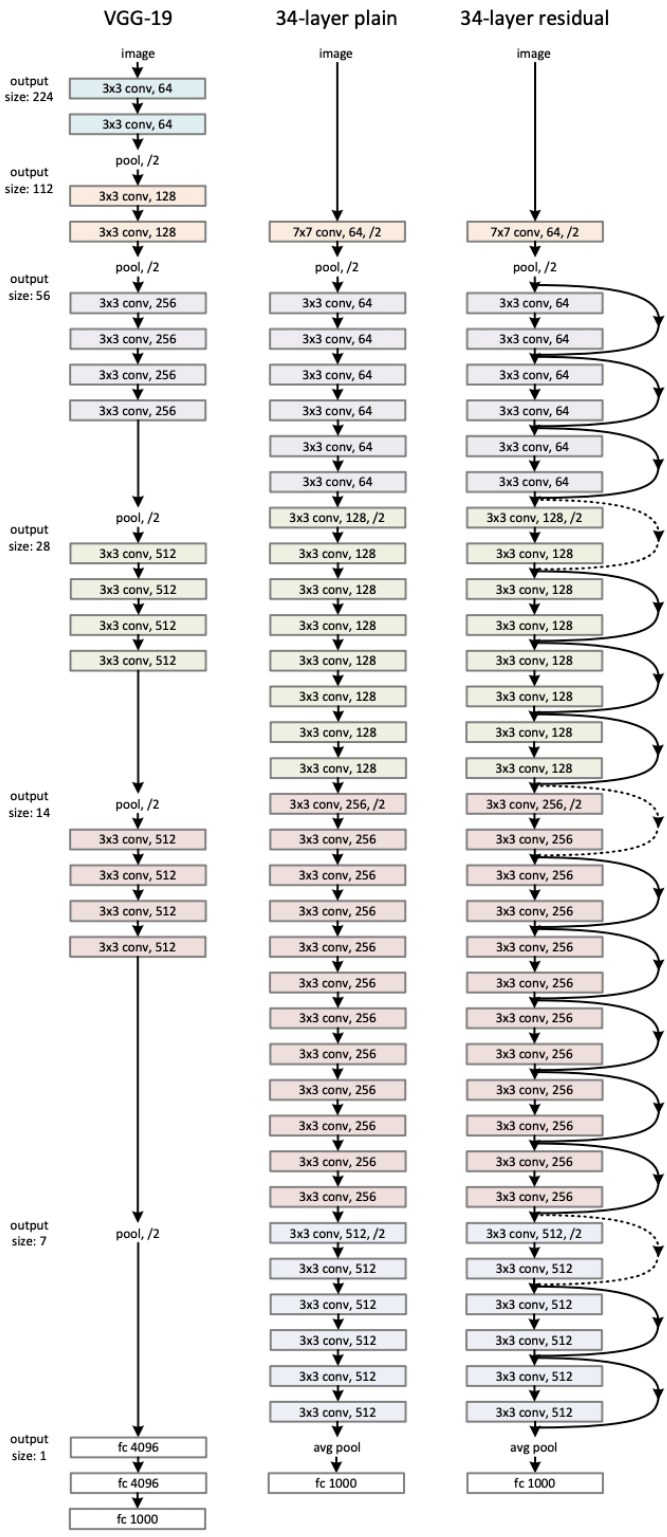


Figure 6: Architecture of the ResNet-34 compared to VGG-19 and a plain CNN with 34 layers (He et al., 2015)

In (Vaswani et al., 2023) a new architecture was introduced, the Transformer, that revolutionized the Natural Language Processing (NLP) world. By using only attention mechanisms, the Transformer was able to solve both of RNNs main problems as, in theory, attention is able to capture relations between any tokens of the sequence independently of their distance and has high parallelization potential. Although the focus of this dissertation is Computer Vision and not NLP, understanding the Transformer architecture is very important as it is the basis for the Vision Transformer, an application of the Transformer architecture for image related tasks.

2.3.1 Architecture

The Transformer is an attention based encoder-decoder type architecture (figure 7). The encoder maps the input sequence to a continuous representation that holds all the learnt information of that input. The decoder then takes that representation and, step by step, generates a single output token while also being fed the previous outputs. The encoder is a stack of identical layers (the number of layers in (Vaswani et al., 2023) is 6 but it may be more) composed of two sub-modules, the multi-head attention module and the fully connected module. Each of them has a residual connection and is followed by a layer normalization. The decoder is very similar to the encoder, however each layer now has an additional self-attention module that also operates over the output of the encoder (the number of layers of the decoder is the same as the encoder). This new module also has a residual connection and is followed by a normalization layer as well. The output of the decoder is then fed to a linear layer and a softmax layer for prediction.

2.3.2 Input Embedding

The first step of both the encoder and decoder networks is the input embedding layer. This is a learnable layer that is trained to convert the input tokens to a vector representation in a way that preserves the semantics and relations between tokens. The dimension of this vector will determine the dimension of the model, therefore is defined as d_{model}. Considering that the transformer architecture relies solely on attention instead of convolution or recurrence, in order for the model to exploit the order of the input sequence, we need to add some kind of positional information to the token embeddings. For each embedding vector, a new vector of the same dimension (d_{model}) that contains the respective positional information is created and added to it. These vectors are created using the cosine function for all odd vector indexes and the sine function for all even indexes as shown in equations 2.8, where pos is the position of the token in the sequence and i is the index of the positional encoding vector. These functions were chosen due to the author's hypothesis that the linear properties they possess would allow the model to easily learn to attend

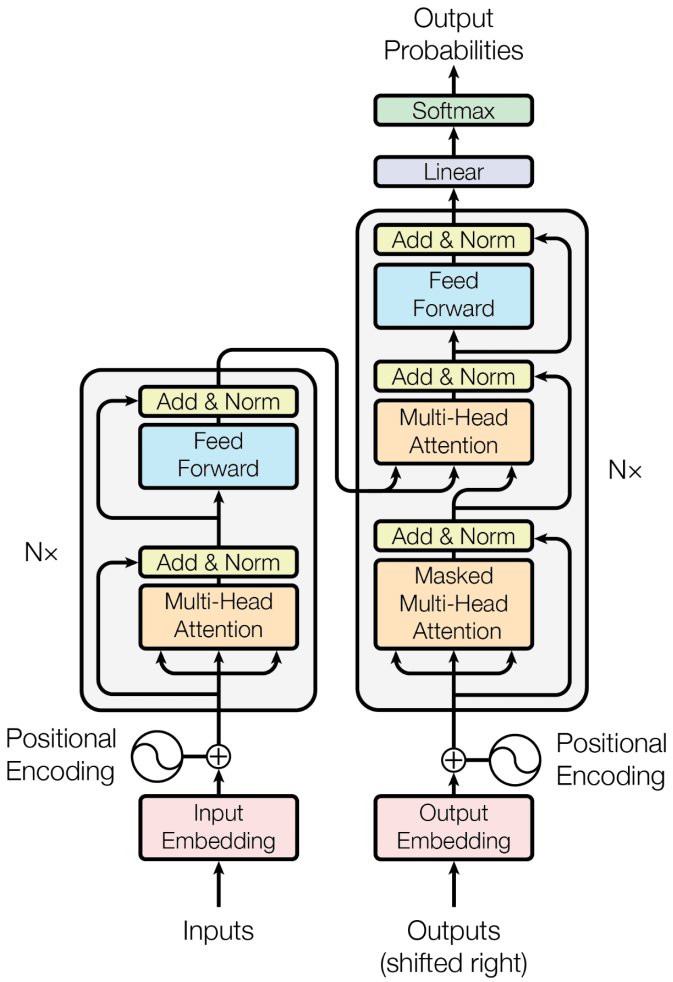


Figure 7: Architecture of the Transformer (Vaswani et al., 2023)

by relative positions.

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$
$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.8}$$

2.3.3 Attention

An attention function is essentially a map of a query and a set of key-value pairs to an output. Intuitively we can think of attention as being similar to retrieval systems were we cross the query with the set of keys and assign a value for each key that increases with the compatibility between the query and said key. From here we select the values which are associated to the most compatible keys.

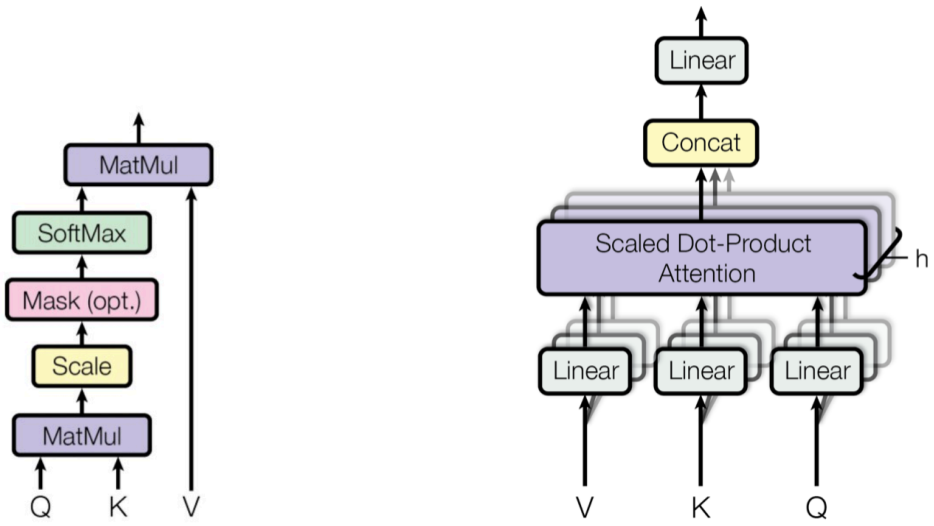
More formally speaking, the system proposed by the authors is named "Scaled Dot-Product Attention". This attention mechanism relies on three vectors all computed as a result of a linear transformation of the input: query, key and value (q, k, v). Having these vectors allows us to compute the attention weighting by calculating the scaled dot-product of q and k and then extract features with high attention by multiplying with v. Generalizing to matrix product where $Q \in \mathbb{R}^{seq_len \times d_k}$,$K \in \mathbb{R}^{seq_len \times d_k}$ and $V \in \mathbb{R}^{seq_len \times d_v}$ (where seq_len is the number of input tokens and $d_k = d_v = d_{model}$) are the matrices with all q,k, and v vectors, we calculate the attention (figure 8a) as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.9}$$

According to (Vaswani et al., 2023), the scaling factor $\sqrt{d_k}$ is required because for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients. The Q,K,V matrices are obtained by projecting the input matrix $X \in \mathbb{R}^{seq_len \times d_{model}}$ using three different weight matrices of dimension $d_{model} \times d_{model}$.

2.3.4 Multi-Head Attention

By projecting the tuple (Q, K, V) h times with different learned linear projections (W_i^Q, W_i^K, W_i^V), downsizing the Q,K and V matrices from dimension $d_k = d_{model}$ to $d_k = d_{model}/h$ and computing attention for each of them, we obtain h different attention heads. The Multi-head attention block is then complete by concatenating all the outputs of the different heads and projecting them with a linear layer (equation 2.10 and figure 8b):



(a) Scaled Dot-Product Attention

(b) Multi-Head Attention

Figure 8: Attention mechanism (Vaswani et al., 2023)

$$\begin{aligned} MultiHead(Q,K,V) &= Concat(head_1,...,head_h)W^O \\ \text{with } head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.10}$$

Where $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ are learnable parameter matrices.

In theory, the model should train each head to attend to information from different representation subspaces at different positions, giving the model more representation power. Due to the reduction of dimensionality in each head, computing multi-head attention is as fast as computing standard attention.

2.3.5 Applications of Attention

In the Transformer architecture, multi-head attention is used in three different ways. The first use is the encoder "self-attention" in which the queries, keys and values come from the output of the previous attention layer, allowing for each position in the encoder to attend to all positions in the previous layer. The second use is the decoder "self-attention" that is very similar to the encoder self-attention, however uses a mask in all the values in the input of the softmax that correspond to illegal connections (figure 8a) in order to preserve the auto-regressive property. The third use is the "encoder-decoder attention" in which the queries come from the previous decoder layer but the keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence.

2.3.6 Point-Wise Feed-Forward Network

Each of the encoder and decoder blocks contains a fully-connected feed-forward network module which is applied to each position separately and identically. Similarly to the multi-head attention module, this module is wrapped by a residual connection and followed by a layer normalization. This feed-forward network consists of two linear layers with a ReLU activation in between. While the linear transformations are the same across different positions, they use different parameters from layer to layer.

2.3.7 Advantages

The Transformer architecture proved to have so many advantages over the previous start-of-the-art architectures like LSTMs and CNNs that these quickly became obsolete for most of NLP related tasks. One of the biggest advantages is the use of attention mechanisms that are computed using mostly matrix multiplications, allowing us to parallelize the computations which was previously impossible due to the sequential nature of LSTMs. This means that we are able to scale the size of the model and keep training times low by using more hardware. This inherent parallelism also significantly speeds up training and inference compared to sequential models, leading to faster convergence and reduced training times.

A big problem of RNNs is the dificulty to capture long term dependencies since information from the previous steps in the hidden state tends to lose importance over longer sequences. This motivated the appearance of LSTMs and GRUs to somewhat mitigate this problem and allow for the use of slightly longer sequences. However, in theory, Transformers are able to capture long-range dependencies independently of the size of the sequence, provided with enough computing power. This enables the Transformer to better understand and generate coherent text, making it well-suited for tasks such as language modeling, machine translation and text summarization.

Transformers also offer greater interpretability, as attention weights provide insights at which parts of the input are relevant to generate the output token, allowing us to better debug and understand the model behavior. This is extremely valuable, specially in cases where transparency and explainability are important, such as healthcare or legal text analysis.

2.4 Vision Transformer (ViT)

Due to Transformers computational efficiency and scalability, it has become possible to train models with far more parameters than ever. Inspired by the Transformers scaling success in NLP, there were multiple

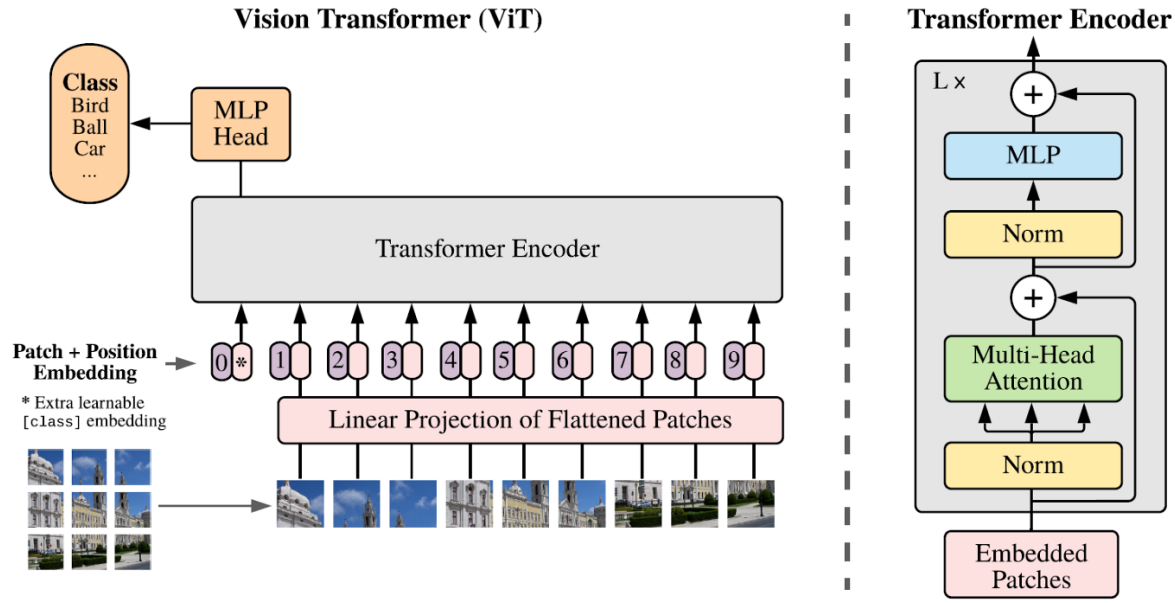


Figure 9: Architecture of the Vision Transformer (Dosovitskiy et al., 2021)

attempts to use this model in computer vision tasks, with the most successful one resulting in the Vision Transformer (Dosovitskiy et al., 2021).

The ViT uses an unaltered Transformer encoder to extract relevant feature vectors from images and use them as input to a head for the task required (for example, classification). However, in order to use this architecture in images, we need to convert them to a sequence of linear embeddings of "tokens". To do so, the authors propose to split the image in patches, treating them similarly to words in an NLP application. Figure 9 depicts an overview of the Vision Transformer architecture.

2.4.1 Patch Embeddings

The standard Transformer takes as input a 1D sequence of token embeddings. To match this input, the image $x \in \mathbb{R}^{H \times W \times C}$ is converted to a sequence of flattened 2D patches $x_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$ where (H, W, C) is the dimensions of the original image, (P, P) is the dimension of each patch, and $N = HW/P^2$ is the number of patches. The Transformer also uses a constant latent vector size D through all of its layers, therefore the flattened patches are then projected to D dimensions using a trainable linear projection, resulting in the final patch embeddings.

2.4.2 Class Token

Every input sequence is preceded by a learnable embedding (following BERTs strategy with the [*class*] token (Devlin et al., 2019)), whose state at the output of the encoder will be used as the representation of

the input image and fed to a classification head implemented by a one hidden layer MLP during pre-training.

2.4.3 Positional Encoding

Similarly to the original Transformer architecture, we add a positional encoding to the embeddings in order for the model to retain information about the tokens relative position to each other. However, in this case, instead of using a combination of sine and cosine functions, the Vision Transformer uses standard learnable 1D position embeddings. Considering that the input is now an image, learnable 2D-aware position embeddings should perform better in order to preserve information along both height and width dimensions, however significant gains were not observed, leading to the authors opting for the simplest solution.

