Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik

Bachelor's Thesis

Exploiting Temporal Coherence in Video to Learn Useful Features

Mehmet Yasin Cifci
Matriculation Nº: 390743

16.06.2023

Reviewers
Prof. Dr. Klaus-Robert Müller
Prof. Dr. Grégoire Montavon

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den _____

_____

Unterschrift

Abstract

Pre-training convolutional neural networks on large-scale datasets like ImageNet is a well-established paradigm for transferring knowledge to a target task through features that have been learned in the previous training step. However, if there is video footage of the target tasks' data-generating process, we are arguing that more useful features can be learned by pre-training on that video data, which will ultimately lead to a better downstream performance on the target-task. In this thesis, we are evaluating the effectiveness of this approach by developing a self-supervised method for learning image features from video by exploiting its underlying temporal coherence and evaluating its effectiveness on various computer-vision tasks for American Sign Language hand signs. We are reporting a significant improvement in image-classification accuracy, which becomes especially apparent for small training sets, and a reduction in training time for object detection and instance segmentation.

German Translation

Zusammenfassung

Das Pre-Training von Convolutional Neural Networks auf großen Datensätzen wie ImageNet ist ein gut etabliertes Paradigma für den Transfer von Wissen auf eine Zielaufgabe durch Features, die im vorherigen Trainingsschritt gelernt wurden. Wenn es jedoch Videomaterial von dem Datengenerierenden Prozess der Zielaufgabe gibt, argumentieren wir, dass nützlichere Features durch Pre-Training auf diesen Videodaten gelernt werden können, was schließlich zu einer besseren Leistung bei der Zielaufgabe führt. In dieser Arbeit werten wir die Effektivität dieses Ansatzes aus, indem wir erst eine selbstüberwachte Methode entwickeln, die in der Lage ist Features aus Videodaten durch Ausnutzung der zugrundeliegende zeitliche Kohärenz zu erlernen und evaluieren anschließend ihre Effektivität für verschiedenen Computer-Vision Aufgaben für American Sign Language Handzeichen. Wir erreichen eine signifikante Verbesserung der Bildklassifizierungsgenauigkeit, die besonders bei kleineren Mengen an Trainingsdaten zum Vorschein kommt, und eine Reduzierung der Trainingszeit für die Objekterkennung und Instanzsegmentierung.

Contents

List of Figures

1 Introduction

Often times it is the case that a machine-learning model for a computer vision task needs to be deployed in a domain in which extensive video footage of a data-generating process is readily available while labeling that footage on a larger scale is not a viable option. Convolutional neural networks (CNNs) have been widely used in various computer vision applications, including image classification [35, 26], object detection [51] and semantic segmentation [27]. These deep learning models are known to perform exceptionally well on large-scale datasets when trained from scratch. However, training a CNN model from scratch requires a significant amount of labeled data, especially for high-dimensional data like images [12], which would make them less suitable for our use case.

Transfer learning is a technique that addresses this challenge by leveraging pre-trained CNN models on large-scale datasets like ImageNet [14] and fine-tuning them on a smaller target dataset for a specific task. In transfer learning, the pre-trained CNN model usually learns general features that are useful for a wide range of tasks, which can then be fine-tuned on the target task with less data and computation. However, while this method has been successfully applied to a wide range of use cases, its success is not guaranteed. For transfer-learning to work well, some requirements must be satisfied to a certain degree, one of which is the similarity of the source-domain and the target-domain. In the case that the target-domain does not show enough similarity with the source domain, using transfer-learning will often not lead to the desired improvement in the model's performance, or even impact it negatively [67].

1.1 Research Challenge

The goal of this paper is to research and develop a method that is able to learn useful feature-representations for a specialized domain in order to address the aforementioned shortcoming of ImageNet pretraining, for a scenario in which there is an abundance of unlabeled image data in the form of video frames, but only a small subset of these frames are labeled. To achieve that, we are using a learning paradigm called self-supervised learning. Self-supervised learning works by deriving a supervisory signal from the raw unlabelled data, which can then be used to define a learning problem. Training a model to solve that problem will allow it to learn features that can be used on downstream tasks without the need for manual labeling. We intend to use the naturally occurring temporal ordering of the frames in video data as the supervisory signal for a self-supervised learning task. By doing so, we believe that we can learn features that will be more useful for the intended downstream tasks than features learned on ImageNet because it will eliminate the problem of dissimilarity between the source and target domain.

1.2 Novelty

The idea of learning invariant representations by exploiting the temporal coherence in video has been studied in multiple works before ours. Most notably, Wiskott et al. introduced the concept of slowness (or slow features) in "Slow Feature Analysis" [63]. "SFA aims to learn invariant representations by encouraging the learned representations of nearby frames to be also close in representation space" [31]. After that, multiple papers have been published that built on top of this concept. Mobahi et al. are using temporal coherence in [46] as a regularizer to enforce the similarity of representations in terms of the L1-norm. They are training a network on a supervised task while simultaneously minimizing the contrastive regularization term for the frames of an unlabelled video. [46, 31, 4, 21, 18] are using a similar approach of incorporating temporal coherence, with [18] perhaps being the most similar approach to ours. In [18], temporal coherence is used as a supervisory signal for self-supervised pre-training of a ConvNet. Like in our method, they are also obtaining positive sample pairs by sampling frames that are temporally close to each other. Additionally, a second negative sample is sampled uniformly from outside the vicinity of the positive ones, and the contrastive loss function enforces the positive frames to be close to each other while pushing the second negative sample's representation away.

Our approach is novel in the sense that it incorporates a new type of non-contrastive loss function introduced in [65], thereby eliminating the need for finding negative samples. Instead, it relies on the principle of redundancy reduction. In addition to that, we introduce a more general framework for defining the distribution from which the positive pair is sampled and investigate the effect of using different distributions.

1.3 Anticipated Impact

While much of the current research is focused on learning general-purpose representations on large-scale datasets like ImageNet [65, 28, 8], we see a need for methods that provide an alternative for specialized domains where such methods can not provide the desired improvement. With this work, we intend to explore the possibility of obtaining more specialized representations without creating an additional overhead through labeling, and as a result, we aim to provide a framework to easily learn representations from video while simultaneously demonstrating the usefulness of our approach for the specified use case. The resulting codebase is made available at `https://github.com/myasincifci/bachelor_thesis_code`.

1.4 Outline

This paper is structured into several chapters, each with a specific focus. Chapter 2 provides an introduction to the basic concepts and terminology relevant to our research, giving an overview of the field and commonly used methods. In Chapter 3, we summarize previous work on the subject, focusing on two papers in particular that are important for our work. Chapter 4 presents our proposed method, explaining the underlying idea and implementation details. In Chapter 5, we evaluate the effectiveness of our method on a dataset we have specifically created for this purpose,

comparing the results to a baseline. Chapter 6 analyzes the representations obtained using different methods, using LRP to explain the network's decisions. Chapter 7 conducts ablation studies to investigate the effects of limiting the sampling range and training on randomly initialized weights and discusses the implications of our findings. Finally, Chapter 8 provides a comprehensive discussion of our research findings, examining the strengths and limitations of our approach and its potential applications.

2 Prerequisites

This chapter is intended to provide a review of the necessary concepts and techniques that are required to understand the subsequent chapters. It will cover the fundamental learning paradigms, network architectures, and theories used in this work.

2.1 Learning Paradigms

There are several learning paradigms in machine learning that provide different approaches and frameworks for algorithms to learn from data. These paradigms include (but are not limited to) supervised learning and unsupervised learning. In the following, we will provide an introduction to the two.

2.1.1 Supervised Learning

In supervised learning, the learning-algorithm is provided a label together with each training sample and tries to learn a function that maps the samples to their corresponding labels [54]. Formally the supervised learning task can be defined in the following way:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell \left(f_\theta(\mathbf{x_i}), \mathbf{t_i} \right) \tag{2.1}$$

Where $\mathbf{X} = [\mathbf{x_1}, \dots, \mathbf{x_N}]$ are a set of samples with their corresponding set of labels $\mathbf{T} = [\mathbf{t_1}, \dots, \mathbf{t_N}]$. The goal is to learn the parameters θ of the function $f_\theta(\mathbf{x_i})$ such that a loss function $\ell(\mathbf{y_i}, \mathbf{t_i})$ is minimized, which is a measure of the discrepancy between the predicted label and the ground truth label.

Supervised learning has been the predominant paradigm in computer vision, and the availability of large-scale labeled image datasets has been a decisive factor in the recent success of deep learning in the field. However, the need for manual labeling has become a major bottleneck for the advancement of these systems [38], which is why recently, a lot of research is focussed on unsupervised or self-supervised learning.

2.1.2 Unsupervised/Self-Supervised Learning

Unlike supervised learning, **unsupervised learning** does not require labeled data to learn. Instead, unsupervised learning algorithms seek to learn useful patterns from the data itself [19]. A well-known example of an unsupervised learning task is clustering, where the goal is to group data points into clusters based on the similarity of their features without having to assign specific labels.

Self-supervised learning (SSL) is a term that emerged in recent years and is used to describe a learning paradigm that can be considered a specialized subset of unsupervised learning. As for unsupervised learning, no labels are explicitly given to the learning algorithm. Instead, a supervisory signal is derived from the data itself [38] to automatically generate pseudo-labels. These are then used to define what is called a **pre-text task**. Solving this task is rarely the main objective (thus the name), which is why the self-supervised training is usually followed by a supervised fine-tuning stage [32]. The goal is to transfer the knowledge via the learned representations to the so-called **downstream-task** (more on that in sections 2.2 and 2.3). SSL algorithms are commonly divided into two categories [50]:

- **Contrastive:** Methods that rely on contrastive learning use pairs of samples with a positive or negative relationship to learn representations that are similar or dissimilar [11]. Contrastive SSL algorithms for vision include [28, 8, 44].
- **Non-contrastive:** These methods only rely on positive relationships between samples [50]. Non-contrastive SSL algorithms for vision include [65, 23, 10]. Our algorithm also falls into this category as it is based on [65].

SSL has proven itself tremendously successful in the field of natural language processing (NLP), where it is most famously used for the pre-training of language models [15, 7]. For language modeling, given an incomplete sequence of words, the task is to complete that sequence by predicting the missing words. In that case, self-supervised learning presents itself as a natural choice because large amounts of unlabelled text are readily available, so the pseudo-labels (labels automatically generated for SSL) can be generated by taking complete text sequences and masking certain parts. Although unlabelled data that can be used for SSL for vision-based tasks is also available in large amounts in the form of images and video, SSL has not yet brought the amount of improvement to computer vision that it has to NLP [38]. Nevertheless, many of the aforementioned algorithms managed to achieve results that are on par with their supervised counterparts. In the context of this work, we are going to explore the possibility of exploiting the temporal coherence in video as a supervisory signal to learn representations with a non-contrastive SSL algorithm.

2.2 Transfer Learning

In the field of machine learning, transfer learning refers to the process of enhancing learning in a novel task by leveraging the knowledge obtained from a related task that has already been learned [49]. This phenomenon has been widely observed in humans, where it is believed that the basic foundation of human intelligence is rooted in "common sense" knowledge, which defines a model of the world around us and to acquire specialized knowledge, relevant components of this foundational knowledge are transferred to the new domain [38]. The setting of transfer learning involves a **source-task**, where initial knowledge is obtained, and a **target-task**, where that knowledge is subsequently transferred to. The success of the transfer can then be assessed based on performance in the target-task, and can be evaluated using three key measures [49]:

- The performance on the target task at the beginning of the training only using the knowledge transferred from the source-task (higher start on the graph in Figure 2.1).

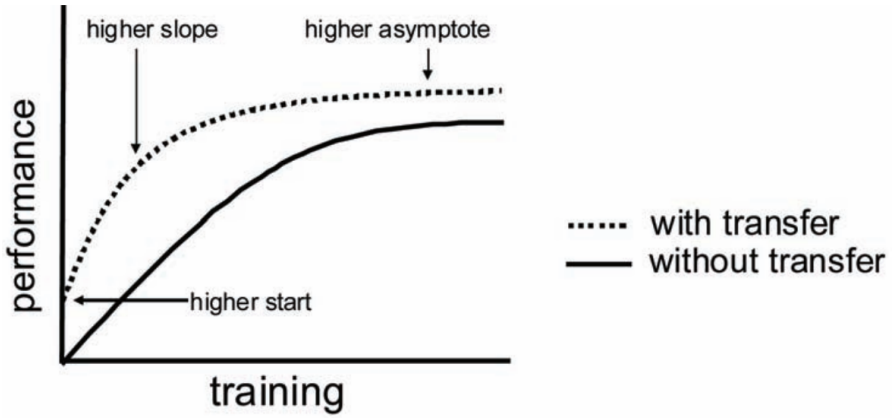


Figure 2.1: Example of training curves demonstrating successful knowledge transfer with improvements regarding the three measures highlighted (taken from [49]).

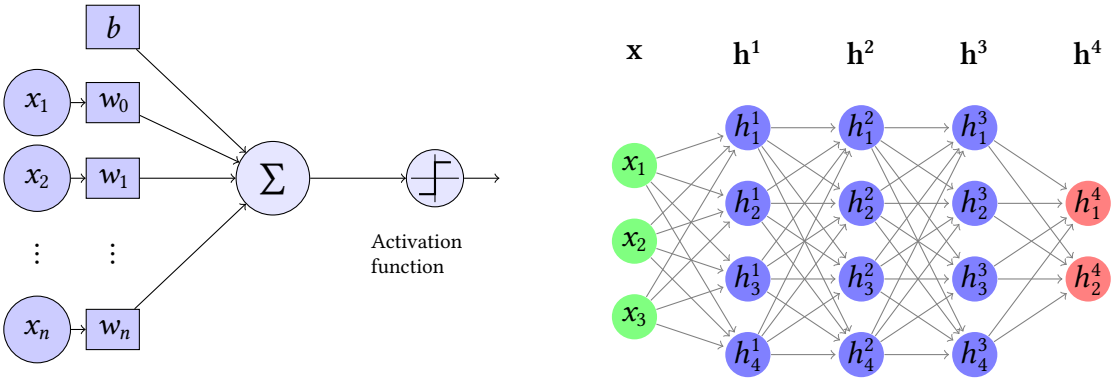
- The rate at which the target-task is learned in comparison to no transfer (higher slope on the graph in Figure 2.1).
- The final performance achieved in the target task (higher asymptote on the graph in Figure 2.1).