2.4.4 Advantages

The Vision Transformers brought to the table several advantages over the previous state-of-the-art CNN architecture. As it was the case with standard Transformers and LSTMs in NLP, ViTs uses attention mechanisms to capture global context and relationships between distant image regions, enabling better understanding of spatial dependencies compared to CNNs, which are inherently limited to local receptive fields. Even with these advantages, in (Dosovitskiy et al., 2021) it is shown that ViTs performance still falls a bit short of CNNs when trained on mid-sized datasets, however, ViTs take the lead achieving state-of-the-art results when considering pre-training in larger datasets (14M to 300M images) and using transfer learning for tasks with fewer datapoints. Considering the recent tendencies of the world of deep learning where scaling in parameter count and dataset sizes is the name of the game, Vision Transformers emerge as the preferred architecture in computer vision.

Chapter 3

Self-Supervised Learning

Self-Supervised Learning (SSL) is a relatively new machine learning paradigm that was created due to the ambiguity of classifying certain learning frameworks as supervised or unsupervised learning, as they are closely related to both.

The relation between SSL and unsupervised learning comes from the fact that neither use labels in the training process and must learn intrinsic patterns and correlations instead of learning correlations between data and labels from annotated datasets. Due to this focus on unlabeled data, one might consider SSL as a subset of unsupervised learning. However, this is where the similarities between SSL and unsupervised learning end and the similarities with supervised learning start.

Although SSL does not use explicit labels from annotations (often referred to ground truth), it does measure its results against a ground truth that is implicitly derived from unlabled training data (supervisory signals), similarly to supervised learning. It also uses loss functions to calculate the divergence between model predictions and ground truth and gradient descent during backpropagation to tweak the model parameters in a way that minimizes loss.

This means that SSL frameworks are useful for tasks usually solved using supervised learning with the advantage of not needing a large annotated dataset, which is very costly, specially in two of the most prominent domains of problems where a large amount of data is required: Computer Vision and NLP.

However these advantages do not come for free. The task learned by the model needs to be framed in a way that allows the model to extract implicit labels in order to compute the loss function required to actually train the model, which means that the task itself might not be of direct interest. These tasks (usually denoted as pretext tasks or representation learning) instead teach the model to learn representations of the data that are very useful for downstream tasks that we want the model to perform. This means that models are pretrained using SSL to learn representations of the data from a large general dataset and then fine-tuned using supervised learning with a few labeled examples in the intended downstream task. Note that the domain of the downstream task should be closely related to the domain of the pretext task (for

example, pretraining using SSL on ImageNet (Deng et al., 2009) and then fine-tune in a more restrictive task such as fashion-MNIST (Xiao et al., 2017)). There are cases where the pretext task is more useful than just than just teach the model high quality representations of the data. An prominent example of this is next token prediction used to train Large Language Models (LLM) that when used recursively by appending the predicted token to the input and predicting the next token based on the original input and past predictions, allows the model to generate coherent texts.

3.1 Types of self-supervised learning methods

3.1.1 Autoassociative or Self-predictive Learning

Self-predictive learning is a category of SSL where the model is trained to learn important features from the data in order to reconstruct its own input or a portion of it.

Autoencoders An autoencoder encodes its input to a lower dimensional space (latent space) representation and then reconstructs (decodes) the input using that representation. The model uses the reconstruction error (difference between reconstructed input and the original input) as loss and is trained by minimizing this value.

Due to the reduced dimensionality of the latent space when compared to the input feature space, the model is forced to learn to encode the most important features from the input that allow for reconstruction.

Autoregression Autoregression models time-series data and use the values of previous timesteps to predict the value of the following time-step. In conventional regressions, the independent variables are used to predict a dependent variable. In autoregression, the regression is performed on the variable itself, by considering as independent variables the values of previous timesteps of the dependent variable (lags).

Masking This method involves masking some parts of the input and tasking the model with predicting or reconstruction those missing parts. Similarly to autoencoders, the reconstruction error obtained by measuring the reconstructed version against the input is used as loss to train the model.

3.1.2 Joint embedding contrastive SSL

Using a certain data sample as the target or anchor, training data can be divided in positive examples (other samples that match) and negative examples (do not match). In supervised learning, the positive

and negative examples are defined by having or not the same label as the target. Usually for SSL, for a batch of unlabled data, we apply a random combination of transformations (data augmentation) to each sample producing positive pairs. By assuming every pair of samples in the batch not produced in this way to not match, we have the negative pairs. Each of the samples and augmented versions are encoded into a vector representation and a contrastive loss function that minimizes agreement between negative representations and maximizes agreement between positive representations is used to train the model. These representations are robust to variations (due to augmentation) and very useful for downstream tasks.

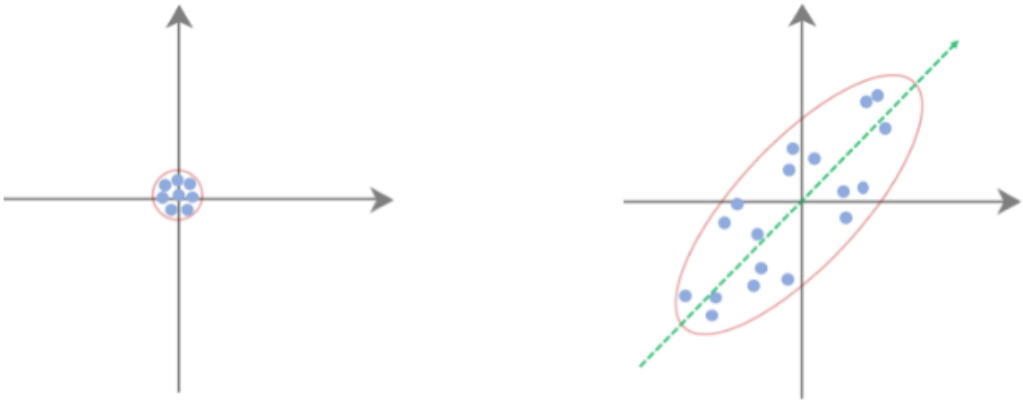
3.1.3 Joint embedding non-contrastive SSL

Non-contrastive SSL is very similar to contrastive learning. In this case, models are only trained using positive pairs and learn to minimize the difference between their vector representations, discarding the contrastive factor of negative pairs and using a standard loss function instead of a specialized contrastive function.

3.2 Collapse

When considering the use of SSL for computer vision, these methods learn useful representation of images for downstream tasks by minimizing some distance between two embedding vectors. These vectors are obtained by feeding the input image and an augmented version of itself to an encoder (there are cases where there are two encoders, one for each image and cases where both the images are augmented versions of the input).

However, this problem has a trivial solution: the encoder maps all the input images to a single vector in the representation space (Figure 10a). This means that for every input, both embeddings would be the exact same and therefore the distance would be minimized (loss function would always be 0). This is called complete collapse. This problem can be addressed by several methods, with contrastive methods being the most simple. By adding a new explicit term to the loss function that penalizes agreement between embedding vectors of negative pairs (contrastive loss), the loss function will have high values when the model approaches this trivial solution and it will encourage the model to push the representation vectors away from each other in the embedding space. In non-contrastive methods where this explicit loss term is not present, we need to find other ways to encourage separation in the embedding space. In the framework BYOL (Grill et al., 2020), for example, complete collapse is avoided by adding a projector to one of the



(a) Complete collapse on a 2D embedding space

(b) Dimensional collapse on a 2D embedding space

Figure 10: Patterns of collapse (Hua et al., 2021)

encoders (online network) and by the fact that the other encoder (target network) is not updated by gradient descent, thus avoiding convergence to the loss function trivial minimum.

There is also another collapse pattern that must be considered when dealing with SSL methods, both contrastive and non-contrastive, which is dimensional collapse. In this case, instead of all embedding vectors being the same, these vectors will not make use of all the dimensions from the embedding space and instead occupy a lower-dimensional subspace (Jing et al., 2022) (Figure 10b).

In contrast to complete collapse, dimensional collapse is not inherently 'bad'. If the number of dimensions used by the embeddings is not that much lower than the number of dimensions of the embedding space, it just means that the model did not need to use the whole space and managed to compress the representations in a smaller subspace without loss of information. As such, even though its common to use techniques to reduce this effect, it can still be found in state-of-the-art models.

3.3 SimCLR: Contrastive SSL framework

One of the SSL frameworks used in this work is SimCLR: "Simple contrastive learning of visual representations" (Chen et al., 2020a). Although not exactly a state-of-the-art method at the time of writing, this contrastive framework still stands its ground as a benchmark for all SSL methods for computer vision, which is the purpose for its use here.

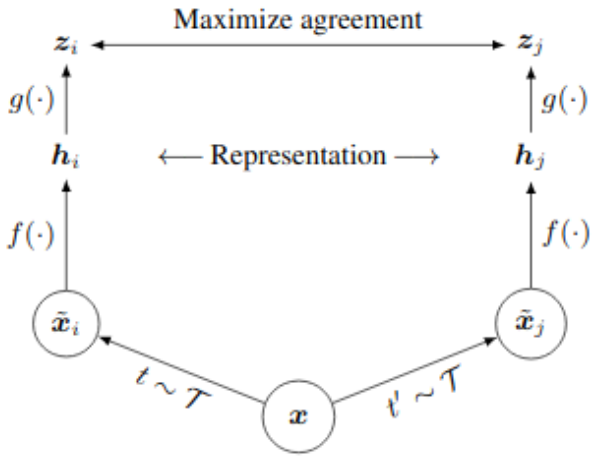


Figure 11: SimCLR architecture (Chen et al., 2020a)

3.3.1 Method

The framework structure is very similar to other contrastive SSL methods with a few key differences that allowed it to reach state-of-the-art results at the time. An overall view of the framework is illustrated in figure 11.

First, for each example image x we sample two compositions of augmentations t and t' from the same family of augmentations $\mathcal{T}$ and apply them to x, generating two augmented views of x, $\tilde{x}_i = t(x)$ and $\tilde{x}_j = t'(x)$. Then these two views are encoded by a neural network encoder $f(.)$ (backbone) that extracts the representation vectors $h_i = f(\tilde{x}_i)$ and $h_j = f(\tilde{x}_j)$. The backbone network $f(.)$ is the one that will be used to extract features for a classifier of a downstream task. The last steps of the training iteration are projecting the representation vectors to the space where the loss function will be calculated using a projection head $g(.)$, generating $z_i = g(h_i)$ and $z_j = g(h_j)$ and calculating the loss values for updating the network.

3.3.2 Data Augmentation

One of the main parts of this framework that made it stand out from other previous works is the emphasis on data augmentation. The authors of the paper consider data augmentation to be sufficient to define the predictive tasks. Therefore, a large number of data augmentation combinations were tested, including spatial/geometric transformations (random cropping and resizing, horizontal flip, random rotation, cutout) and appearance transformations (color dropping, brightness, contrast, saturation, hue, jitter, Gaussian blur, Sobel filtering). The best results were obtained by composing random cropping and resize back to the original size, random color distortions, and random Gaussian blur which were used in the default

configuration of SimCLR. From the tests, the combination of random cropping and color distortion was essential to achieve a good performance.

3.3.3 Backbone

The architecture of the backbone network $f(.)$ is not given much consideration as the framework is supposed to support several choices of architecture without constraints. In order to simplify things, the most common and state-of-the-art architecture for computer vision at the time was used: ResNet50. However, the few tests that were performed revealed that the gap between models trained using SimCLR and using supervised learning shrinks as the model size increases.

3.3.4 Projector

The importance of the projector is also studied using linear evaluations. Three variants of the architecture of the projector head were considered: identity mapping (equivalent to no projector head), linear projection and nonlinear projection with one hidden layer. The results showed that the nonlinear projection was the better option, with an advantage of 3 percentage points over the linear projection and an advantage of 10 percentage points over no projection head. The dimension of the output of the projection head was also tested ($32, 64, 128, 256, 1024, 2048$) and produced the same results. The main conclusion from these tests is that the representation vectors h are better representations ($> 10\%$) than the projection vectors $z = g(h)$.

3.3.5 Loss

For this framework, three different contrastive loss functions were tested: NT-Logistic loss (Mikolov et al., 2013a), Margin Triplet loss (Schroff et al., 2015) and NT-Xent loss (Sohn, 2016)(Wu et al., 2018)(van den Oord et al., 2018). From these three, NT-Xent was the best performing loss function and as such, was the chosen for this framework.

In this framework, after sampling a random batch of N images, the contrastive prediction task is defined by generating positive pairs using random augmentations, resulting in $2N$ datapoints that are converted in $2N$ projection vectors. Let (z_i, z_j) denote one of the positive pairs. Instead of sampling negative pairs explicitly, we consider the remaining $2(N-1)$ examples to be negative pairs for both z_i and z_j. This assumption is valid because when pre-training using large datasets with many classes (ImageNet with 1000 classes for example), the expected number of pairs derived from images of the

same class in the same batch is very low.

Now with the contrastive task defined, let $sim(u, v) = u^\top v/\|u\|\|v\|$ denote the cosine similarity between u and v. Then, for a positive pair (z_i, z_j), the NT-Xent loss (normalized temperature-scaled cross entropy loss) is defined as:

$$l_{i,j} = -log\frac{exp(sim(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k\neq i]} exp(sim(z_i, z_k)/\tau)} \tag{3.1}$$

where $\mathbb{1}_{[k\neq 1]} \in \{0, 1\}$ is an indicator function evaluating to 1 *iff* $k \neq i$ and τ is the temperature parameter. Finally, batch loss is calculated by the sum of the loss for each positive pairs (z_i, z_j) and (z_j, z_i):

$$L = \frac{1}{2N}\sum_{k=1}^{N}(l_{2k-1,2k} + l_{2k,2k-1}) \tag{3.2}$$

Intuitively, NT-Xent loss is applying multi-class cross-entropy loss to the *softmax* of the similarity of each positive pair over the similarity of all pairs in the batch. The temperature parameter τ controls the sharpness of the *softmax* distribution, sharpening the distribution when $\tau < 1$, smoothing the distribution when $\tau > 1$ and not affecting the distribution when $\tau = 1$.

3.3.6 Batch size

The diversity of negative examples is crucial for the model to identify what features are important to separate one image from the rest and produce richer representations for downstream tasks. Contrastive learning methods tends to benefit from a large number of negative examples due to this fact, facilitating convergence. In SimCLR, considering that the number of negative examples is directly related to the batch size, this model benefits a lot from larger batch sizes. This benefit can be obtained by training the model for more epochs, assuming a random resampling of the batches.

3.3.7 Avoiding collapse

Simply by being a contrastive method, SimCLR avoids complete collapse due to the negative examples pushing representations from different input images apart in the representation space. Although dimensional collapse was not a consideration in this framework design, the addition of a projection head contributed for a considerable reduction of this collapse pattern. The tests proved that the representations generated using a nonlinear projection head were better for classification and in (Jing et al., 2022) it was

proved that this improvement derived from a large reduction of collapsed dimensions in the representation space generated by the model.

3.4 DINO: Non-contrastive SSL framework

Most SSL methods for computer vision share a very similar structure across the board: from an input example x, generate two augmented views x_1 and x_2, using some kind of encoder $g(.)$ obtain representation vectors from these views $g(x_1)$ and $g(x_2)$, and finally compute loss using these representations. The variations that are introduced by each method are novel ways of producing the augmented views, different encoder designs or different losses that formulate different pretext tasks.

In this section we will be discussing DINO (Caron et al., 2021) (knowledge DIstillation with NO labels), which follows a very different approach to the problem, although in practice results in a similar structure.

3.4.1 Method

The strategy for DINO is to formulate the pretext task as a knowledge distillation problem with no labels. Knowledge distillation is mainly used to transfer the knowledge of a large pretrained model, denoted as the teacher network, to a smaller untrained network, denoted as the student, by training the student model to predict the teacher's output. The result is a compressed version of the large pretrained model.

To formulate this problem, in DINO, we have two networks with the same architecture g, the teacher g_{θ_t} with parameters θ_t and the student g_{θ_s} with parameters θ_s. Given two augmented versions of an input image x, x_1 and x_2 (in practice x_1 and x_2 are sets of augmented views but for simplicity we will consider it as only one view each and discuss it further later), the output probability distribution of both networks P_t and P_s is obtained by normalizing the output of g with a *softmax* function (equations 3.3). Similar to SimCLR, parameters τ_t and τ_s are temperature parameters that control the sharpness of the *softmax* function for the teacher and student networks respectively. Then, we train the student to match the teacher's probability distribution using the cross-entropy loss function. An overall view of this framework is illustrated in figure 12.

$$
\begin{aligned}
P_s(x)^{(i)} &= \frac{exp(g_{\theta_s}(x)^{(i)}/\tau_s)}{\sum_{k=1}^{K} exp(g_{\theta_s}(x)^{(k)}/\tau_s)} \\
P_t(x)^{(i)} &= \frac{exp(g_{\theta_t}(x)^{(i)}/\tau_t)}{\sum_{k=1}^{K} exp(g_{\theta_t}(x)^{(k)}/\tau_t)}
\end{aligned}
\tag{3.3}
$$

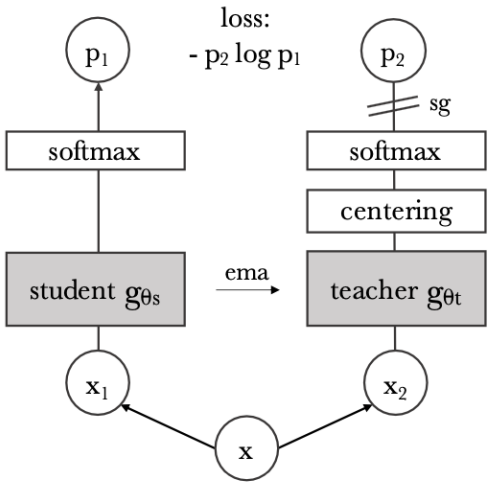


Figure 12: DINO architecture (Caron et al., 2021)

3.4.2 Data augmentation

When it comes to data augmentation, DINO takes inspiration from SwAV (Caron et al., 2020) and uses a multi-crop strategy. For each input image x, we generate a set V of different views containing two global views x_1^g and x_2^g (cover more than 50% of x) and several local views (cover less than 50% of x) of smaller resolution. Each of these views is distorted using a composition of augmentations inspired by BYOL (Grill et al., 2020) that includes a random horizontal flip, random color jitter, Gaussian blur and for one of the global views, solarization. During training, the global views are passed through the teacher network while the local views are passed through the student network, encouraging the student to find "local-to-global" correspondence.