The goal is to have an improvement with respect to at least one of these measures. In the case that the performance has decreased compared to the ignorant model (model without transfer), **negative transfer** has occurred. When we evaluate the performance of our method in Chapter 5, these three measures have to be considered. Transfer learning for CNNs is realized by transferring the features that have been learned for the source task to a target task which makes it is closely related to the concept of feature learning that will be introduced in the following section.

2.3 Representation/Feature Learning

When we make a prediction, we need to give the prediction model an input vector $\mathbf{x}$. This vector is what we call a **representation** of our data, and the components of this representation-vector are called **features** [25]. How difficult or easy it is to solve a particular task can strongly depend on how we choose to represent the information we have at hand [5, 20]. Especially the performance of shallow algorithms like logistic regression is known to depend heavily on the quality of the representations that are used [58].

The area concerned with extracting appropriate features for a subsequent machine-learning task is called feature engineering and is an integral part of many machine-learning pipelines. While there are algorithms (e.g., SIFT [42] and HOG [13]) for extracting image features that have been successfully applied to computer vision tasks [68, 13], these methods have been mostly replaced by deep learning-based ones for more complex image-based machine learning tasks such as large-scale image classification. Their shortcoming is due to the large number of possible features that can be found in image data which makes it infeasible to handcraft these features. **Representation Learning** presents itself as a possible solution to that problem. The idea is to use machine-learning



(a) Perceptron/Neuron [1]

(b) Multi-Layer Perceptron [2]

not only for the prediction task but also use it in an intermediary step to discover representations that will be useful for the subsequent prediction task [20]. This approach has proven itself very successful, especially in the domain of computer-vison in the form of deep convolutional neural networks, which we will give an introduction to, in the following sections. Representations produced by representation learning have been shown to be especially useful in combination with transfer learning as they are able to capture a multitude of explanatory factors of the data that are applicable to a wider range of tasks [5]. For that reason, it is also desirable to learn general-purpose representations on large-scale datasets like ImageNet.

2.4 Deep Neural Networks for Feature Extraction

This section aims to give an introduction to how neural networks and convolutional neural networks work and to provide the background necessary to understand how they can be used to learn features from image data.

2.4.1 Multi-Layer Perceptron

In 1958 Frank Rosenblatt introduced the perceptron algorithm [52], which can be considered the first predecessor of modern-day artificial neural networks. Artificial neural networks (ANNs) are a class of machine learning models that are loosely modeled after the brain. They consist of nodes called artificial neurons (Fig. 2.2a) which are meant to simulate a single neuron by taking a linear combination of the input signals $\mathbf{x_i}$ weighted by some learnable parameters $\mathbf{w_i}$. The resulting signal is then fed into a non-linear activation function like ReLU or Sigmoid that fires upon reaching a certain threshold, similar to how a real neuron will behave. This architecture alone is able to do binary classification on an arbitrary input vector but is limited by the fact that it can only learn to represent linearly separable functions [60]. This limitation was overcome with the introduction of the multi-layer perceptron, which is depicted in Figure 2.2b.

[1] Figure based on answer from: https://tex.stackexchange.com/questions/104334/tikz-diagram-of-a-perceptron

[2] Figure based on answer from: https://tex.stackexchange.com/questions/362238/multiple-hidden-layers-in-neural-network-diagram

Like the name states, they consist of two or more layers (with the layers apart from input and output being called **hidden layers**), where each layer $\mathbf{h}^{\mathrm{i}}$ is an array of neurons/perceptrons. The input signal $\mathbf{x}$ is fed into the first of multiple hidden layers and propagates through the network in a feed-forward manner. Each layer of neurons represents a linear transformation followed by a non-linear activation function f and can be expressed by the following formula:

$$\mathbf{h}^i = f(\mathbf{W}^i \cdot \mathbf{h}^{i-1} + \mathbf{b}^i) \tag{2.2}$$

where $\mathbf{W}^i$ and $\mathbf{b}^i$ are learnable parameters and can be optimized by backpropagation [53] in combination with a gradient-based optimizer like stochastic gradient descent (SGD) or similar. Having multiple layers enables the network to learn non-linear decision boundaries [60]. It has been shown in [30] that, in theory, a shallow MLP-based architecture with only one hidden layer has the ability to approximate any function arbitrarily well, but while that is true in theory, actually learning such a function is often deemed impractical, necessitating the development of deeper networks and more sophisticated architectures like convolutional neural networks.

2.4.2 Convolutional Neural Networks

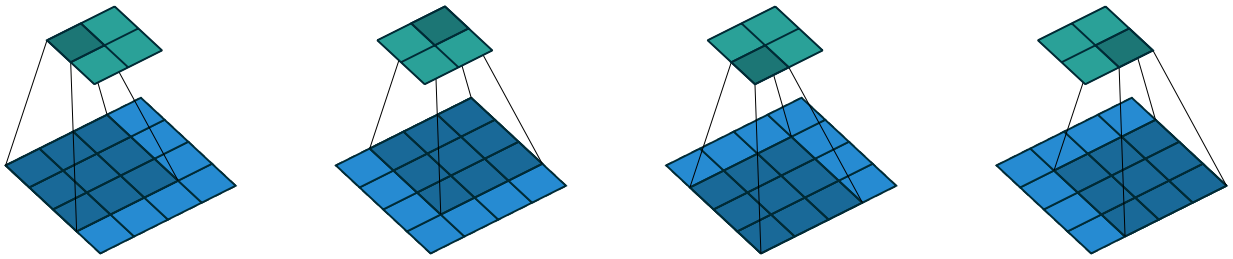
The idea behind convolutional neural networks (ConvNets) was first introduced by Fukushima et al. under the name of Neocognitron [17], who proposed an architecture that mimicked the visual system by utilizing convolution and pooling operations. Later the term convolutional (neural) network was introduced by LeCun et al. in [39], who, for the first time, trained a ConvNet using backpropagation. Later on, ConvNets gained popularity after they beat the ILSVRC [1] competition with AlexNet [35] for the first time, achieving state-of-the-art on ImageNet [14]. After that, ConvNets have dominated the competition as well as many vision-based machine learning tasks for many years to come. Apart from classification, they have notably produced state-of-the-art results for image-detection[51] as well as instance-segmentation [27]. In the following, we will have a look at the underlying mechanisms of a ConvNet to understand what makes them suitable for computer vision. The two main building blocks of ConvNets are convolution and pooling layers. To understand how they are used in ConvNets, let us have a look at them in detail.

2.4.3 Convolution

The function of the convolution layer is to extract features from an input. How exactly that is achieved can be understood by looking at the definition of convolution, more specifically, two-dimensional discrete convolution.

When working with image data, the image is usually represented as a $C \times H \times W$ tensor. C is the number of channels of the image, which is most commonly three for RGB or one for grayscale images. H and W are the number of rows and columns of a matrix containing the pixel-luminance values of each channel. For the sake of simplicity, let us limit ourselves to the case of single-channel images for now. Then the convolution operation can be intuitively thought of as sliding a matrix $\mathbf{W}$ across each position of the matrix $\mathbf{x}$ representing the input image and at each step, taking a sum of the current region, weighted by $\mathbf{W}$ (as illustrated in Figure 2.3).

[1] https://www.image-net.org/challenges/LSVRC/



(a) (b) (c) (d)

Figure 2.3: Step-by-step illustration of how the filter **W** (dark blue) moves along the input matrix (light blue) and produces the feature map (green) (taken from [16]).

The underlying idea is to learn the weights of **W** such that it produces a high activation when the corresponding pixels represent a certain feature and a low activation if not. The produced output matrix consisting of the activations will then be a mapping of the input matrix where the occurrence of a feature is highlighted by a high activation. Accordingly, the output of the convolution operation is called a **feature map**, and the weight matrix is called the **filter** or **kernel**.

To gain an intuition for the process, let's have a look at the following filter in equation 2.3:

$$\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \odot \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{2.3}$$

The left matrix represents the filter and the right one represents an arbitrary patch of the input image. Assuming that we are operating on a matrix where the entries are luminance values, it is easy to see that this filter will produce the highest activation when the pixels a and c are dark (close to zero) and b and d are bright. This corresponds to an edge transitioning from dark to bright in the input image, meaning the result of applying this filter to an input image will be a feature map of the edges found in the image. In general, it can be observed that these types of simple features, like edges or textures, are learned in the shallower layers of the network, and the features get more sophisticated (representing parts of objects, etc.) with increasing layer depth [66] because they take the features extracted in the previous layer as their input. This can be seen in Figure 2.4.

Figure 2.4: Illustratuin of the features from various layers of a ConvNet (taken from [20]).

Definition

For a feature-map $\mathbf{y}$ each entry $\mathbf{y_{i,j}}$ is computed as shown in equation 2.4 where $\mathbf{x}$ is the $n \times m$ input-matrix representing a single channel image, and $\mathbf{W}$ is the $k_1 \times k_2$ filter matrix. s is the stride, which defines by how much the filter matrix is shifted at each step (the example in Figure 2.3 has stride=1)

$$\mathbf{y_{i,j}} = \sum_{l=0}^{k_1-1} \sum_{m=0}^{k_2-1} \mathbf{x}_{si+l,sj+m} \mathbf{W}_{l,m} \tag{2.4}$$

The definition is extended to inputs with c channels by applying 2.4 to each channel, where a separate filter is learned for each one, and then summing over the channel dimension of the resulting feature-maps (as defined in 2.5).

$$\mathbf{y_{i,j}} = \sum_{i=0}^{c-1} \left[\sum_{l=0}^{k_1-1} \sum_{m=0}^{k_2-1} \mathbf{x}_{c,si+l,sj+m} \mathbf{W}_{c,l,m} \right] \tag{2.5}$$

After the convolution operation follows an activation function that will filter out activations that do not pass a certain threshold, and the result will then be passed to the next layer.

2.4.4 Pooling/Subsampling

The pooling operation (also referred to as subsampling) performs a local summarization of the feature-map and thereby reduces its dimensionality. This has the effect that the output becomes invariant to small translations in the input [20]. As for the convolution operation, a kernel needs

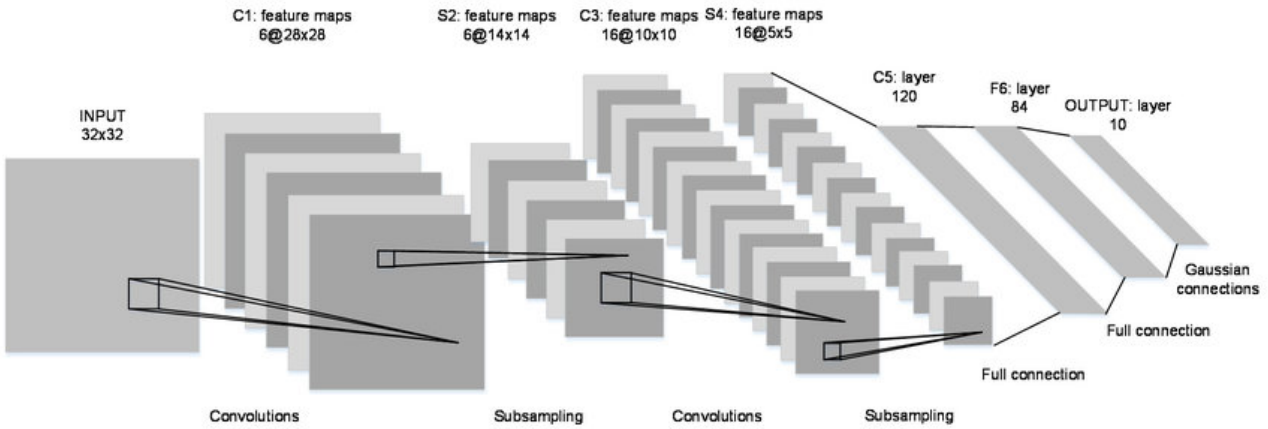


Figure 2.5: LeNet5 architecture (Figure taken from [39]).

to be defined together with a stride (often, the stride is equal to the kernel size) as parameters that define the size and the positions of the region to summarize. But different from convolution, there are no learnable parameters. Instead, the type of pooling operation defines the kernel that performs the summarization. The two most common types are max-pooling and average-pooling, where max pooling returns the highest value in each region and average-pooling returns the mean value.

2.4.5 Architecture

In Figure 2.5, we can see the LeNet5 architecture from [39], which demonstrates the typical structure of a CNN. Convolution and pooling/subsampling layers are applied in an alternating manner, followed by multiple fully connected layers (layers of fully connected neurons, like in equation 2.2). The receptive field for an activation in a feature-map is the area of pixels in the input image that it is connected to [39]. As a consequence, the receptive field for an activation grows larger the deeper it is positioned in the network [20]. Because each layer gets the feature-map of the previous one as an input, the features of each layer are a composition of the features learned by the previous layers. This hierarchical structure of features allows for consequent layers to compose more complex features from the ones passed to them by the previous ones.

2.4.6 Convolutional Neural Networks from the Perspective of Representation Learning

If we quickly recapitulate on section 2.3, the goal of representation-learning is to discover representations that allow to make better predictions. Through the lens of representation learning, a ConvNet can be seen as consisting of two stages. The first stage is the fully convolutional part which is extracting features from the input image and essentially learning representations that are then passed to the second fully-connected stage, which is making a prediction based on the representations provided by the first stage. That is why the fully convolutional part of a ConvNet is also referred to as a feature extractor, and the subsequent fully-connected layer is called the prediction

head.

2.4.7 ResNet

In comparison to classical CNNs, the ResNet [26] architecture implements two additional components that allow the architecture to scale beyond what is possible with the standard building blocks, in terms of network depth:

Batch normalization layers are added in between the convolution layers and the activation function to normalize the activations along the batch dimension. This is addressing the vanishing/exploding gradient problem [29] that occurs during the computation of the gradient via backpropagation and is amplified with increasing network depth. It is making training more stable [55] and ultimately allows very deep networks to converge [26].

Shortcut connections aim to tackle the issue of diminishing training accuracy, which occurs when the network dept is increased after the accuracy reaches saturation [26]. They are allowing intermediary activations to skip layers, as depicted in Figure 2.6, by bypassing them (illustrated by the arrow) and adding the activations directly to the activations produced by the bypassed layers.