3.4.3 Teacher and Student Networks

Like SimCLR, the encoder g is composed by the backbone f and a projection head h with $g = h \circ f$. The features that are used for downstream tasks are the outputs of f. This framework also allows for variety in network architecture choices for the backbone network. However, it was designed with focus on ViT networks, which proved to achieve better performance when compared with ResNets trained using DINO. Following SwAV (Caron et al., 2020) again, the projector is a 3-layer MLP with hidden dimensions of 2048 with l_2 normalization and a weight normalization layer (Salimans and Kingma, 2016).

One fundamental difference between DINO and normal knowledge distillation is that instead of a pretrained teacher network, DINO builds the teacher network during training using an exponential moving

average (EMA) of the student network (this is possible since both networks architectures are exactly the same, only changing the weights), making it a momentum encoder similar to what is used in other SSL frameworks derived from MoCo (He et al., 2019), with the role of mean teacher (Tarvainen and Valpola, 2017) instead of substitute for a queue in contrastive learning .

3.4.4 Teacher and Student weight updates

The student network is updated using a stochastic gradient descent (by default using the optimizer AdamW (Loshchilov and Hutter, 2019)). Given the probability distributions of the teacher and student networks $P_t(x)$ and $P_s(s)$, respectively of an input x (defined in equations 3.3), we train the student network by minimizing the cross-entropy loss, w.r.t the parameters of the student network θ_s as defined in 3.4, where $H(a, b) = -a \cdot log(b)$

$$\min_{\theta_s} \quad H(P_t(x), P_s(x)) \tag{3.4}$$

When considering that x is a set of crops V with two global crops that pass through the teacher network and several local crops that pass through the student network, the minimization problem defined in equation 3.4 is now defined as equation 3.5.

$$\min_{\theta_s} \sum_{x \in \{x_1^g, x_2^g\}} \sum_{\substack{x' \in V \\ x' \neq x}} H(P_t(x), P_s(x')) \tag{3.5}$$

The teacher is updated following an EMA of the student network, with the update rule defined in 3.6, where λ (will be referred as *teacher_momentum* moving forward) is an hyperparameter that controls which fraction of information from past iterations of the student network is kept. This hyperparameter follows a cosine schedule during training from 0.996 to 1 (Section 5.1).

$$\theta_t \longleftarrow \lambda\theta_t + (1 - \lambda)\theta_s \tag{3.6}$$

3.4.5 Avoiding collapse

Since DINO does not use a contrastive loss function like SimCLR, we need other ways to prevent the complete collapse of the model representation. To avoid complete collapse, DINO applies batch centering to the output of the teacher network, which can be interpreted as adding a term c to the output of the network on each training iteration: $g_{\theta_t}(x) \longleftarrow g_{\theta_t}(x) + c$. The center term c (vector with the same dimension as the output of the teacher network) is initialized as the null vector and updated with an EMA

of the mean output of the current batch, as shown in equation 3.7, where B is the batch size and $m > 0$ is the parameter that controls the EMA update rate (will be referred as center momentum moving forward).

$$c \longleftarrow mc + (1 - m)\frac{1}{B}\sum_{i=1}^{B} g_{\theta_t}(x_i) \tag{3.7}$$

This operation however, despite preventing one dimension to dominate the others (complete collapse), it is shown to encourage the collapse of the representation to the uniform distribution. In order to counter this undesirable effect of batch centering, we sharpen the distribution of the teacher network by using a low value for the temperature parameter of the scaled *softmax* function τ_t. This sharpening has the complete opposite effect of centering and encourages the model reprentation to converge to a complete collapse. By applying both operations we can balance their effects and safely avoid the collapse of the network.

Chapter 4

Semi-Supervised Learning

Machine learning methods, more specifically deep learning, depend on large amounts of data and the most common tasks where deep learning is used (such as classification) usually depend on supervision to be solved, requiring the labeling of the data that is used. However in many practical applications, the acquisition of labeled data is not viable due to the long annotation process that is very costly and/or requires experts to label data (medical applications for example). Semi-supervised learning alleviates the dependence on labeled data by leveraging knowledge from both labeled and unlabeled data. The goal is to use unlabeled data to extract knowledge of the high-level structure of the data distribution and labeled data to learn the fine-grained details of the task in question (Murphy, 2022). In some tasks, the lack of labeled data requires the use of unsupervised learning, but in tasks where a supervised solution already exists, semi-supervision can improve the performance by adding structural knowledge provided by cheap unlabeled data.

4.1 Types of Semi-supervised learning methods

There is a plethora of types, however most of them are not relevant for this dissertation. Therefore we will give a brief overview of some of them, while going more in depth in the group of methods that will be used on the experiments.

4.1.1 Deep generative methods

Deep generative methods use Generative Adversarial Networks (GANs) and Variational Auto-Encoders (VAEs), used mostly for unsupervised learning, and adapt them to accommodate labeled data (for details on these architectures, refer to (Murphy, 2022)). These methods have emerged to explore the distribution of the training dataset and generate new examples.

GAN based methods: To use GAN's in a semi-supervised setting, we combine the unsupervised GAN value function with a supervised classification objective function (Yang et al., 2023). When receiving data from the generator, the critic is trained to output the standard *real/fake* binary classification. When receiving unlabeled data, it is trained to predict the class from the supervised task. When receiving labeled data, the critic behaves as a standard supervised model. The loss function has now 3 terms: The standard GAN loss (*real* vs. *fake* classification), a supervised classification loss for labeled real data and an unsupervised component encouraging confident predictions on unlabeled real data.

VAE based methods: VAE's are adapted to the semi-supervised setting by adding the label vector y_i which contains the one-hot encoded labels of labeled data points to the latent representation z and is used as an additional latent variable for unlabeled data (van Engelen and Hoos, 2020). The new loss function now contains 3 components: standard VAE reconstruction loss (how well the decoder reconstructs the input), KL divergence (to ensure the latent space is well-structured) and a supervised classification loss (for labeled data only).

4.1.2 Consistency regularization methods

Consistency regularization methods are based on the idea that introducing some kind of consistency constraints/perturbations to the loss function should not cause dramatic changes to the model's output. The reliance on only on model outputs make these methods inherently applicable to unlabeled data and as such can be used to create loss functions for semi-supervised learning (Murphy, 2022). These perturbations can be introduced on three levels: Input perturbations (such as data augmentation or noise), model weights perturbations (dropout or weight noise) and layer perturbations (Yang et al., 2023).

4.1.3 Graph-based methods

Graph-based methods are built on the assumption that points that are close according to some similarity measure should share the same label. Following this assumption a graph is constructed where the nodes are the data points and the weighted edges represent the degree of similarity of the connected nodes by that edge (Murphy, 2022). In traditional methods such as label propagation, the known labels for nodes representing labeled data points are then propagated through the graph to nodes representing unlabeled data points. In deep learning, node information is encoded in small vectors that represent their roles and information of the neighbourhood in the graph. These embeddings are then fed to neural networks such as auto-encoders or graph neural networks (GNN).

4.1.4 Pseudo-labelling methods

Pseudo-labelling methods use the model trained on the labeled data to infer predictions on unlabeled data and then treat these "pseudo-labels" as true labels for the next training steps. One of the key points of pseudo-labelling is the criteria used for selecting which pseudo-labels incorporate the training-set, which usually is setting a threshold in model confidence predictions. These methods have a common problem dubbed confirmation bias, where the model generates incorrect predictions in the first inference steps and is trained on them, becoming progressively worse over time (Murphy, 2022).

Co-training is an extension of this procedure to multiple supervised classifiers (the procedure with one classifier is called self-training). In co-training, each classifier adds their most confident pseudo-labels to the training set of the other classifiers and vice-versa. Multi-view co-training assumes the existence of multiple views of the data for each classifier , which generally are different subsets of features (van Engelen and Hoos, 2020).

4.2 Transfer Learning from Self-Supervised Models

Transfer learning is a widely used method that consists of two phases: pretrainig and fine-tuning. In the pretraining phase, the model, composed of a representation network and a head network, is trained on a the source task (usually a large general dataset) in order to train the representation network's weights. Then, on the fine-tuning phase, these weights are used to initialize another representation network with a different head that will be trained further on the target task (smaller and more specific dataset, somewhat related to the source task). The motivation behind this method is that the pretrained network will produce very useful representations of the target task data due to how closely related the tasks are and these representations will have very high quality due to the large amounts of data in the source task data.

Transfer learning from self-supervised models is a variant of transfer learning in which pretraining is done with self-supervised methods instead of using supervision. Since self-supervised learning does not use labels in the source task, the learned representation network is not biased to these labels and is presumably more generalizable for the target task. However, a potential downside is that the representation network is learned without human supervision, which renders the network less discriminative (Yang et al., 2020).

4.2.1 Advantages

In terms of performance, transfer learning from self-supervised models seems to keep up or even surpass standard transfer learning (Ericsson et al., 2021) while keeping all the advantages of being a semi-supervised learning method, such as being able to perform well in scenarios where data is scarce. When comparing with other semi-supervised methods, transfer learning from self-supervised models decouples the unlabeled learning phase from the supervised fine-tuning, enabling more efficient use of massive unlabeled datasets and better scaling properties instead of relying on careful balancing of labeled and unlabeled data. Also, while other semi-supervised methods usually assume similar distributions between labeled and unlabeled data, transfer learning from self-supervised models offers better robustness to distribution shifts, derived from learning fundamental visual or semantic features that generalize effectively across tasks of the same domain.

4.2.2 Challenges

One of the main challenges in transfer learning is catastrophic forgetting, a phenomenon that occurs specifically when the network is trained sequentially on multiple tasks, because the weights in the network that are important for the source task are changed to meet the objectives of target task (Kirkpatrick et al., 2017). This means that aggressive adaptation to the target task risks losing valuable pretrained representations and replace them with overfitted representations due to the low quantity of data. This challenge requires more careful attention in self-supervised transfer learning compared to supervised pretraining, as the pretrained representations often capture more nuanced and generalizable features. To overcome this problem we need to reach a balance between preserving general features and adapting to specific tasks. While the pretrained representations offer valuable general understanding, excessive preservation may hinder optimal performance on the target task. This balancing act becomes particularly important given the label-agnostic nature of self-supervised representations compared to those from simpler supervised approaches.

Domain shift between pretraining and target data introduces additional complications, though self-supervised models have an advantage by often showing better resilience to such shifts compared to other approaches. When significant domain differences exist, careful assessment and adaptation strategies become necessary. These might include gradual domain adaptation techniques or intermediate fine-tuning stages, leveraging the flexible and generalizable nature of self-supervised representations.

Another concern when using transfer learning from self-supervised models is the computational cost

of this approach. While self-supervised pretraining is costly in computational resources, its efficient use of unlabeled data and flexible transfer capabilities often make it more practical than alternatives requiring extensive labeled datasets.

4.2.3 Strategies

The process of fine-tuning requires careful consideration of various strategies, especially when considering self-supervised pretrained models. Full fine-tuning involves training all model parameters during the fine-tuning phase, offering maximum flexibility, but requiring significant computational resources. Some strategies to perform a more stable fine-tuning process, while not being mutually exclusive, include partial fine-tuning, layer-wise learning rates, and gradual unfreezing of layers.

Partial fine-tuning selectively updates specific layers while preserving others, making it particularly valuable when working with limited computational resources or when trying to maintain crucial pretrained features (Ye et al., 2023). The most extreme case of partial fine-tuning is when the representation network is frozen and only the head is trained on the target data (Song et al., 2022), relying on the capability of the representation network to produce high quality representations of the target data, significantly reducing computational costs, since the head is usually composed of small MLPs and sometimes only a linear layer.

Layer-wise learning rate adjustment provides a sophisticated approach to fine-tuning by employing different learning rates across the model's layers. Early layers, responsible for extracting lower-level and more generic features receive lower learning rates in order to preserve valuable pretrained representations. In contrast, later layers that usually extract higher-level and more task-dependent features receive higher learning rates to facilitate adaptation to specific tasks (Ro and Choi, 2021).

Gradual unfreezing techniques progressively allow different parts of the model to adapt. This method typically begins with task-specific layers and gradually incorporates earlier layers into the training process, helping maintain stability while allowing effective task adaptation (Howard and Ruder, 2018).

4.3 Fine-tuning self-supervised models with few examples

Few-shot learning is an area of machine learning that aims to train a model that is able to adapt to unseen tasks during training using minimal labeled data. Most established methods rely on supervised meta-learning (learn-to-learn) techniques that require large labeled datasets. While there have been attempts to adapt these methods to incorporate some kind of self-supervision such as data augmentation or pseudo-labels, they sadly fail to reach the performances of their supervised counterparts. However, recent works

are questioning whether self-supervised pretrained models can be used as few-shot learners (Chen et al., 2020b) (Lu et al., 2022).

Building upon the general representations of data from pretrained self-supervised models, this approach enables efficient learning from a low number of samples per class, usually one, five or ten examples (one-shot learning, five-shot learning and ten-shot learning respectively). The methodology focuses on preserving the valuable representations learned during pretraining while adapting the model to recognize new classes or perform new tasks with minimal supervision. This is a staple method in NLP where the model is pretrained using large sources of unlabeled text in order to learn the overall structure and semantics of a language, and then fine-tuned using few-shot learning for specific tasks.

4.3.1 Advantages

Few-shot fine-tuning with self-supervised models offers significant advantages over traditional fine-tuning approaches. By leveraging pretrained representations, models can adapt to new tasks with little data, making it feasible to deploy AI solutions to solve tasks where labeled data is scarce or expensive to obtain. This approach significantly reduces the time and resources required for model adaptation, enabling rapid deployment of models for specific applications. The method also benefits from the robust features learned during pretraining that achieve better generalization when compared to models trained from scratch on small datasets.

4.3.2 Challenges

The risk of overfitting is particularly relevant in few-shot scenarios with larger models, where millions of parameters can easily memorize the few available training examples rather than learning generalizable patterns. The lack of variability found within the examples of each class might lead the model to overfit on certain features that do not generalize well for other examples outside of the training data.

Usual validation approaches become less reliable due to extremely limited data. Meta-learning approaches, modified cross-validation techniques and ensemble methods combining multiple fine-tuning runs can be used to overcome this problem. Also, traditional validation splits may not be feasible, leading to the use of leave-one-out cross-validation or specialized few-shot validation approaches.

4.3.3 Strategies

The success of few-shot fine-tuning relies on several key technical considerations. The initialization phase requires careful attention to learning rates, using typically much lower values, to enable adaptation while preventing the degradation of useful pretrained features. Learning rate scheduling strategies, such as cosine scheduling with warm restarts or layer-wise learning rate decay help maintain this delicate balance throughout the fine-tuning process (Chen et al., 2020b). Regularization techniques such as higher dropout rates (between 0.5 and 0.7) and carefully tuned weight decay can also play a big role in preventing overfitting due to the low amount of examples.

Chapter 5

Technical concepts and metrics

In this chapter, we introduce some metrics and methods that will be necessary for the comprehension of the numerical experiments and subsequent evaluations.

5.1 Cosine annealing schedule

Cosine annealing schedule is a technique proposed in (Loshchilov and Hutter, 2017) for a learning rate scheduler with warm restarts. The learning rate will be scheduled to decay according to the cosine function within a set range of values and, after a certain number of iterations, the schedule will be restarted, resetting the value to the upper bound and restarting the decay. The learning rate parameter for each iteration (batch) t is defined as:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min}) \left(1 + cos\left(\frac{T_{cur}(t)}{T_i}\pi\right)\right) \tag{5.1}$$

where η_{min} and η_{max} are ranges for the learning rate, T_{cur} is the number of iterations since the last restart and T_i is the number of iterations between two warm restarts. When $T_{cur}(t) = Ti$, $\eta_t = \eta_{min}$ and when $T_{cur}(t) = 0$ after a restart, $\eta_t = \eta_{max}$.

In the experiments section, this schedule will be used for other parameters as well, such as weight decay and T_i will be set as the number of batches of the whole training section (batch size $\times$ epochs). As such, there will be no warm restarts of the scheduler and the parameters will strictly decrease (or increase, by setting η_{min} as the upper bound and η_{max} as the lower bound) during training following the cosine function (figure 13).

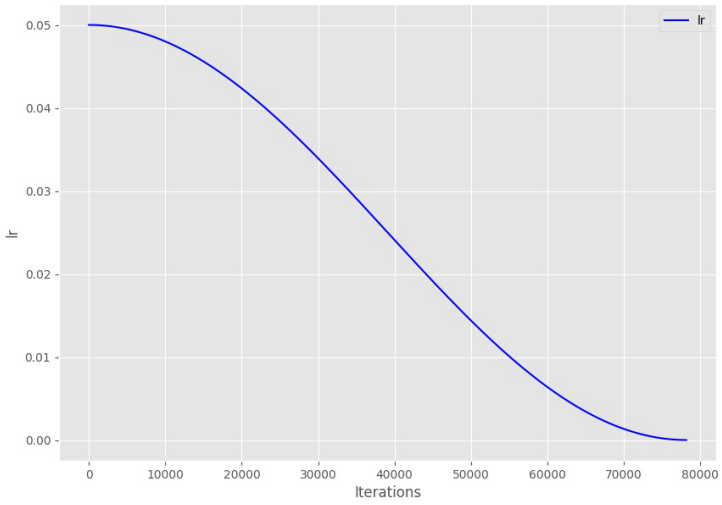


Figure 13: Example of learning rate during training using cosine annealing schedule without warm restarts (source: (Rath, 2021))

5.2 One epoch learning rate range test

The learning rate is the hyperparameter that controls how much the optimizer adjusts the weights of our network with respect to the loss gradient. It measures how much a model can "learn" from a new mini-batch of training data, meaning how much we update the model weights with information coming from each new mini-batch.