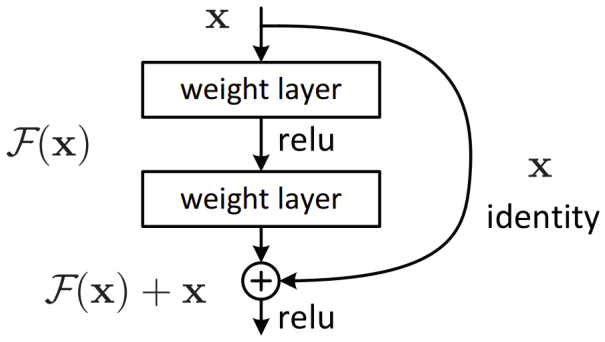


Figure 2.6: Residual block (figure taken from [26]).

The intuition behind it is that additional layers can, in the worst case, more easily learn an identity mapping by learning zero weights for the layers between the skip connection. This should allow the network to perform at least as well as its counterpart that is missing the additional layers [26].

By using this architecture, He et al. have been able to train ConvNets that are much deeper than previous ones and achieved the state-of-the-art winning the ILSVRC competition in 2015. The architecture for a 34 layers-deep ResNet (Resnet-34) is depicted in Figure 2.7 where each convolution is followed by a batch-normalization layer and a ReLU activation function with skip connections between blocks of two convolutions.

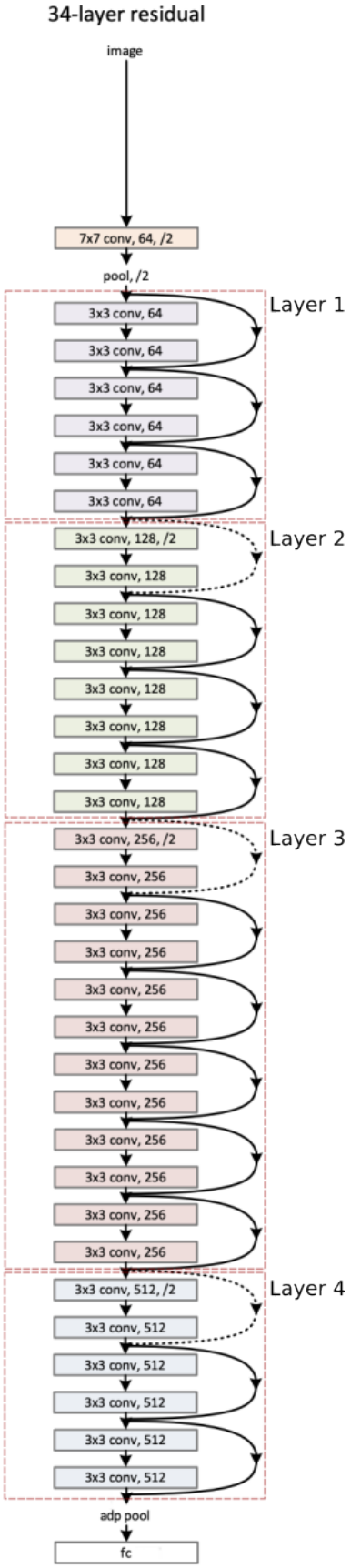


Figure 2.7: Network graph of ResNet-34 architecture (figure taken from [26] and modified).

3 Previous Works

3.1 Slow Feature Analysis

In [63], Wiskott et al. argue that for the temporal sequences of frames in a video, the low-level primary sensory signals, like the value of a single pixel, will vary very quickly in comparison to high-level features, like the position of an object in the frame, which is not expected to change very abruptly. Therefore one should find a function that maps the primary sensory input signal to a signal that only varies slowly in the latent space to learn high-level features. Further research on learning representations from video has been largely based on this idea of slowness [46, 31, 4, 21, 18]. It is applied to deep learning by enforcing representations of frames that are close in terms of the temporal structure of the video, to be also close in representation space. This is done by either adding a regularization term that enforces this property while training on a supervised task [46, 31] or in a self-supervised feature learning/pretraining setting [4, 21, 18].

Apart from SFA-based methods, there are also other approaches that aim to learn image representations from video by exploiting its temporal structure, one of which is to learn to predict the ordering of sequences of video frames [45, 40].

In the following, we will have a deeper look at [46], which has largely influenced our method, and also introduce BarlowTwins [65], a self-supervised algorithm for learning image features that we have adapted for our algorithm to incorporate the slowness prior.

3.2 Deep Learning from Temporal Coherence in Video

In "Deep Learning from Temporal Coherence in Video"[46], Mobahi et al. introduce an algorithm that is able to take advantage of the temporal-coherence in video to improve the accuracy of a deep learning model on a classification task.

It motivates the idea of learning similar representations for consecutive frames by making the assumption that consecutive frames in a video have a high probability of containing the same object or scene while being slightly transformed versions of each other due to movement in the video. The method aims to make the learned representations invariant to said translations by defining a coherency regularization term that incorporates the slowness assumption by encouraging representations of consecutive frames also to be close. The regularization term (equation 3.1) is added to the loss function of a supervised task, making the learned representations more robust to the translations naturally occurring in the video. The regularization term is defined as follows:

$$L_{coh}(\boldsymbol{\theta}, \boldsymbol{x}_1, \boldsymbol{x}_2) = \begin{cases} \|z_\theta(\boldsymbol{x}_1) - z_\theta(\boldsymbol{x}_2)\|_1, & \text{if } \boldsymbol{x}_1, \boldsymbol{x}_2 \text{ consecutive} \\ \max(0, \delta - \|z_\theta(\boldsymbol{x}_1) - z_\theta(\boldsymbol{x}_2)\|_1), & \text{otherwise} \end{cases} \tag{3.1}$$

where $\boldsymbol{\theta}$ are the learnable parameters, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are two input frames, and $z_\theta(\boldsymbol{x}_1)$ and $z_\theta(\boldsymbol{x}_2)$ are their respective representations produced by the network. This function has the effect that the

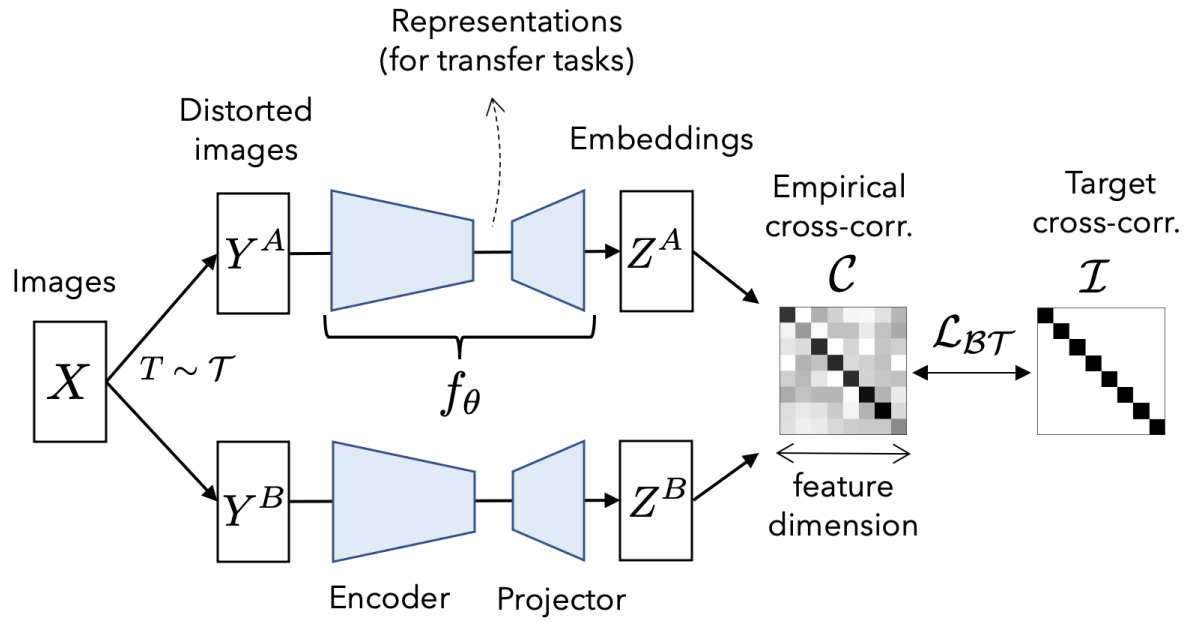


Figure 3.1: Barlow Twins architecture. (Figure taken from [65]).

produced representations are pulled together (in terms of the L1 norm) if the frames $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are consecutive in the ordering of the video. Otherwise, they are pushed apart up to a margin δ.

Using this method, Mobahi et al. were able to achieve an improvement in testing accuracy of as much as 20,76% in supervised image classification on the COIL100 dataset [48] in comparison to training without the regularization term.

3.3 Barlow Twins

In "Barlow Twins: Self-Supervised Learning via Redundancy Reduction" [65], Zbontar et al. introduce a novel, non-contrastive architecture for self-supervised learning that is based on the principle of redundancy reduction. In the following chapter, we will provide a detailed description of our method, which is based on the aforementioned architecture. As such, it is beneficial to provide an overview of Barlow Twins to better understand the workings of our approach.

The idea behind the redundancy reduction principle is to encode data points that contain highly redundant information such that only statistically independent pieces of information are preserved [3]. Barlow Twins incorporates this principle in its loss function by minimizing statistical dependence within the embeddings it learns. Figure 3.1 depicts the architecture. Similar to other self-supervised methods for learning image representations, Barlow Twins operates by maximizing the similarity between the representations of two transformed versions of the same image. First, a batch of images X are sampled from a dataset. Then two transformed versions Y^A and Y^B are created by applying transformations that are randomly sampled from a set of possible transformations

$\mathcal{T}$. These distorted versions of X are then fed into two identical networks that share a set of parameters θ, commonly referred to as Siamese Networks [24]. The networks consist of an encoder- and a decoder part. The encoder is a fully convolutional neural network that is used to extract features from the input images, as described in the previous chapter. The high-level representations from the last layer of the encoders are then passed to the projectors, which produce a batch of embedding vectors Z^A and Z^B each, with $Z^A, Z^B \in \mathbb{R}^{N,D}$, where N is the number of samples in a batch, and D is the dimensionality of the embeddings. After normalizing them along the batch dimension, they are then fed into the loss function, which can be seen in equation 3.2.

$$\mathcal{L}_{\mathcal{BT}} \triangleq \underbrace{\sum_i (1 - \mathcal{C}_{ii})^2}_{\text{invariance term}} + \lambda \underbrace{\sum_i \sum_{j \neq i} \mathcal{C}_{ij}{}^2}_{\text{redundancy reduction term}} \tag{3.2}$$

It corresponds to the squared difference between the cross-correlation matrix $\mathcal{C} = (Z^A)^T Z^B$ and the identity matrix. By enforcing the diagonal terms to be close to one, the "invariance term" encourages the entries of the embeddings that represent the same feature to be correlated. This forces the network to learn representations that are equal between the distorted views of an image, therefore making the network invariant to these distortions. Similarly, the "redundancy reduction term", by encouraging off-diagonal entries to be close to zero, decorrelates the features between each other. As a result, the features of the embeddings can not represent the same information, making them non-redundant. The parameter λ is weighting the importance of the redundancy reduction term compared to the invariance term.

Representations learned by Barlow Twins produce competitive results with competing self-supervised methods in linear and semi-supervised evaluation on ImageNet classification, as well as transfer learning on various classification, object detection, and instance segmentation tasks. It is also able to work with relatively small batch sizes (256) in comparison to competing methods [65].

4 Implementation

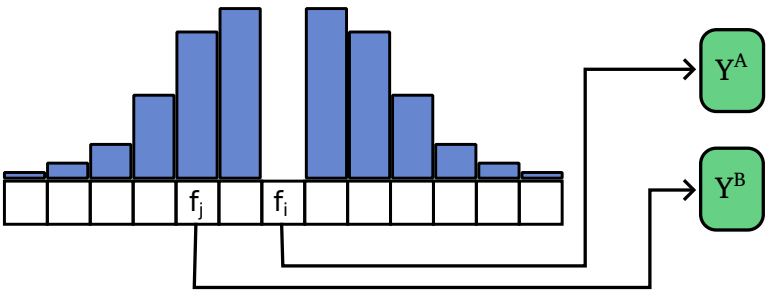


Figure 4.1: Illustration of the sampling process for our algorithm. For every frame f_i added to the batch $\mathbf{Y^A}$, a corresponding frame f_j is added to $\mathbf{Y^B}$ according to the probability distribution depicted by the blue bars.

4.1 Idea

Following the idea of using temporal coherence as a supervisory signal over unlabelled video data, we have developed a self-supervised method that is able to utilize the unlabelled data to learn useful representations from it, similar to [46]. We are extending the assumption made in [46] that only consecutive frames are likely to contain the same class of object and therefore should contribute to the coherency term, to a more generalized version. Instead, for each frame, we assign every other frame in the video a certain probability of containing the same object. The assumption in [46] can then be seen as a specialized version of ours, in which the frames before and after each frame are assigned a probability of 100%, and every other frame is assigned a probability of 0%. The selection of these probabilities serves as a method of incorporating prior knowledge about the structure of the video and can be treated as tuning hyperparameters.

To construct our method, we are making use of this assumption together with the architecture introduced in Barlow Twins. We are using the same siamese network [6] architecture and loss function as in the Barlow Twins paper (which can be seen in Figure 3.1) with a different way of sampling the input images: Instead of applying transformations manually to produce the two distorted batches $\mathbf{Y^A}$ and $\mathbf{Y^B}$, as it is done in the Barlow Twins algorithm, we make use of the previous assumption to obtain the distorted images. We create $\mathbf{Y^A}$ by randomly sampling a batch of images from the video. Then, for each $f_i \in \mathbf{Y^A}$ a slightly distorted version f_j is obtained by sampling it from the video, according to the provided probability distribution (see Figure 5.1). The second frame is chosen such that it has a high probability of containing the same object while still being subject to some transformations induced by the movement in the video. Then by minimizing the loss function, we aim to learn representations that are invariant to these transformations.

We have chosen to use the Barlow Twins architecture over the one used in [46] and other possible architectures because of its non-contrastive loss function. Because it is non-contrastive, it does not require negative samples, which allows it to work well with comparably small batch sizes [65]. This results in reduced hardware requirements with the added benefit of being easier to implement.

4.2 Method

Let $\mathbf{F}$ be the ordered set of all frames in the unlabelled video where f_i is the ith frame with $i \in \{0, \dots, I-1\}$. We assume that for every frame $f_i \in \mathbf{F}$, there exists a function P_{f_i} that is mapping f_i, the reference frame, onto a probability for all $f_j \in \mathbf{F}$. These probabilities represent the likelihood of f_j containing the same object as f_i. Then, to sample a frame similar to f_i, one needs to sample from a probability density function p_{f_i} constructed according to P_{f_i}.