During training, too large of a learning rate leads the optimizer to take bigger steps towards the minimum of the loss function and potentially fail to reach it, causing a plateau of the loss values during training or an explosion in more extreme cases. Conversely, too small of a learning rate causes the optimizer to take smaller steps and therefore requiring way too many training iterations / updates to reach the minimum of the loss function (figures 14 and 15).

The learning rate range test is an useful technique to find suitable maximum and minimum values for the learning rate for a specific training session. In this process, we train the model during one epoch and the learning rate is increased slowly every mini-batch, starting in a very low value and ending in a very high value. Starting with small learning rates, the model will slowly converge until it eventually reaches a 'breaking point', where the learning rate becomes too high and loss values increases. This point is clearly an upper bound on the ideal learning rate. Therefore, as a rule of thumb, a value one magnitude lower than this maximum value (ten times lower) will be a safe choice as it is big enough for the model to converge and small enough to avoid instability during training (Smith, 2018) (Tanksale, 2018). In this

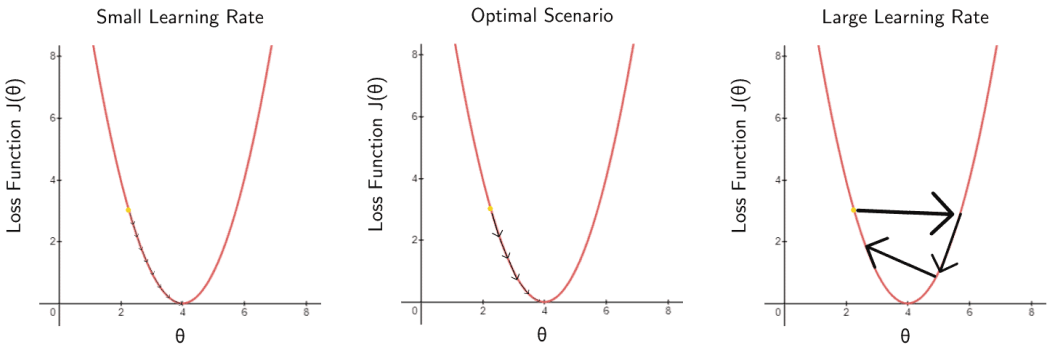


Figure 14: Comparing the effects of learning rate on optimizer steps during gradient descent (source: (Barreto))

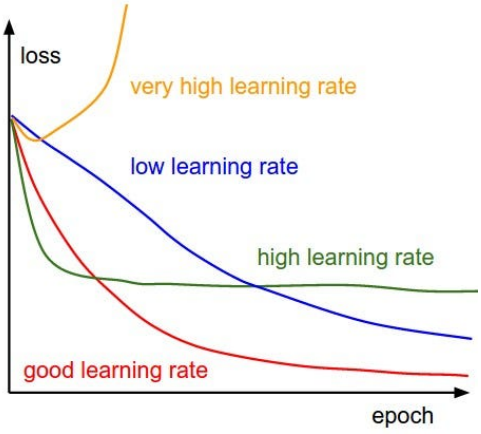


Figure 15: Comparing the effects of learning rate on loss values during training (source: (Rakhecha, 2019))

work, in order to scan learning rate ranges with more granularity, instead of a single run starting with very low learning rate values and ending with very high values, we will have several runs where each one tests one order of magnitude. For example, one run will test values from 5×10^{-6} to 5×10^{-5} while other run will test values from 5×10^{-5} to 5×10^{-4}, and so on.

5.3 Collapse Analysis

The job of a model in self-supervised learning is to map the inputs from the input space of dimension d_{input} to vector embeddings/representations of dimension $d_{embedding}$ (usually $d_{embedding} < d_{input}$). The vector space that all the possible vector embeddings inhabit is called the embedding/representation space. During self-supervised training, the pretext task is formulated in a way that encourages the model to learn a mapping such that embeddings from similar inputs are close in the embedding space, embeddings from dissimilar inputs are distant in the embedding space and that both distances and directions in the embedding space represent semantic information about the input domain. An embedding space with such characteristics would be the embedding space from Word2vec (Mikolov et al., 2013b) (maps words to high dimension embeddings), where the model captures both syntactic and semantic similarities between words. A famous example of these capabilities is that the difference vector between the embeddings of the words 'Queen' and 'King' in the embedding space encodes the semantic meaning of gender. As such, this vector is very similar to the difference vector between the embeddings of the words 'Woman' and 'Man' (Vector('Queen') - Vector('King') = Vector('Women') - Vector('Man')).

To verify if the model is not suffering from dimensional collapse (section 3.2) we can determine if the dimension of the embedding space is smaller than the dimension of the model embedding vectors. The dimension of the embedding space is the rank of the matrix with all the possible embeddings that the model can generate. This matrix is impossible to obtain since it would require us to feed the model with all the possible combinations of inputs. However we can estimate the dimension of the embedding space by calculating the rank of the matrix containing a small set of embeddings. As such, in this work, we will be calculating the dimension of the subspace occupied by the embeddings of a small validation set.

In (Garrido et al., 2023), the rank of the embedding matrix Z is estimated using the number of non-zero singular values of Z (singular spectrum analysis). However, we will be following a similar approach (Jing et al., 2022). The dimension of the embedding space will be evaluated by collecting the embeddings of the validation set in a matrix V. Then we compute the singular values of the covariance matrix, $cov(V)$, plotting these values in sorted order and in logarithmic scale. The number of singular values

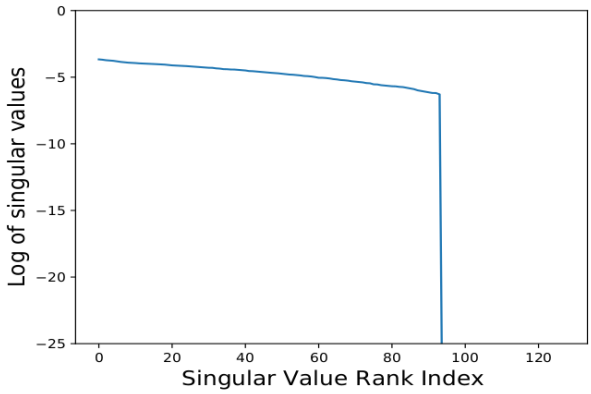


Figure 16: Example of collapse analysis of an embedding space (source: (Jing et al., 2022))

that drop to zero will represent the collapsed dimensions of the embeddings. In the example in figure 16, the embedding vectors have dimension 128 but they only inhabit a subspace with a smaller dimension (between 90 and 100).

5.4 K-nearest neighbours classifier

The k-nearest neighbours (KNN) classifier is one of the simplest supervised classification algorithms. The main idea behind this algorithm is to classify a new input based on the labels of the nearest samples of the training set, working off the assumption that similar points can be found near one another. The two main hyperparameters are the number of neighbours used for classification K and a distance metric $d(x, x')$. Commonly used distance metrics include the euclidean distance 5.2 and the manhattan distance 5.3.

$$d_{euclidean}(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2} \tag{5.2}$$

$$d_{manhattan}(x, y) = \sqrt{\sum_{i=1}^{n} |y_i - x_i|} \tag{5.3}$$

Let $X = \{x_1, x_2, ..., x_n\}$ the training dataset and $Y = \{y_1, y_2, ..., y_n\}$ the corresponding labels where $y_i \in C = \{1, 2, ..., C\}$ represents the class label. For a certain query datapoint x_q, its predicted label y_q is obtained by calculating the distance between x_q and all the training datapoints, selecting the closest K samples to x_q (neighbourhood of x_q), $N_K \in X$ with $|N_K| = K$ and then computing the majority label of all the samples in the neighbourhood $x_i \in N_K$:

$$y_q = \underset{c \in \{1,2,\ldots,C\}}{\text{argmax}} \sum_{i=1}^{|N_K|} \mathbb{1}_{[y_i=c]} \tag{5.4}$$

where $\mathbb{1}_{[y_i=c]} \in \{0, 1\}$ is an indicator function evaluating to 1 *iff* $y_i = c$.

In this work, the parameters for the KNN classifier will be $K = 20$ and, since the inputs of the classifier will be d-dimensional representation vectors of images, the distance metric used will be a scaled dot product (with the scale $\tau = 0.07$, follwing (Caron et al., 2021)) of the two representations:

$$d(x, y) = (x \cdot y)/\tau \tag{5.5}$$

Also the majority voting will be replaced with weighted majority voting, where each vote will have a weight based on its distance to the query datapoint (datapoints that are closer have bigger weights):

$$y_q = \underset{c \in \{1,2,\ldots,C\}}{\text{argmax}} \sum_{i=1}^{|N_K|} \alpha_i \mathbb{1}_{[y_i=c]} \tag{5.6}$$

where $\alpha_i = \exp(d(x_i, x_q))$ is the weight contribution of the training sample x_i to the classification of the query datapoint x_q.

5.4.1 KNN retrieval

KNN retrieval follows the exact same logic as the KNN classifier. However, instead of computing the label of the query point based on the majority voting of the neighbours, it just returns the original inputs behind the embeddings of the neighbourhood N_K of the query point. This will allow us to inspect what datapoints from the training set are being clustered together in the embedding space. Since the original inputs are images, the hyperparameter K will be set to 5 so it becomes manageable to display. The metric used to calculate the distance between the embeddings is the same that was used in the KNN classifier (equation 5.5).

Chapter 6

Numerical Experiments

This chapter consists of the experiments carried out in this dissertation and analysis of the results obtained. The goal is to train a model for classification of car model and make in images, using as few labeled data as possible, in order to surpass the costly need of a labeled dataset. We will start by evaluating the performance of a pretrained model using DINO as the self-supervised training framework (Caron et al., 2021). Then we will move on for a "self-supervised fine-tuning" of said model using DINO and hyperparameter optimization of this training step. The last experiments using DINO will include fine-tuning of a partially pretrained model and also training a model from scratch using DINO. Lastly we will compare DINO's performance with another self-supervised framework, SimCLR (Chen et al., 2020a) and perform few-shot classification using model checkpoints from the experiments described above.

Thanks to the RNCA (Rede Nacional de Computação Avançada), we had access to four NVIDIA A100 GPUs belonging to the Vision Cluster for most of the development stage of the dissertation. All the experiments using DINO were executed in one NVIDIA A100 GPU allowing the execution of four different experiments at the same time.

As mentioned in (Caron et al., 2021), in the DINO framework, the teacher network consistently outperforms the student network during training, therefore all evaluations of this framework will focus on the teacher network and only using the student network for training.

6.1 Pretrained model (baseline)

Using a pretrained model allows us to bypass a lot of the disadvantages of using a large model like the transformer models. Training a transformer model from scratch requires large amounts of data, time and computational resources, making this process very difficult to achieve using low capacity hardware and the Stanford Cars dataset (Krause et al., 2013), which only contains 16 185 images for 196 classes, 8 144 for the training set and 8 041 for the test set. Some examples of images contained in the dataset can be

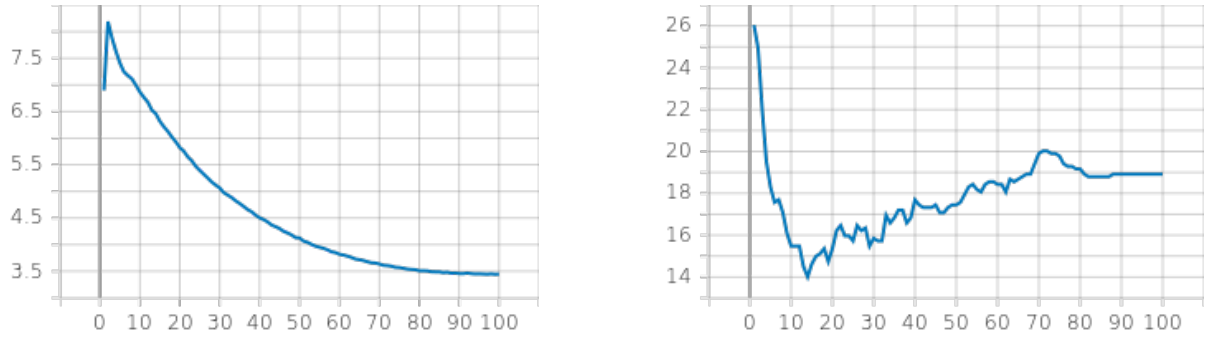
visualized in section 6.4.2. Pretrained models mitigate this issue by exploiting the knowledge from large datasets like ImageNet, which contains 14 197 122 annotated images for 1000 classes, and cut down most of the time and resources needed for training, as pretrained models on generic datasets usually learn features that are highly relevant for a wide range of tasks, which helps them generalize well even with limited task-specific data. Moreover, these general features learned by the model can be a strong starting point for training, only requiring a fine-tuning to adapt to the new tasks. However, this would not be a viable solution as fine-tuning is usually done by training the pretrained model on the target dataset in a supervised way, defeating our purpose of finding a way to solve this problem without the need of annotated data. To solve this problem, we will explore the idea of a "self-supervised fine-tuning" where we start with a model pretrained using a self-supervised framework in a large dataset and then train the model using the same framework on our target dataset. We will evaluate whether or not we can achieve similar performance using this method instead of a supervised fine-tuning.

To test the viability of this process, we decided to start by assessing how well a ViT network pretrained on the ImageNet dataset would perform on a validation split of the Stanford Cars dataset (the train split of the dataset will be further divided into train and validation sets, with a $90\%/10\%$ split) for our baseline. We chose a ViT-S16 network (12 transformer blocks, embbeding dimension of 384, patch size of 16×16) pretrained on the ImageNet dataset using the DINO framework, which in the original paper achieves 93.0% accuracy on the Stanford Cars dataset after a supervised fine-tuning, and used a KNN classifier, trained on a 90% split of the training set and evaluated on a 10% validation split of the training set, achieving a top-1 accuracy of 26.54% and a top-5 accuracy of 50.86%.

Although this performance is not as high as we would've hoped for, it is still a stronger baseline than a fresh untrained model and will hopefully save us a lot of time when training the model for our target dataset.

6.2 Fine-tuning using DINO

Following the rather underachieving raw model evaluation, the next step would be to fine-tune our model using DINO to fit the Stanford Cars dataset instead of ImageNet. Our hope is that the model can retain some general features from pretraining like edge and contour detection and now learn new useful and more specific features that help separate the model and make of the cars in the images of the new dataset.



(a) Training loss per epoch

(b) Validation accuracy per epoch

Figure 17: Results from training with default parameters for 100 epochs

6.2.1 Default parameters

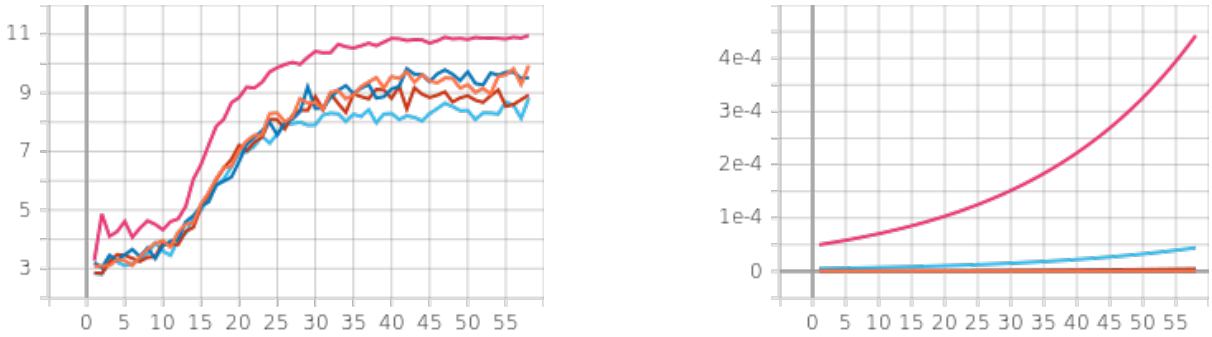
The first step would be how far we could go by training the model using DINO with the default parameters. We ran a training session of 100 epochs while monitoring the training loss as well as the validation accuracy to have a first impression of training performance before hyperparameter tuning.

The parameters that will be subject to change in hyperparameter tuning are the Learning rate (Lr), Teacher Momentum and Center Momentum. In its default configuration, DINO uses the AdamW optimizer (recomended for ViT) with learning rate following an initial linear warmup from 0 to 5×10^{-4} (this value is then linearly scaled with the batch size) in the first 10 epochs and a cosine schedule (described in section 5.1) until the end of training, with the minimum value set as 10^{-6}. Teacher Momentum starts at 0.996 and is increased to 1 at the end of training following a cosine schedule as well. Finally, Center Momentum is set to 0.9 through the whole training process.

As we can see in the figure 17a, we have a small loss spike in the first 2 to 3 epochs but after that the training seems to be proceeding smoothly as the loss value continues to go down. However, the figure 17b suggests a big representation destruction (catastrophic forgetting) in the first 10 to 15 epochs which is rebuilt in the following epochs but never reaching the same accuracy values as the pretrained baseline. Our hypothesis is that these phenomenon occur due to the sudden change of domain from ImageNet to Stanford Cars, which caused the model to have a small destruction phase as it adjusts to the new representation space in the early stages of training.