Our method can take an arbitrary probability density function (pdf) of the form $p_{f_i} : \mathbf{F} \mapsto \mathbf{F}$, which is then used for sampling during training. We hypothesize that an ideal pdf $p^*_{f_i}$ would be chosen such that it maps each frame only onto frames that represent the same object class i.e. have the same class label. The task of finding such a function is, however, not feasible because it corresponds to the task of labeling the video, which defeats the purpose of our algorithm. Instead, we can try to approximate $p^*_{f_i}$ by exploiting the underlying temporal structure of the video. We have defined and implemented two such functions, the first of which can be seen in equation 4.1.

$$p^{uni}_{f_i,\tau}(f_j) = \begin{cases} \dfrac{1}{\min\{i, \tau\} + \min\{I - i - 1, \tau\}}, & \text{for } i - \tau \leq j \leq i + \tau \text{ and } i \neq j \\ 0, & \text{else} \end{cases} \tag{4.1}$$

Based on the assumption that frames in a certain proximity of f_i likely contain the same object, $p^{uni}_{f_i,\tau}(f_j)$ constructs a uniform distribution around the reference frame f_i within a certain proximity, which is denoted by the parameter τ. This parameter can be tuned according to the specifics of the video e.g., the framerate. $\tau = 1$ corresponds to the setting in [46], only sampling frames adjacent to f_i. Building on this idea, we define a second function:

$$p^{nml}_{f_i,\sigma}(f_j) = \frac{n(j, i, \sigma) \cdot \mathbb{1}_{[\mathrm{i}\neq\mathrm{j}]}}{\sum_{k=0}^{I-1} n(k, i, \sigma) \cdot \mathbb{1}_{[\mathrm{i}\neq\mathrm{k}]}} \tag{4.2}$$

where n is the normal distribution defined as:

$$n(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \tag{4.3}$$

Again, the main idea is to choose a frame in the vicinity of f_i, but additionally, we make the assumption that the probability decreases with increasing distance to f_i. How quickly it decreases can be controlled by the parameter σ.

After establishing the way of sampling the training frames, we will now describe the training procedure itself. As the architecture for the feature extractor, we are using a ResNet-34. Although using a deeper architecture (e.g., ResNet-50) results in an increase in performance, we have found ResNet-34 to be a good balance between model complexity and computational cost. The weights of

the feature extractor are initialized with weights that have been pre-trained on ImageNet [14], as we have found this to be highly beneficial (see section 7.2 of the ablations chapter for a comparison). After initialization, the projection head is added to the network. It consists of three linear layers with D neurons each, where D is the dimensionality of the embeddings, and the first two linear layers are followed by a batch-norm layer and ReLU each. We are then training the network using stochastic gradient descent with weight decay. The pseudo-code for the training process can be seen in listing 4.1. It is a modified version of the algorithm introduced in [65] where we have integrated the sampling method proposed by us:

```
# Produces samples for Y^B according to provided pdf
def sample_YB(indices_YA, pdf, loader):

    indices_YB = pdf(indices_YA)
    images_YB = loader.get_item(indices_YB)

    return indices_YB, images_YB


# model: ResNet34-Backbone + Projetion Head
# N: batch size
# D: dimensionality of the embeddings
# eye(D): creates DxD identity matrix
# loader: pytorch style data-loader
for YA in loader:

    # sample Y^B from Y^A
    indices_YA, images_YA = YA
    indices_YB, images_YB = sample_YB(indices_YA, p_uni, loader)

    # compute embeddings
    z_A = model(images_YA) # NxD
    z_B = model(images_YB) # NxD

    # normalize
    z_A_norm = (z_A - z_A.mean(0)) / z_A.std(0) # NxD
    z_B_norm = (z_B - z_B.mean(0)) / z_B.std(0) # NxD

    # cross-correlation matrix
    C = z_A_norm.T @ z_B_norm / N # DxD

    # compute loss
    invariance_term = ((C*eye(D) - eye(D))**2).sum()
    redund_red_term = lambda * (((-eye(D) + 1)*C)**2).sum()
    loss = invariance_term + redund_red_term

    # optimization step
    loss.backward()
    optimizer.step()
```

Listing 4.1: Pytorch-like pseudo code for trainig (based on Barlow Twins pseudo-code [65]).

5 Evaluation

For the evaluation of our method, we will employ a benchmarking suite similar to the one proposed in [22] for the benchmarking of self-supervised models. We are evaluating the representations learned by our method on the tasks of image-classification, object-detection, and instance-segmentation. As a baseline, we will compare the results produced by our method to the results obtained from a model that has been pre-trained on the ImageNet-1K [1] dataset on a supervised image-classification task.

5.1 Dataset

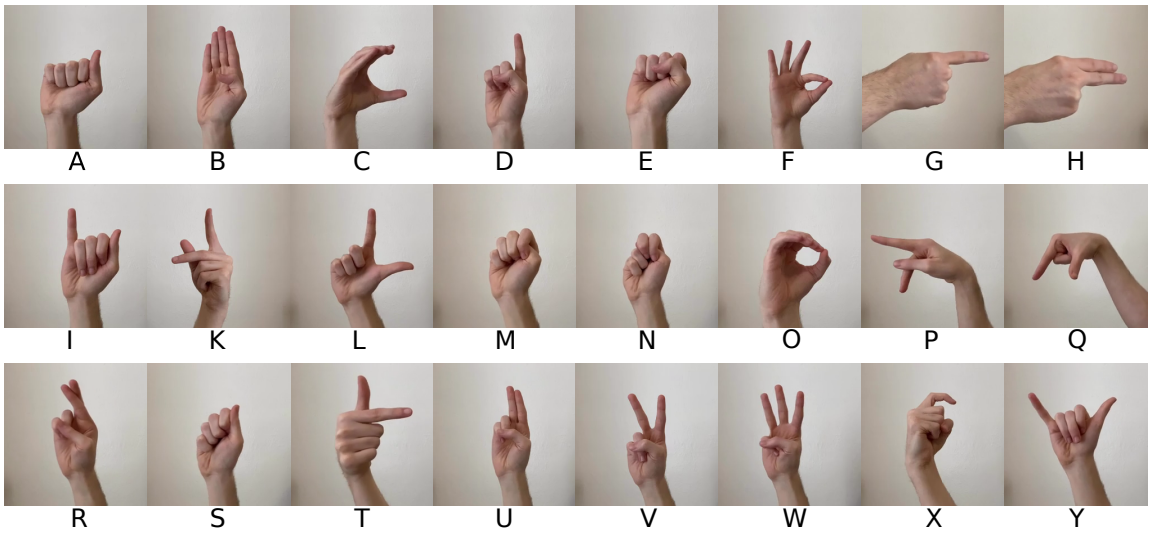


Figure 5.1: One sample of each class of the AMA training dataset.

As we have stated in Chapter 1, our objective is to provide an alternative to ImageNet features for specialized domains to which they might not transfer well. Therefore we have decided to create a dataset that fits our objective well rather than using a publicly available dataset like COIL100 [48] or NORB [37], which have been used in previous works and usually cover a broader range of classes and concepts that are more likely to coincide with the ones found in ImageNet.