6.2.2 Hyperparameter tuning: Learning rate

Even though the validation accuracy did not reach the values of our pretrained baseline following the fine-tuning with DINO in the default configuration, it did show some promise with constant loss improvement



(a) Training loss per batch

(b) Learning rate per batch

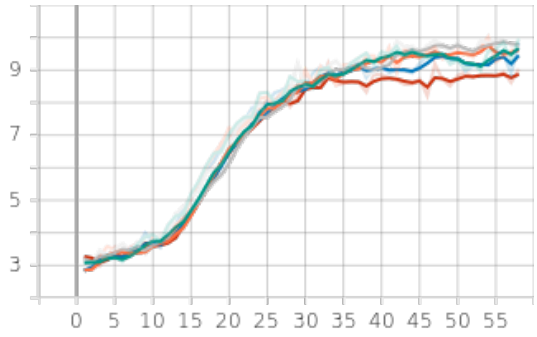
Figure 18: One epoch learning rate range test. Starting Lr values - Orange: 5×10^{-9}, Dark Blue: 5×10^{-8}, Red: 5×10^{-7}, Light Blue: 5×10^{-6}, Pink: 5×10^{-5}

during training as well as a mostly upward trend in validation accuracy after the "domain adaptation phase" in the first 15 epochs. This results prompted us to try to improve accuracy growing rate as well as achieve lower loss values with hyperparameter tuning. Due to the long training times (an average of 8h:25min per 100 epochs), going for a standard grid search of hyperparameters would not be viable, therefore scanning each one individually per order of importance would be the best course of action.

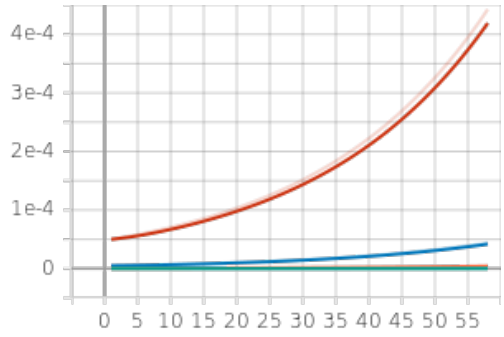
Starting with Lr, since this hyperparameter is not fixed and instead follows a cosine schedule, decreasing its value during training in order to stabilize the loss towards the end of a training session, we need to define an upper bound value to make sure we do not have an explosive gradient step in the early stages of training and the schedule should be able to prevent overshooting the minimum of the loss function. So we ran several one-epoch range tests, each one scanning one order of magnitude in search of a break point in which the loss value increases considerably, covering values for learning rate from 5×10^{-9} to 5×10^{-4} (Section 5.2).

However, as we can see in the figure 18, this proved to be a failure since we had mostly the same behavior across all ranges tested, implying that learning rate was almost a "useless" hyperparameter and changing it would have no significant impact in model performance during the first epoch of training, which is odd since this hyperparameter is usually one of the most important factors in model tuning across the whole spectrum of machine learning projects.

In order to find out what was causing this abnormality, the decision was to start with some basic tests to test some unlikely but possible theories. In the main code file for DINO framework, the researchers recommend AdamW as the optimizer when using ViT architecture for training, therefore this choice had not been questioned. The optimizer Adam and its variants use adaptive learning rate, which means that it calculates the learning rate for each parameter in the model at each iteration. This mechanism usually



(a) Training loss per batch



(b) Learning rate per batch

Figure 19: One epoch learning rate range test with SGD. Starting Lr values - Orange: 5×10^{-9}, Dark Blue: 5×10^{-8}, Red: 5×10^{-7}, Light Blue: 5×10^{-6}

brings the benefits of having an efficient learning rate for each parameter, since a global learning rate might not always be the best value individually. This reduces the need for extensive manual tuning by adjusting the Learning rates during training and overall contributes to more stable, efficient, and effective training of deep learning models. This meant that Adam could be partially responsible for these abnormal Learning rate tests. To verify this we ran the same one epoch range test, switching AdamW with the a basic optimizer, the standard Stochastic Gradient Descent. As we can see in figure 19, the change in optimizer produced loss curves very similar to the ones observed in figure 18, which means that the adaptive Learning rate applied in AdamW is not the cause for these results.

We then moved on to a more extreme test, changing the Learning rate to 0. Theoretically, if the Learning rate is 0, the optimizer would not be able to update weight values, which means that the model does not change with each iteration and, even though the batch changes, it should produce similar loss values across one epoch. The resulting loss curve, shown in figure 20 is again very similar to figures 18 and 19. This means that there are other hyperparameters associated not with the optimizer, but with the framework itself that have high influence in loss values and are not allowing us to tune model-specific hyperparameters.

Following this conclusion, in order to finally tune the Learning rate, we decided to disable all hyperparameters from DINO that could impact the loss values, one by one until we found out which were the ones that caused such similarities across all ranges of Learning rate values. The hyperparameter that caused this problem was the center momentum, which is responsible for controlling how fast the center from batch centering is updated. By turning off this mechanism, we are finally able to run the test with no disturbance and to fix an upper bound for the learning rate.

In the figure 21a, there is a spike in loss when the Learning rate has values between 5×10^{-4} and

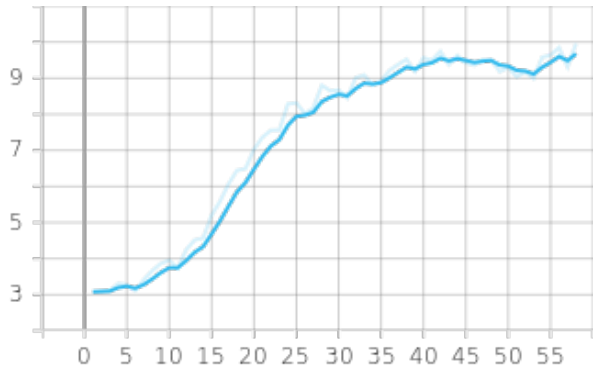
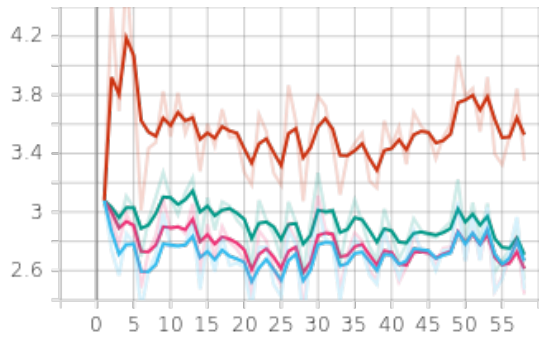
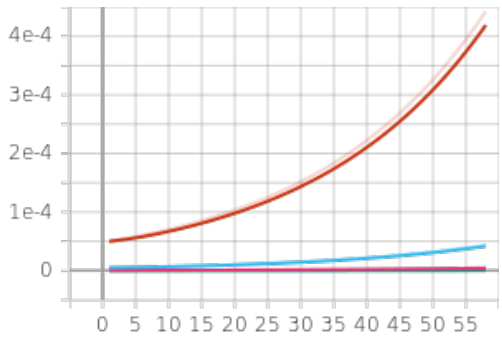


Figure 20: Loss across one epoch with Learning rate set to 0



(a) Training loss per batch



(b) Learning rate per batch

Figure 21: One epoch learning rate range test. Starting Lr values - Red: 5×10^{-4}, Blue: 5×10^{-5}, Pink: 5×10^{-6}, Green: 5×10^{-7}

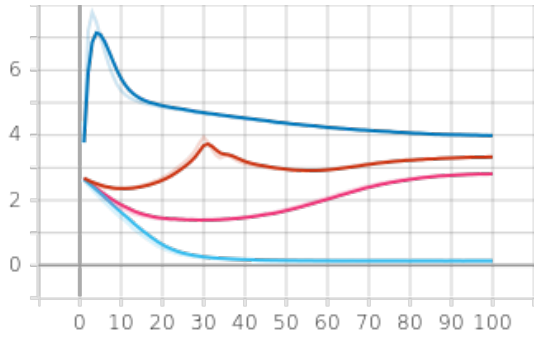
5×10^{-3}. By setting the Learning rate as 5×10^{-5} and applying a cosine schedule during training, we should maximize the "size" of the gradient steps while avoiding explosive gradients, resulting in more stable yet fast training sessions.

6.2.3 Hyperparameter tuning: Center Momentum

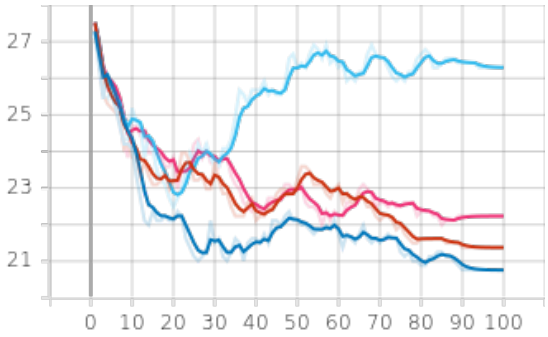
Following the learning rate tuning, it became clear that the most impactful hyperparameter in terms of loss values was the center momentum and it should be the main focus of our tuning. Our hypothesis is that the loss spike in 17a is related to the rate at which the batch center updates. The batch center is initialized as a tensor with the same shape as the network's output tensor and filled with zeros. This tensor is then updated with an exponential moving average rule of the last batch center controlled by the center momentum. This centering technique was designed for training a model from scratch, where the center is updated continuously and simultaneously with learning. However, the pretrained model's representation is centered in the representation space of the ImageNet dataset and, since the Stanford Cars dataset's domain is included in the ImageNet dataset (ImageNet also has images of vehicles, cars in particular), the Stanford Cars representation is expected to be highly localized in the ImageNet representation. This causes a fast update of the teacher's representation in few iterations that the student model fails to keep up with. While the batch center converges to the expected values, we get a big spike in loss (from values around 3 to values close to 7) since the student and teacher representations that should be close in the representation space due to pretraining, will now be pushed apart thanks to the centering of the teacher representation. Having low loss values right at the start would not encourage the model to change and to search for new features to learn, making this fine-tuning process meaningless. However, this spike in loss, which could be helpful in the early stages of the training process, seems to be accompanied by a destruction of the representation, as shown in 17b where we can see a huge decrease of accuracy in the first few epochs.

Some degree of catastrophic forgetting might be an unavoidable step due to the change in domain and it might still be the best option when compared with training from scratch. In order to try to slow down the center update and thus mitigate the degradation on the representation in the first epochs of the training process, we decided to test higher values of center momentum. By slowing down the rate at which the batch center updates, the period of adaptation will increase, but hopefully it will make this process more stable and reduce the catastrophic forgetting.

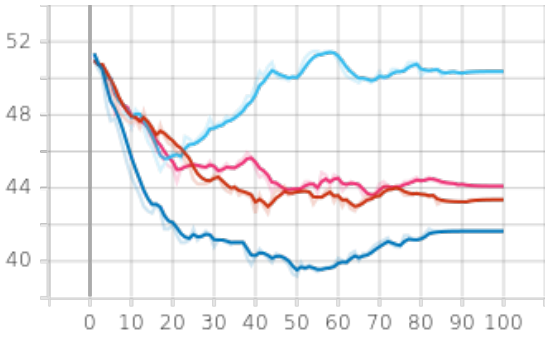
In this new experiment we trained the model for 100 epochs using higher values for center momentum than 0.9 (default value for DINO) which corresponds to slowing down the batch center update via



(a) Training loss per epoch



(b) Top 1 KNN validation accuracy



(c) Top 5 KNN validation accuracy

Figure 22: Center momentum test: Values - Dark Blue: 0.97, Red: 0.999, Pink: 0.9997, Light Blue: 0.9999

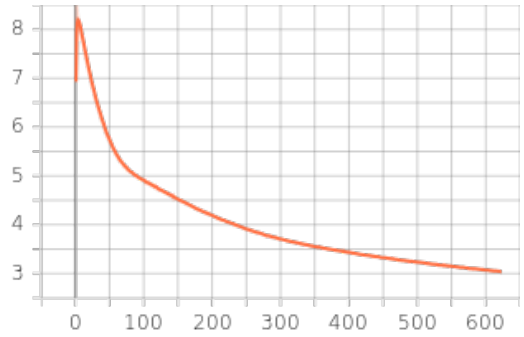
exponential moving average. The goal is to select the one with a smaller loss spike as well as a smaller accuracy decrease.

The results in figure 22 prove our hypothesis correct. By slowing down the batch center update rate (increasing center momentum) we were able to reduce the loss spike and the catastrophic forgetting. As we can see in 22a and 22b, the light blue plot which corresponds to the highest value (0.9999) produced the best outcome.

6.2.4 Scaling

To test the effectiveness of the hyperparameter tuning, we ran extended training sessions in order to evaluate how the training progresses over time. For these experiments, we disabled training stabilizers like learning rate scheduling for the student model and teacher momentum scheduling for the teacher model in order to observe the raw performance of the model and when does the training efficiency starts to drop, as the previous tests suggest that we are still far away from convergence and there is still no need for them (these schedulers slow down training, which can be mistaken as convergence).

In the first experiment we fixed the learning rate to the best value determined in section 6.2.2 (5×10^{-5})



(a) Training loss per epoch



(b) Top 1 KNN validation accuracy



(c) Top 5 KNN validation accuracy

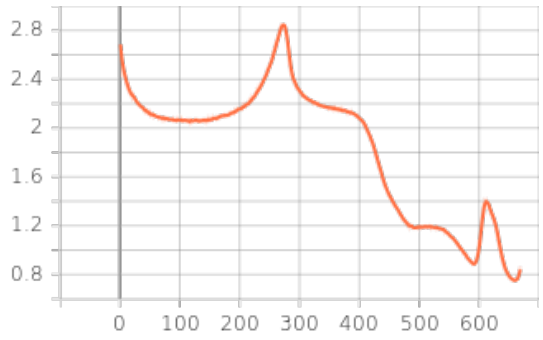
Figure 23: 600 epochs with fixed learning rate, fixed teacher momentum and default center momentum (0.9)

and teacher momentum to the default value (0.996). Every other parameter was left as default from the source code.

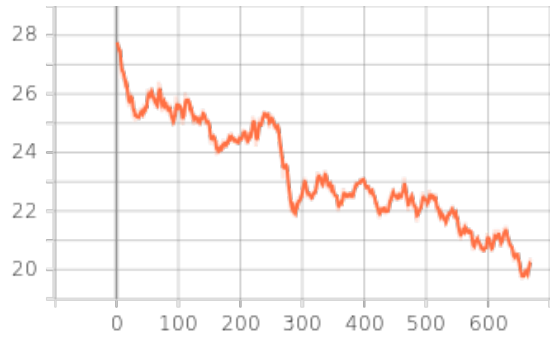
By only changing the learning rate (figure 23), we managed to significantly reduce the decrease in validation accuracy from the experiment in section 6.2.1 where the lowest accuracy achieved was approximately 14% at around 15 epochs. In this experiment, the lowest accuracy score was approximately 18%, much later in training at around 575 epochs, in a spike that was instantly recovered. Even though we are still far away from our goal, this change in learning rate already produced meaningful improvements in training performance over time.

The next experiment aims to evaluate the effect of the center momentum optimization. As in the previous experiment, the learning rate and teacher momentum scheduling was disabled and this time we also changed the center momentum to the best value obtained in 6.2.3 (0.9999).

As discussed in section 6.2.3, the loss values start much lower than the previous experiment due to the slower update of the batch center and as a result, we managed to achieve loss values under 1 (figure 24). However this loss improvement does not seem to correlate to a significant improvement in accuracy. The accuracy values consistently keep going down over time and in this experiment, the last value was



(a) Training loss per epoch

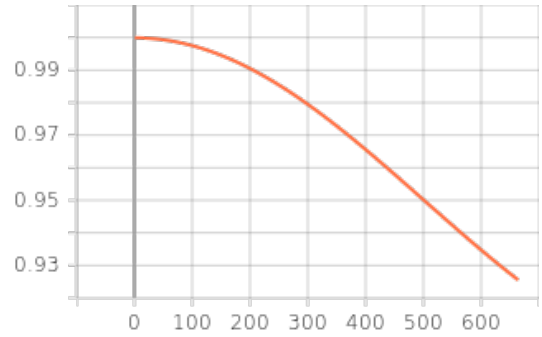


(b) Top 1 KNN validation accuracy

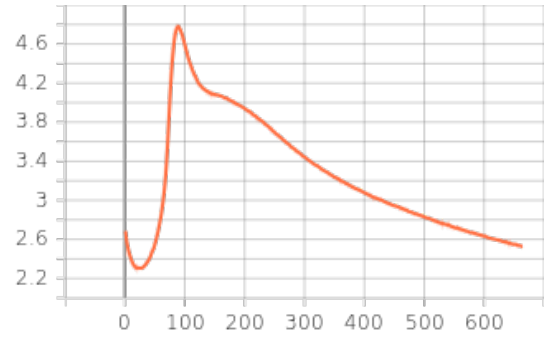


(c) Top 5 KNN validation accuracy

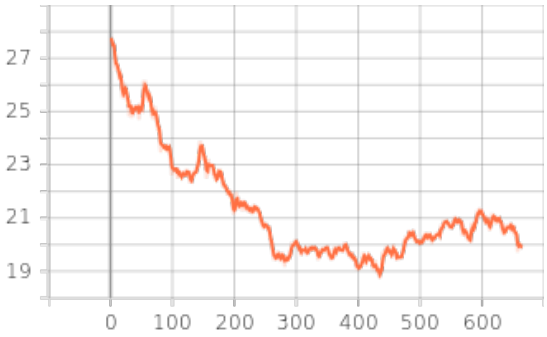
Figure 24: 600 epochs with fixed learning rate, fixed teacher momentum and fixed center momentum (0.9999)



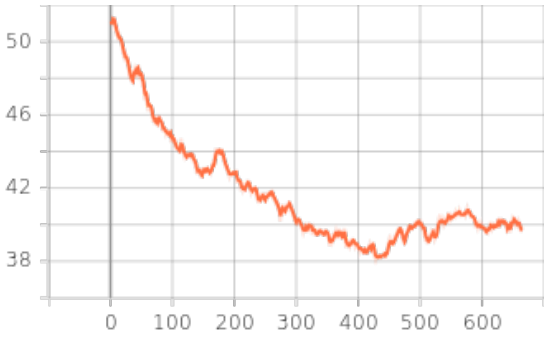
(a) Center momentum per epoch



(b) Training loss per epoch



(c) Top 1 KNN validation accuracy



(d) Top 5 KNN validation accuracy

Figure 25: 600 epochs with fixed Learning rate, fixed teacher momentum and center momentum schedule

even lower than in the previous one.

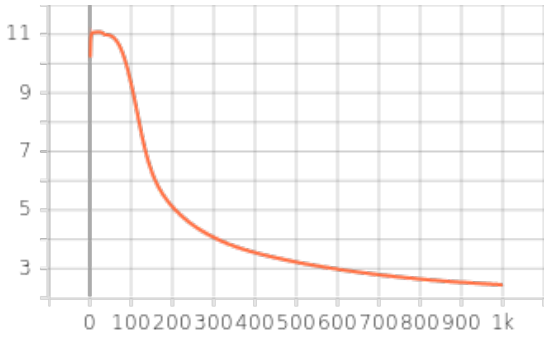
Considering that a slower batch center update is mostly important in the beginning of training, we were motivated to test how well a center momentum cosine scheduler similar to that of the learning rate could work, in which we accelerate the update over time by lowering this parameter from 0.9999 to 0.9 (default value)

This change proved to be fruitless as we got a sharper decrease in accuracy values and the model ended up having a worse performance in terms of loss values (figure 25).

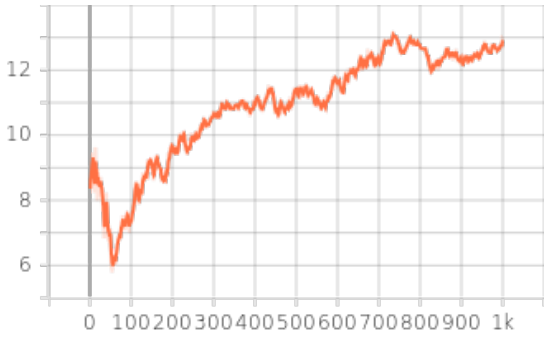
6.2.5 Partial pretrained model

After optimizing two major hyperparameters, one from the framework side and one from the optimizer side, and still failing to achieve any meaningful improvements on accuracy, we concluded that hyperparameter optimization would not be near enough to reach a threshold of accuracy that allows us to validate the use of self-supervised learning to solve this type of problems.

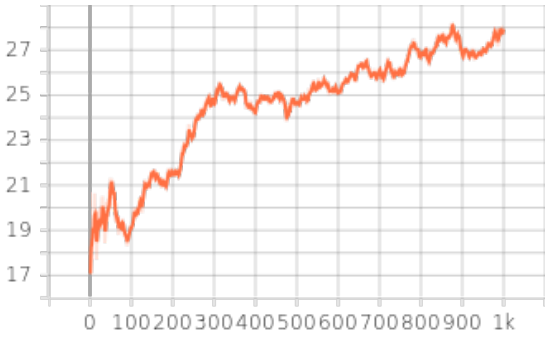
Another problem that might be occurring is that the Stanford Cars dataset belongs to a domain which is a subdomain of the ImageNet dataset in which our model is pretrained on. In other words, the model was pretrained to group images of cars that might belong to diferent classes in the Stanford Cars dataset



(a) Training loss per epoch



(b) Top 1 KNN validation accuracy



(c) Top 5 KNN validation accuracy

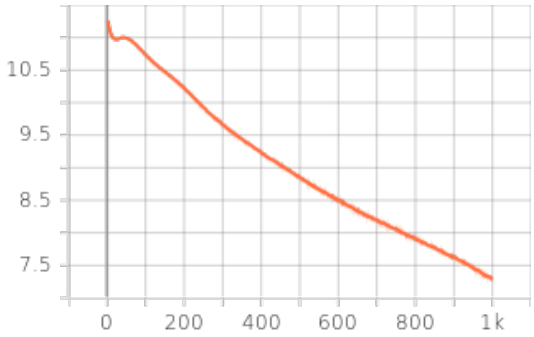
Figure 26: 1000 epochs with reset layers, fixed Learning rate and fixed teacher momentum

under the same class from the ImageNet dataset, like sports car (label 817 in ImageNet) or pickup truck (label 717 in ImageNet). This means that in this fine-tuning process, the model needs to "forget" some features that it learned during pretraining and learn new high level features that will allow the model to separate the dataset in the desired way.

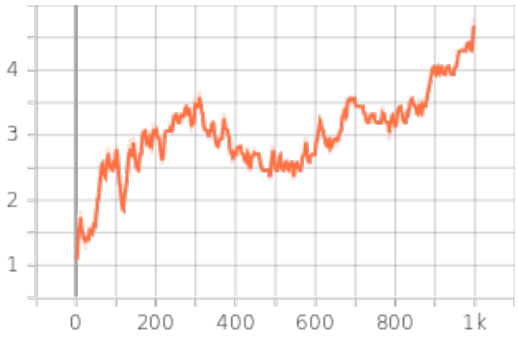
Considering that image models like CNNs and ViTs learn their features in a hierarchical way, with low level features being learnt in the first few layers and high level features in the last few and that our goal is to learn new high level features, this means that we have to "rebuild" the last layers of our model. Looking back at the previous results, the lack of improvement in accuracy values means that we have not been able to do this successfully and that we need a different approach.

Following these considerations, we decided to re-initialize one third of the attention blocks (four blocks) and fine-tune the model, training for 1000 epochs with fixed learning rate, teacher momentum and center momentum (0.9999).

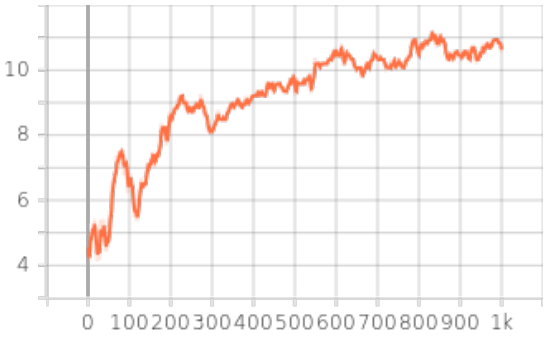
As we can see in figure 26, our initial loss spike is even bigger than with the pretrained model, reaching loss values of 11 instead of around 8. Even though after that we get a decreasing trend in loss values and an increasing trend in accuracy values which is positive, it seems to hit the top 1 accuracy peak at 720 epochs and that the model plateaued. This accuracy peak is also very low, standing at 13%, way lower



(a) Training loss per epoch



(b) Top 1 KNN validation accuracy



(c) Top 5 KNN validation accuracy

Figure 27: 1000 epochs training from scratch with fixed Learning rate and fixed teacher momentum

than the 26.5% achieved with the pretrained model, which makes this technique not valid.

6.3 Training from scratch

As a final attempt using the DINO framework, we decided to see if training from scratch would be more effective than using a pretrained model like we did in the previous experiments. Considering our model now has a different starting point in terms of loss function optimization, we ran an one epoch learning rate range test to find our upper bound for this hyperparameter. We trained the clean model for 1000 epochs with fixed learning rate and teacher momentum so our conclusions do not get skewed due to the stabilizing effects of schedulers of these hyperparameters.

From figure 27, we can confirm our initial assumptions: training a ViT network with low amounts of data is just not viable. Even though our loss values have a steady descending trend and still have a lot of room to improve (ideally we'd have values between 0 and 1 instead of 7.5 at the end of training) it would require at least 3000 epochs of training, assuming that the loss values keep decreasing at the same rate they did in figure 27a. Still we do not have any warranty that our accuracy values would keep increasing and not plateau like it did in section 6.2.5.

Table 1: Summary of the experiments

Starting model	**Learning rate**	**Teacher momentum**	**Center momentum**	**Epochs**	**Top1 kNN validation accuracy**				**Plots**
					Start	**Min**	**Max**	**End**	
pretrained	5×10^{-4} (schedule)	0.996 (schedule)	0.9	100	27.5%	14%	27.5%	19%	17
pretrained	10^{-5} (fixed)	0.996 (fixed)	0.9 (fixed)	600	27.5%	19.8%	27.5%	21.5%	23
pretrained	10^{-5} (fixed)	0.996 (fixed)	0.9999 (fixed)	600	27.5%	19.9%	27.5%	20.3%	24
pretrained	10^{-5} (fixed)	0.996 (fixed)	0.9 (schedule)	600	27.5%	18.9%	27.5%	19.9%	25
pretrained (re-initialize 1/3)	10^{-5} (fixed)	0.996 (fixed)	0.9999 (fixed)	1000	8.4%	5.8%	13.1%	12.9%	26
clean model	10^{-5} (fixed)	0.996 (fixed)	0.9999 (fixed)	1000	1%	1%	4.7%	4.7%	27

6.4 Embedding space analysis

In order to obtain a better understanding of the representations produced by the models resulting from the experiments described in Sections 6.1, 6.2, 6.3, we analyzed the characteristics of each checkpoint's embedding space. We performed a quantitative analysis for the collapse of the embedding space, as well as a qualitative analysis relying on the visualization of retrieved neighbours for certain query images.

6.4.1 Collapse analysis

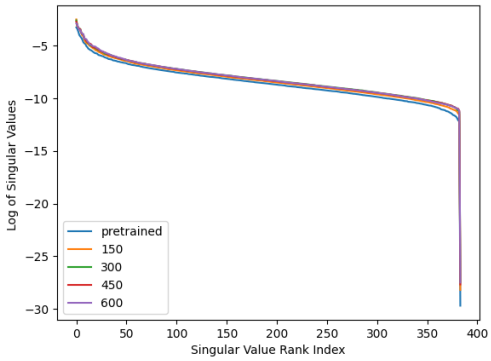
In the previous experiments we managed to consistently achieve low loss values which should be an indicator that the training procedure was going well. However, this did not translate to any improvement in validation accuracy, where we did not even manage to reach the pretrained model starting point of 26.54%. A summary of all these experiments can be summarized in table 1. This made us question the quality of the representation being learned by the model and our conclusion was that either the model was collapsing or the features that it was learning were not useful for the classification task at hand.

For the collapse analysis of the model, we used the method described in section 5.3 to evaluate the representation space of the validation set using the checkpoints of models trained on the experiments of sections 6.2.4,6.2.5 and 6.3 at several points in training. Considering we want to evaluate the representation used for classification, we will be analysing the collapse on the last layer of the backbone of the network used in training which has a dimension of 384.

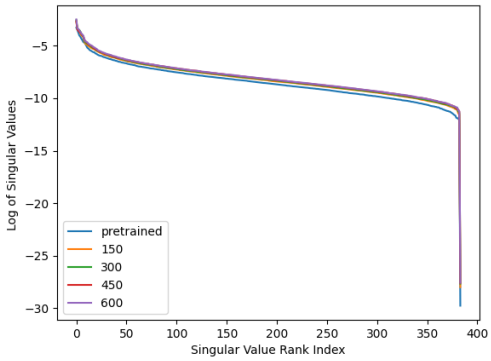
In figure 28 we can see that the backbone managed to avoid both complete and dimensional collapse since the information encoded across different dimensions of the representation was well distributed and no dimension possesses redundant information. In every experiment, we managed to improve this metric throughout the training procedure and in figures 28a, 28b, 28c and 28d we can see that in experiments where we started with the pretrained model or a partial pretrained model, we manage to slightly surpass this baseline.

However, as suggested in (Balestriero et al., 2023), some methods might have collapsed representations after the projector even though the representation is not collapsed before. Since loss is calculated at the projector level, this means that a collapsed representation after the projector might be affecting the training process and producing lower loss values than expected, stagnating the improvement in validation accuracy.

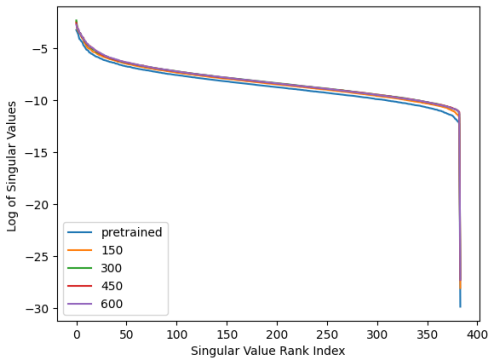
In order to evaluate the collapse of the representation after the projector we decided to run the same method on two layers of the projector: the last layer and an intermediate layer. Since the last layer of the



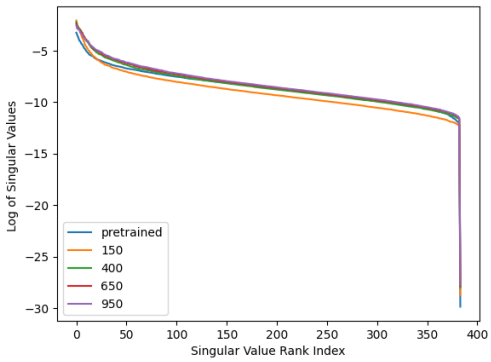
(a) 600 epochs with fixed Learning rate and fixed teacher momentum.



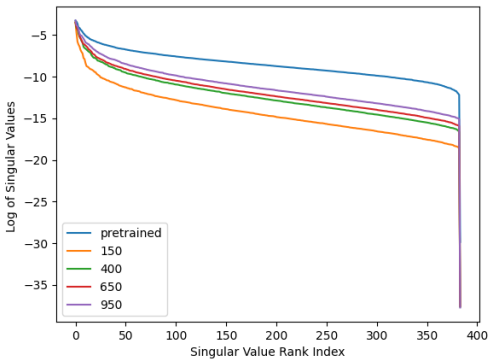
(b) 600 epochs with fixed Learning rate, fixed teacher momentum and fixed center momentum.



(c) 600 epochs with fixed Learning rate, fixed teacher momentum and center momentum schedule.



(d) 1000 epochs with reset layers, fixed Learning rate and fixed teacher momentum.



(e) 1000 epochs training from scratch with fixed Learning rate and fixed teacher momentum.

Figure 28: Collapse analysis of the output after teacher backbone at several points during training.

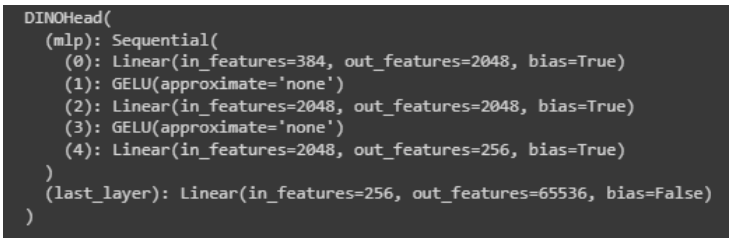


Figure 29: Projector structure

projector is a weight normalization layer with output dimension of 65536 and there is no activation layer between this and the previous one, we can infer the collapse of the representation from the previous layer instead. We proceeded with the analysis of the layers labeled 4 and 2 in figure 29 which have 256 and 2048 output dimensions respectively.

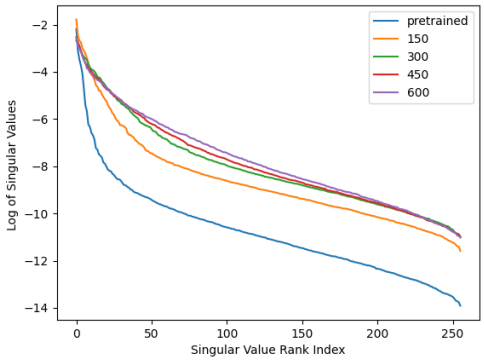
The results in figure 30 confirm that the problem during training was not the collapse of the network since both the backbone and projector are not collapsed and have very similar distributions of information throughout all output dimensions. Even the intermediate layers of the projector are not collapsed, although there is a severe drop-off point in amount of information in each dimension as we can see in figure 31. This drop-off suggests that perhaps a smaller intermediate layer would be preferred to reduce execution times although it is not very relevant.

6.4.2 Neighbour retrieval

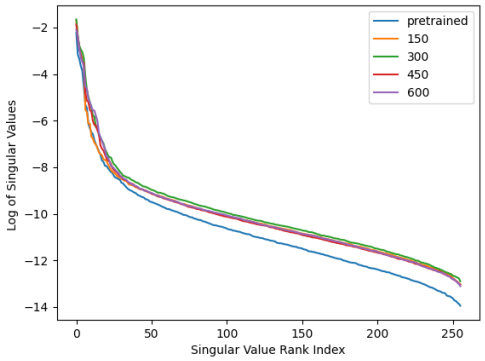
Since the collapse of the representation was not the issue leading to the bad results obtained, the only explanation is that the model is not learning meaningful features that allow us to separate the dataset into the appropriate classes. To verify this hypothesis we decided to see which images have close representations in the representation spaces of the models trained by running a KNN retrieval (Section 5.4.1). This will allow us to see how close wrong classifications are from being right. For example, the model might be grouping cars from the same make together but failing to classify their model, resulting in bad accuracy.

We randomly selected ten query images from the validation set and selected the five nearest neighbours from the training set, in ascending order of distance. For consistency, the same ten query images were used for all models evaluated. In figures 32, 33, 34, 35, 36 and 37 we can see the the results of this test for the pretrained model, as well as the models evaluated in section 6.4.1. The first column of these figures are the query images and, for each row, the next five columns are the nearest neighbours from the respective query image.

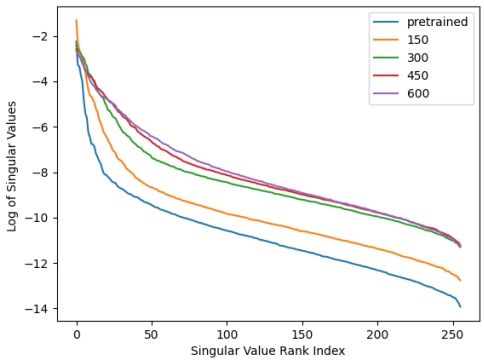
From these results, it is obvious that the model is clearly failing to learn features that are relevant for the task of classification. For every query point, the top five nearest neighbours from the training set in the



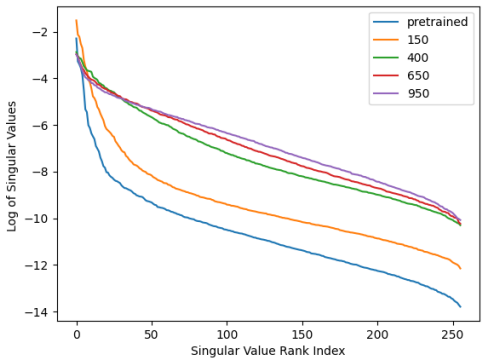
(a) 600 epochs with fixed Learning rate and fixed teacher momentum.



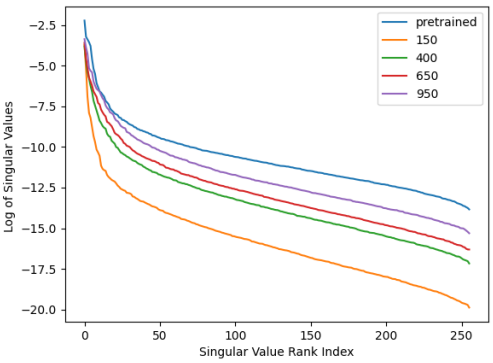
(b) 600 epochs with fixed Learning rate, fixed teacher momentum and fixed center momentum.



(c) 600 epochs with fixed Learning rate, fixed teacher momentum and center momentum schedule.

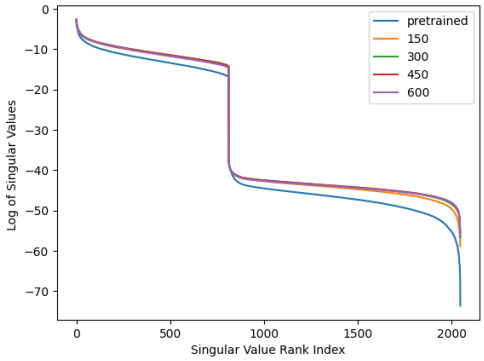


(d) 1000 epochs with reset layers, fixed Learning rate and fixed teacher momentum.

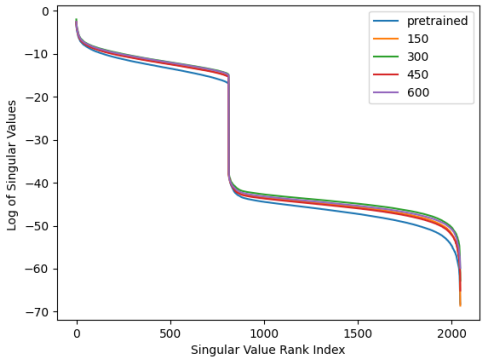


(e) 1000 epochs training from scratch with fixed Learning rate and fixed teacher momentum.

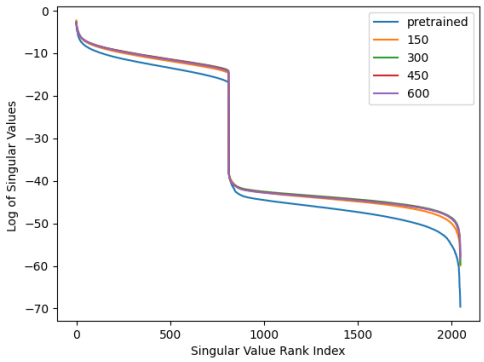
Figure 30: Collapse analysis of the output after projector at several points during training.



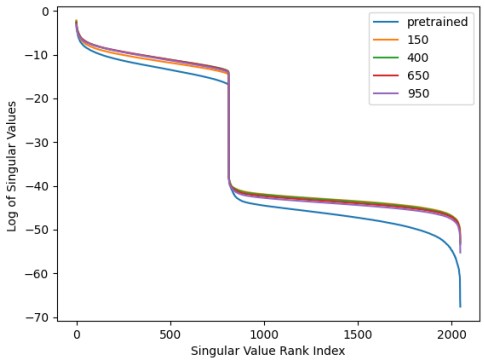
(a) 600 epochs with fixed Learning rate and fixed teacher momentum.



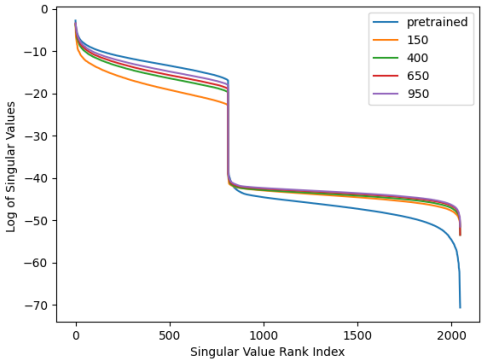
(b) 600 epochs with fixed Learning rate, fixed teacher momentum and fixed center momentum.



(c) 600 epochs with fixed Learning rate, fixed teacher momentum and center momentum schedule.



(d) 1000 epochs with reset layers, fixed Learning rate and fixed teacher momentum.



(e) 1000 epochs training from scratch with fixed Learning rate and fixed teacher momentum.

Figure 31: Collapse analysis of the output after intermediate projector layer at several points during training.



Figure 32: KNN neighbour retrieval from pretrained model.

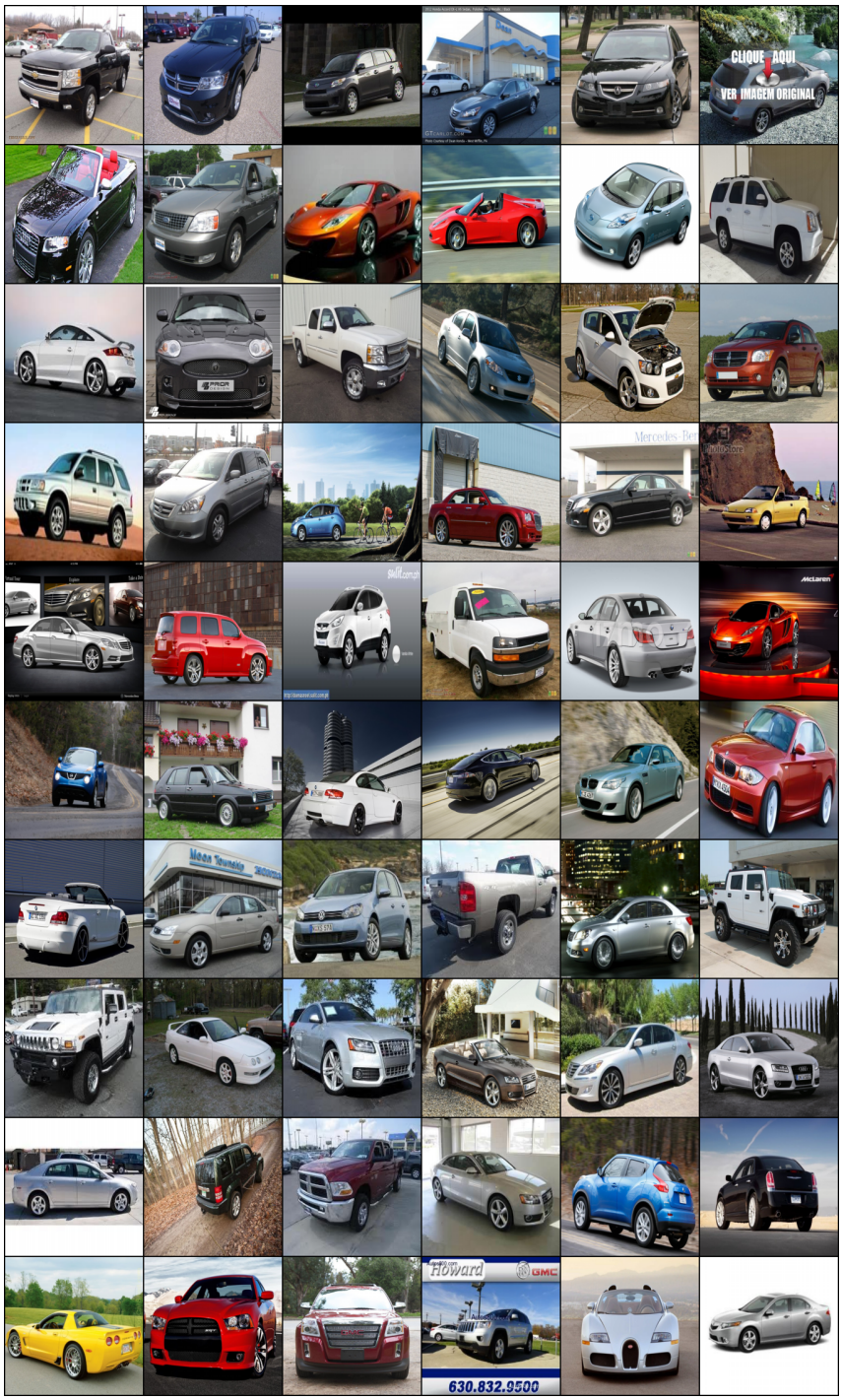


Figure 33: KNN neighbour retrieval from 600 epochs with fixed Learning rate and fixed teacher momentum.



Figure 34: KNN neighbour retrieval from 600 epochs with fixed Learning rate, fixed teacher momentum and fixed center momentum.

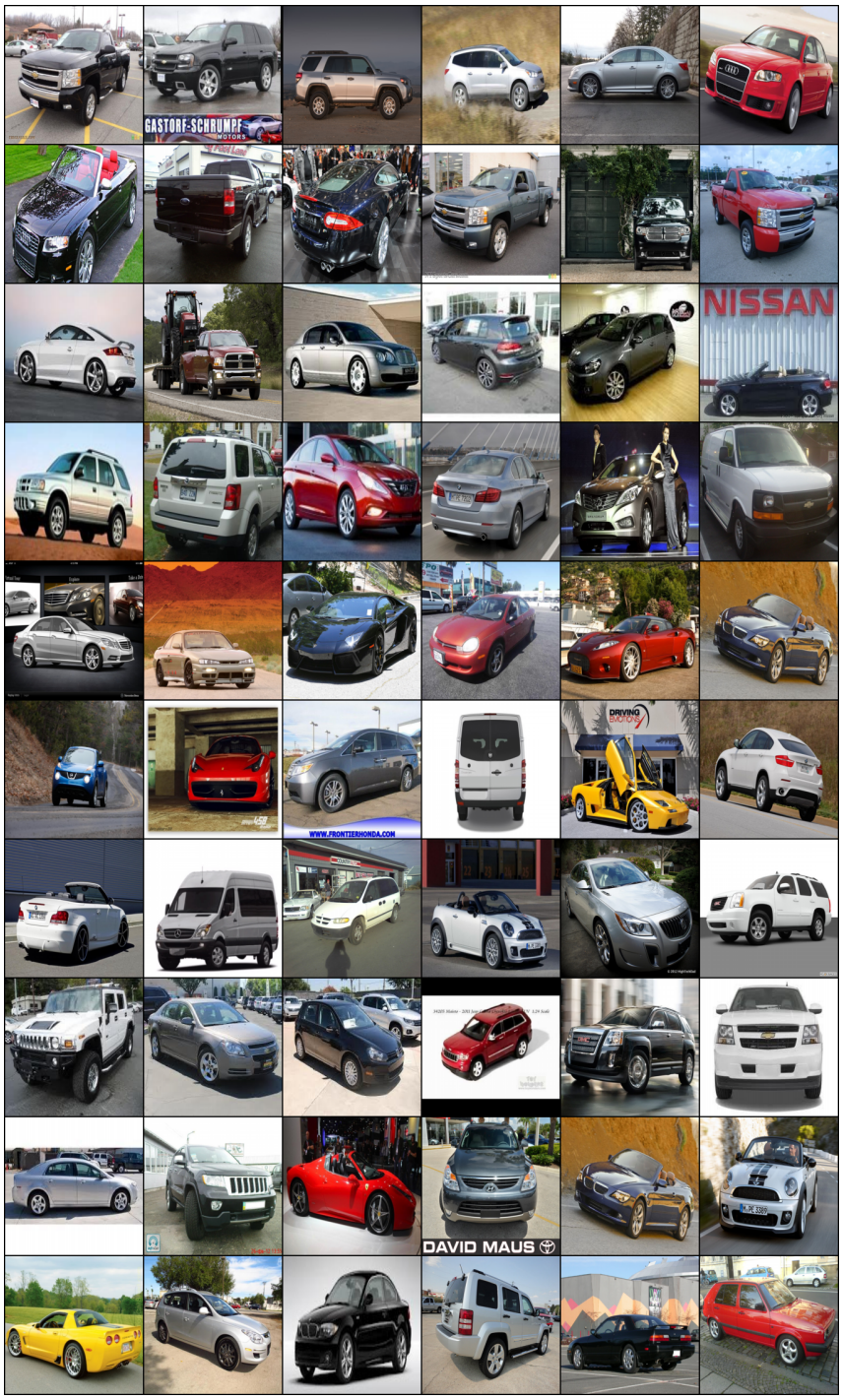


Figure 35: KNN neighbour retrieval from 600 epochs with fixed Learning rate, fixed teacher momentum and center momentum schedule.

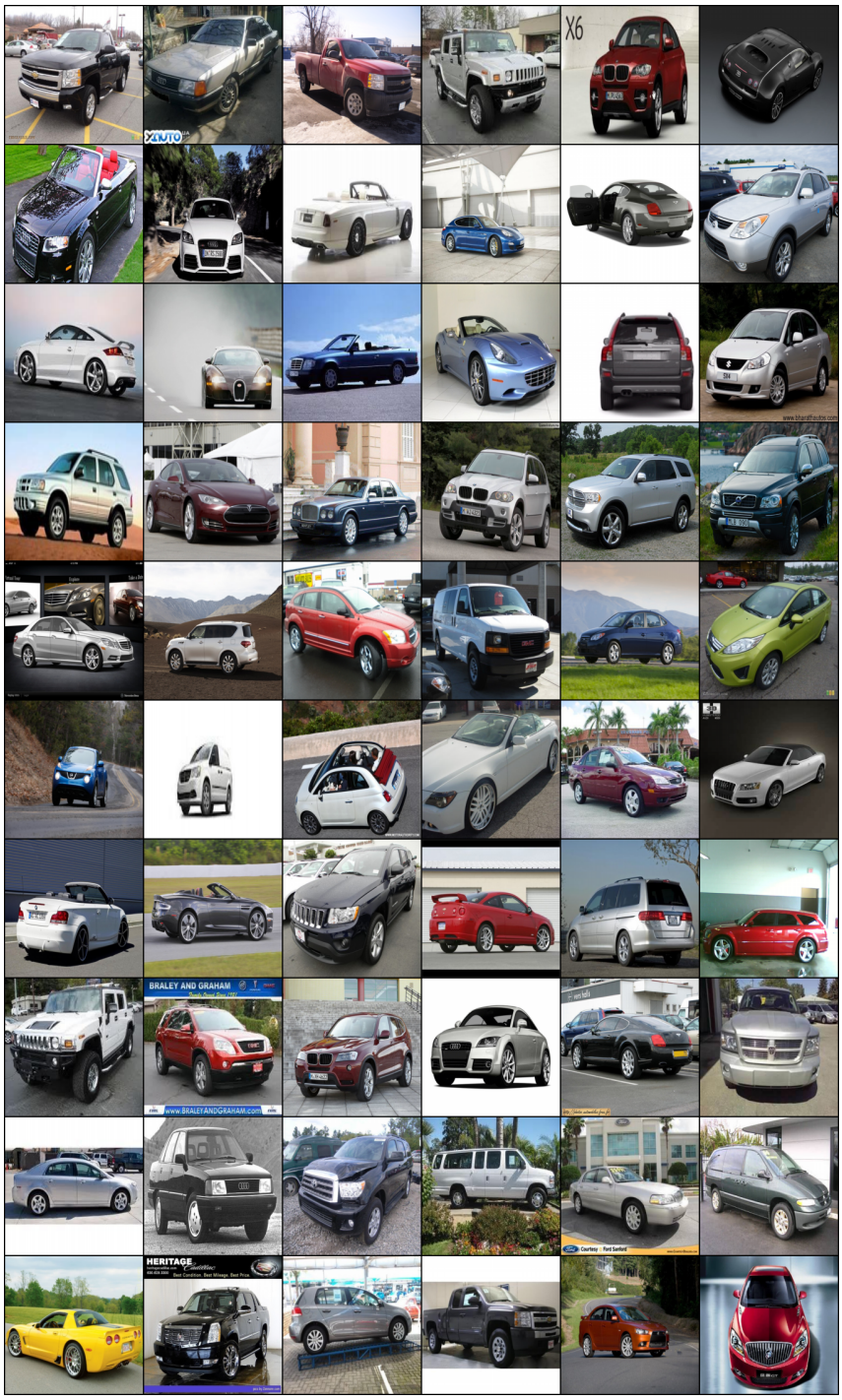


Figure 36: KNN neighbour retrieval from 1000 epochs with reset layers, fixed Learning rate and fixed teacher momentum.



Figure 37: KNN neighbour retrieval from 1000 epochs from scratch with fixed Learning rate and fixed teacher momentum.

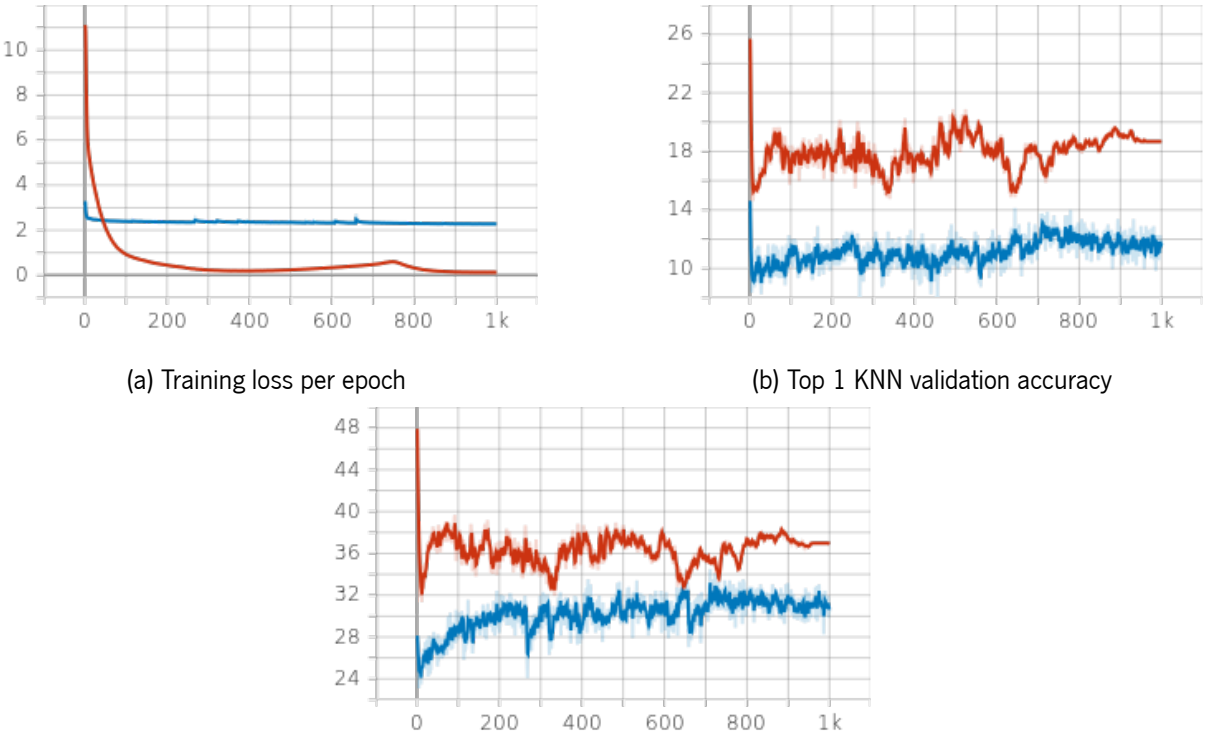
representation space do not seem to share any characteristic between them and the query point. Every model fails to even group the car embeddings by category of car, such as sedans, convertibles or SUVs for example. It is very hard to see what criteria/features the model is using to group embeddings in the representation space but it seems clear that they are not useful to identify either model or make of any vehicle.

6.5 DINO vs SimCLR

As described in section 3.4, in order to prevent collapse, DINO uses a combination of sharpening and batch centering of the output, encouraging separability of the embeddings in the representation space. Since the task of the Stanford Cars dataset is fine-grained classification, which is more challenging than normal classification due to the very subtle differences between classes, this implicit separation might not be enough to allow a KNN classifier or a linear classification head to segment the high dimensional representation space in a way that each segment only contains embeddings belonging to a certain class. Also, the sharpening of the output will attribute less importance to features with lower values which might correspond to the subtle differences that we are actually looking for.

To verify if this is the case, we compared DINO with another self-supervised learning framework, SimCLR, that instead uses an explicit separation technique in the form of a loss function with negative examples (section 3.3). In this experiment, we trained a model using DINO and other model using SimCLR on the Stanford Cars dataset for 1000 epochs. For both runs, the starting point was a checkpoint of a pretrained model using the corresponding self-supervised learning framework on the ImageNet dataset. For a fair comparison, both runs started with a clean projector, since the pretrained checkpoints available from SimCLR only contained weights for the backbone and not for the projector. The backbone model architecture chosen for SimCLR was the ResNet50(1X) due to the similar parameter count when compared with ViT-S16.

For the SimCLR run, we used the default settings, except for the learning rate, which was set to 0.7 ($5.6 \times BatchSize/256$) following a one epoch learning rate range test. The optimizer used was the LARS optimizer with weight decay of 10^{-6}. Learning rate also follows a linear warmup for 10 epochs and then a cosine decay scheduler without warm restarts (similar to DINO). For the DINO run, we used the 'optimized' values for learning rate (10^{-5}) and center momentum (0.9999), while the rest of the settings were the default. Unfortunately, the NVIDIA A100 GPUs were no longer available for these runs, forcing us to switch to a NVIDIA RTX 3080ti, which had significantly lower GPU memory when compared with the



(a) Training loss per epoch

(b) Top 1 KNN validation accuracy

(c) Top 5 KNN validation accuracy

Figure 38: 1000 epochs training Dino (red) vs SimCLR (blue)

previous GPU (12 GB compared with 40 GB). Due to these new memory constraints, the batch size was reduced from 128 to 32.

When looking at the results in figure 38, it seems that both frameworks behave in a similar way during training. Although a bit more robust to it, SimCLR is also affected by catastrophic forgetting, evidenced by a sharp drop in validation accuracy in the first few epochs of training. SimCLR lower values of validation accuracy during training when compared to DINO can be explained by the fact that the starting checkpoint has a much lower performance on the Stanford Cars dataset. Also, as discussed in section 3.3, SimCLR is dependent on larger batch sizes in order to sample sufficient negative pairs for the contrastive loss, which is not the case in these runs due to GPU memory constraints. With this experiment we can conclude that the problem does not lie in specific self-supervised learning frameworks, but might be more related to the incapability of separating images from very similar classes using self-supervision.

6.6 Few-shot classification

For the final experiment, we are evaluating how well some trained models in this chapter perform in a low-shot classification context. The baseline for this evaluations are the pretrained checkpoints using DINO

and SimCLR on the ImageNet dataset. Then we are evaluating the models trained in section 6.5 and finally the model with re-initialized layers trained in section 6.2.5. For this experiment we report the accuracy of a trained linear layer (on top of the frozen features) on the test set of Stanford Cars (table 2). This linear layer is trained for 100 epochs with batch size of 128 using cross entropy loss. The optimizer used is AdamW with learning rate of 0.005 $(0.01 \times BatchSize/256)$ following a cosine decay schedule during training and weight decay of 10^{-6}. The training and validation sets are subsets of the original Stanford Cars training set, with 10 examples per class in each set.

Table 2: Few-shot classification results

Model checkpoint	**Linear evaluation accuracy**
Dino pretrained checkpoint	35.93%
SimCLR pretrained checkpoint	15.01%
DINO trained with re-initialized projector on Stanford Cars (fig. 38)	15.86%
SimCLR trained with re-initialized projector on Stanford Cars (fig. 38)	15.06%
DINO with re-initialized layers (33%) on Stanford Cars (fig. 26)	10.05%

From table 2, we can see that the results from this experiments using DINO are consistent with the validation accuracy values during training with a KNN classifier. Due to catastrophic forgetting, the performance of the models trained does not even reach half of the performance of the pretrained checkpoint and is even worse for the model with re-initialized layers due to low amounts of data to train a transformer network.

SimCLR on the other hand seems to be less vulnerable to catastrophic forgetting due to the contrastive loss and was able to at least reach the same performance as the pretrained checkpoint. During training (figure 38) the loss values stagnated in a relatively high value early in the training session. With better hyperparameter choices it should be possible to lower these loss values and maybe reach a better performance but the baseline accuracy is so low that it is not worth to explore further.

When comparing the performances of both baseline models, SimCLR might have such a disparity in performance due to the lack of normalization of the embeddings, which DINO possesses. With L2 normalization, embeddings are constrained to the surface of a unit sphere, where the dot product of two embeddings is equivalent to the cosine similarity, resulting in embeddings more evenly distributed in terms of magnitude and thus making it easier to draw decision boundaries (Wang et al., 2017). As such, without embedding normalization, the resulting classes may not be as uniformly separable as with normalization, giving an edge to DINO on linear classification.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

During this dissertation, we were faced with several challenges that prevented us to reach the goal of designing a framework that combines self-supervision and semi-supervision in order to classify make and model of vehicles in color images.

The main challenge that we tried to address was the catastrophic forgetting that plagued the models during the domain adaptation, from ImageNet to Stanford Cars. Due to catastrophic forgetting, we were not able to fully utilize the benefits of having a stronger starting point during training from a model pretrained in a large general dataset, as the baseline performances of these pretrained models were almost cut in half in the first few epochs of training, resulting in a starting point only a bit better than a initialized model.

However, even after this phase where the model accuracy on the validation set dropped, the model still failed to learn information from the Stanford Cars dataset useful enough to improve its performance significantly during the rest of the training session, as it still failed to reach the performance of the baseline pretrained model before the catastrophic forgetting phase, which was not the expected behavior. Our hypothesis is that this behavior can be attributed to the dataset used lacking the quantity of data required, as self-supervised learning techniques and models based on the transformer model architecture are known to require large amounts of data to train, which clearly is not the case for the Stanford Cars train set. Although models pretrained on larger datasets are fine-tuned in the Stanford Cars as part of their evaluation (like DINO, for example), this is a standard supervised fine-tuning process. As far as we can tell from the research on this topic, self-supervised fine-tuning is a novel approach that has not been studied much and is fairly safe to assume that it requires large amounts of data to perform, similarly to how normal supervised training requires less data than self-supervised training.

Even though our proposed framework did not reach a satisfactory level of performance, due to the problems mentioned earlier, it is still too premature to discard it completely. Only with further exploration

on this topic can we finally conclude if self-supervised methods by themselves are good enough to learn intricate features that allow the model to perform well in fine-grained image classification tasks such as vehicle make and model recognition, and if so, if our framework provides a valid solution to this problem.

7.2 Future work

Based on the results and limitations encountered in this dissertation, several promising avenues for future investigation have emerged. Our findings highlighted three key areas that warrant further exploration: addressing catastrophic forgetting during self-supervised fine-tuning, investigating the impact of data quantity on model performance, and expanding the framework's applicability beyond fine-grained classification tasks. These research directions aim to enhance the robustness and practical applicability of our approach while contributing to the broader understanding of self-supervised fine-tuning.

Regarding the catastrophic forgetting, the study and application of specific techniques from traditional supervised fine-tuning designed to solve this issue should be adapted into the self-supervised fine-tuning step of our framework, instead of trying to reduce catastrophic forgetting with standard hyperparameter tuning. One of the techniques to consider are gradual unfreezing of layers, where the model starts with most layers frozen, allowing the model to gradually adapt to new data, protecting for longer the lower layers, which usually detect the more general features that are useful across domains, while adapting first the layers responsible for the target task specific features. We tried to encourage the model to adapt the higher layers first with the reset of part of the network, but looking back, this approach would probably be more effective. Another technique to consider that follows the same motivation would be layer-wise learning rates, where lower layers are preserved by having extremely small learning rates, while higher layers have higher learning rates to adapt to the new domain. Although both techniques serve the same purpose and work in similar ways, the effects of their combination can also be studied.

With respect to the data quantity problem, the use of a larger unlabeled vehicle dataset could be used for the self-supervised fine-tuning step, while relying on the Stanford Cars dataset only for few-shot classification and evaluations due to the dataset already being labeled. With this we could have some insights to the amounts of data required for our framework to be effective and a viable solution. With more data and more computational resources, we could also experiment with longer training sessions by using distributed computing with several GPUs.

Finally, for a long-term research we could consider the generalization of this framework to tasks other than fine-grained classification, and experiment with self-supervised approaches other than DINO and

SimCLR. With the landscape of self-supervised image classification constantly evolving, new advancements are achieved and promising new frameworks appear that could significantly increase the performance and viability of our framework.

Bibliography

What is computer vision? https://www.ibm.com/topics/computer-vision. [Accessed in: 10-October-2024].

Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, Avi Schwarzschild, Andrew Gordon Wilson, Jonas Geiping, Quentin Garrido, Pierre Fernandez, Amir Bar, Hamed Pirsiavash, Yann LeCun, and Micah Goldblum. A cookbook of self-supervised learning, 2023.

Saulo Barreto. Choosing a learning rate. https://www.baeldung.com/cs/ml-learning-rate. [Accessed in: 10-October-2024].

Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9912–9924. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/70feb62b69f16e0238f741fab228fec2-Paper.pdf.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers, 2021.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020a.

Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. Big self-supervised models are strong semi-supervised learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22243–22255. Curran Associates, Inc., 2020b. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/fcbc95ccdd551da181207c0c1400c655-Paper.pdf.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

Linus Ericsson, Henry Gouk, and Timothy M. Hospedales. How well do self-supervised models transfer?, 2021. URL https://arxiv.org/abs/2011.13377.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

Quentin Garrido, Randall Balestriero, Laurent Najman, and Yann Lecun. Rankme: Assessing the downstream performance of pretrained self-supervised representations by their rank, 2023. URL https://arxiv.org/abs/2210.02885.

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. URL https://arxiv.org/abs/2006.07733.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning, 2019. URL http://arxiv.org/abs/1911.05722.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1031. URL https://aclanthology.org/P18-1031.

Tianyu Hua, Wenxiao Wang, Zihui Xue, Yue Wang, Sucheng Ren, and Hang Zhao. On feature decorrelation in self-supervised learning. *CoRR*, abs/2105.00470, 2021. URL https://arxiv.org/abs/2105.00470.

David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148:574–591, 1959.

Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. Understanding dimensional collapse in contrastive self-supervised learning, 2022.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, March 2017. ISSN 1091-6490. doi: 10.1073/pnas.1611835114. URL http://dx.doi.org/10.1073/pnas.1611835114.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *2013 IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013. doi: 10.1109/ICCVW.2013.77.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. URL https://arxiv.org/abs/1608.03983.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Yuning Lu, Liangjian Wen, Jianzhuang Liu, Yajing Liu, and Xinmei Tian. Self-supervision can be a good few-shot learner, 2022. URL https://arxiv.org/abs/2207.09176.

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013a. URL http://arxiv.org/abs/1301.3781.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013b. URL https://arxiv.org/abs/1301.3781.

Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL probml.ai.

Aditya Rakhecha. Understanding learning rate. https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de, 2019. [Accessed in: 10-October-2024].

Sovit Ranjan Rath. Pytorch implementation of stochastic gradient descent with warm restarts. https://debuggercafe.com/pytorch-implementation-of-stochastic-gradient-descent-with-warm-restarts, 2021. [Accessed in: 13-October-2024].

Youngmin Ro and Jin Young Choi. Autolr: Layer-wise pruning and auto-tuning of learning rates in fine-tuning of deep networks, 2021. URL https://arxiv.org/abs/2002.06048.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/ed265bc903a5a097f61d3ec064d96d2e-Paper.pdf.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015. URL http://arxiv.org/abs/1503.03832.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay, 2018. URL https://arxiv.org/abs/1803.09820.

Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf.

Yisheng Song, Ting Wang, Subrota K Mondal, and Jyoti Prakash Sahoo. A comprehensive survey of few-shot learning: Evolution, applications, challenges, and opportunities, 2022. URL https://arxiv.org/abs/2205.06743.

Nachiket Tanksale. Finding good learning rate and the one cycle policy. https://towardsdatascience.com/finding-good-learning-rate-and-the-one-cycle-policy-7159fe1c 2018. [Accessed in: 10-October-2024].

Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing*

Systems, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/68053af2923e00204c3ca7c6a3150cf7-Paper.pdf.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018. URL http://arxiv.org/abs/1807.03748.

Jesper E. van Engelen and Holger H. Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2), 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

R. Venkatesan and B. Li. *Convolutional Neural Networks in Visual Computing: A Concise Guide*. Data-enabled engineering. CRC Press, 2018. ISBN 9781498770392.

Feng Wang, Xiang Xiang, Jian Cheng, and Alan Loddon Yuille. Normface: L2 hypersphere embedding for face verification. In *Proceedings of the 25th ACM International Conference on Multimedia*, MM '17, page 1041–1049, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349062. doi: 10.1145/3123266.3123359. URL https://doi.org/10.1145/3123266.3123359.

Zhirong Wu, Yuanjun Xiong, Stella X. Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance-level discrimination. *CoRR*, abs/1805.01978, 2018. URL http://arxiv.org/abs/1805.01978.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

Xiangli Yang, Zixing Song, Irwin King, and Zenglin Xu. A survey on deep semi-supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(9):8934–8954, 2023. doi: 10.1109/TKDE.2022.3220219.

Xingyi Yang, Xuehai He, Yuxiao Liang, Yue Yang, Shanghang Zhang, and Pengtao Xie. Transfer learning or self-supervised learning? a tale of two pretraining paradigms, 2020. URL https://arxiv.org/abs/2007.04234.

Peng Ye, Yongqi Huang, Chongjun Tu, Minglei Li, Tao Chen, Tong He, and Wanli Ouyang. Partial fine-tuning: A successor to full fine-tuning for vision transformers, 2023. URL https://arxiv.org/abs/2312.15681.

M.J. Zaki, W. Meira, and W. Meira. *Data Mining and Machine Learning: Fundamental Concepts and Algorithms*. Cambridge University Press, 2020. ISBN 9781108473989. URL https://books.google.pt/books?id=qKCZxgEACAAJ.