We are going to evaluate the performance of our method on the task of recognizing American Manual Alphabet (AMA) hand signs. The AMA is a set of hand signs used in American Sign Language to spell out words [61]. It has a sign for each of the 26 letters of the alphabet, with 24 of these being static and only two (j and z) being non-static, meaning they require some kind of hand movement. Consequently, the number of classes in our dataset is limited to 24.

[1] ImageNet dataset with only a subset (1000) of the original classes.

We have recorded video footage in which each of the 24 hand signs is demonstrated, one after the other, with each figure being subject to some movement. The video has a total duration of 6:15 minutes and is recorded with a frame rate of 10 fps at a resolution of 128 x 128 pixels, resulting in a total of 3757 frames. The raw frames of the video are then extracted and saved in the form of tuples ($\mathbf{f_i}$, **i**) where $\mathbf{f_i}$ is the ith frame of the video and **i** its index. The complete set of frames is used for the self-supervised pre-training stage. For the subsequent fine-tuning stage, a training set is created by annotating the frames with their respective class-labels. Because we want to evaluate how the different methods perform for a varying amount of data available, we also create multiple training sub-sets with 20, 10, 5, and 1 samples per class. For testing, we have recorded five images of each class in the same setting in which the video was taken, which gives us a total of 120 test samples. Additionally, we have annotated the dataset containing five samples per class with segmentation masks for the detection/segmentation task using the cvat [57] annotation tool.

5.2 Linear Evaluation

To assess the usefulness of the representations on the task of image-classification we are performing **linear evaluation** (also referred to as linear-probing or frozen-protocol in the literature). This is a standard method for evaluating representations [65, 28, 23] in which a linear classifier (e.g., logistic regression, linear kernel svm) is trained for the downstream task on the fixed representations produced by the feature extractor. It has been shown that there is a linear correlation between the linear-evaluation performance and the performance achievable by fine-tuning the complete network (semi-supervised evaluation) [9]. Also, considering the fact that only the linear classifier needs to be tuned, it is a good evaluation metric that is fairly cheap in terms of compute (which makes it even feasible to be used as an online evaluation method).

Following this protocol, first we are pre-training a ResNet-34 backbone with a projection head with embedding dimensionality $D = 1024$ using our algorithm (we will refer to this model as "Tempo" in the following). We are training for 10 epochs at a learning rate of 0.001, a weight decay of 0.001 and $\lambda = 0.001$ sampling from a uniform distribution with proximity parameter $\tau = 30$. As described in [22], we then extract the representations from both networks, Tempo and Baseline, and train a linear single-layer neural network on top of them. It is to note that we have performed multiple runs of pre-training with our algorithm and are reporting the best-performing one. However, the results between different runs can vary slightly. This can be seen in Figure 7.1 in the ablations section.

Before running further experiments, we evaluate the representations from different layers of the network and proceed with the best-performing ones, respectively. Figure 5.2 shows the highest achieved accuracy for various layers of the network (in linear evaluation on the complete dataset). Following the naming convention introduced in section 2.4.7 (see Figure 2.7), layers **L.1-L.4** correspond to the layers at the end of the first four residual blocks (layer1-4 in 2.7), and **average-pooling** is the layer following the last one, before the linear classification head. It can be seen that the performance increases with increasing depth, which is in line with the findings of [34] for the ResNet architecture. Therefore the following linear evaluation will be conducted on representations extracted at the adaptive pooling layer (right before the linear head).

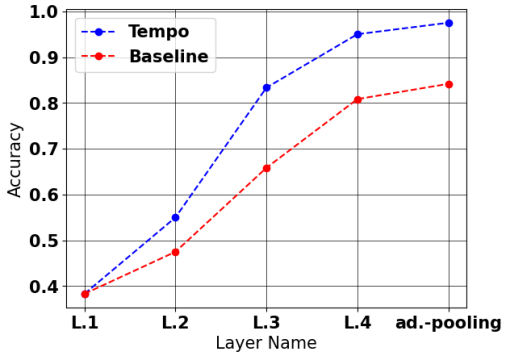
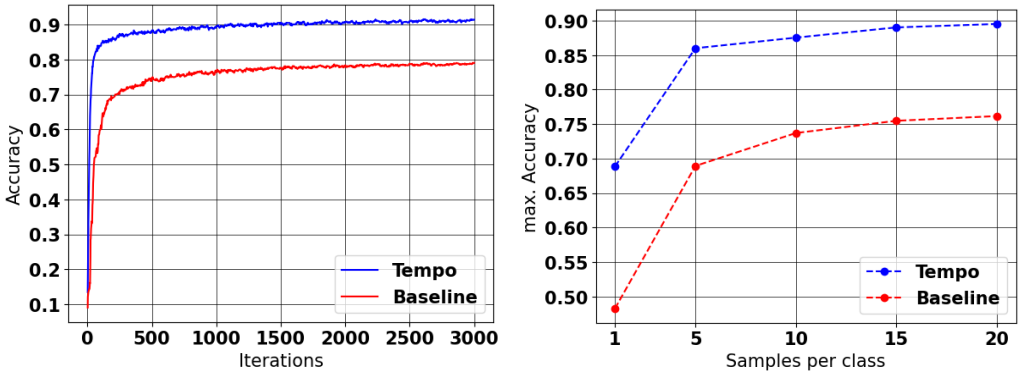


Figure 5.2: Top linear-evaluation accuracies for various layers of the network.

To alleviate differences between different runs of training of the linear classifier, we perform 100 runs of linear evaluation on the same set of representations and report the mean of the respective metrics. We are training the linear classifiers for 3000 iterations using SGD with a batch size of 20, learning rate of 0.01, and weight decay of 1e-4. The results are shown in Figure 5.3.

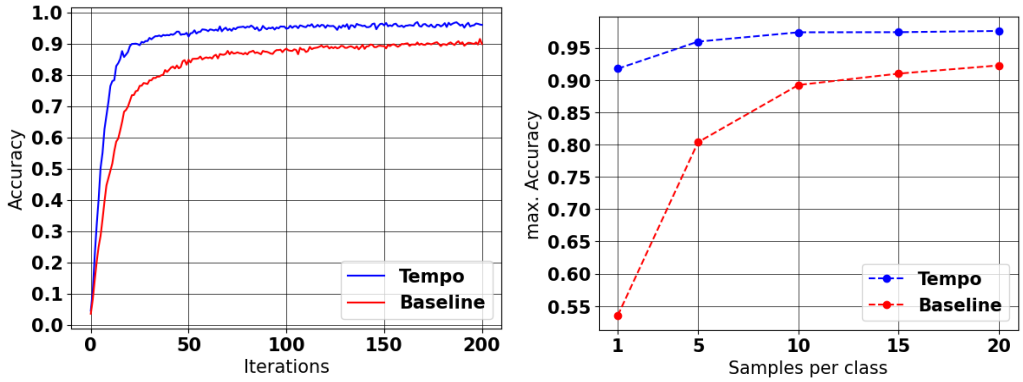


(a) (b)

Figure 5.3: Linear evaluation on full dataset (a) and subsets (b).

In Figure 5.3a, we can see the evaluation curve for linear-evaluation on the complete dataset. We observe a steeper slope for our model in comparison to the baseline as well as a higher asymptote with a maximum accuracy of 91.5% for our model in comparison to a maximum accuracy of 79.0% for the baseline. Both curves show roughly the same accuracy at the start, which was to be expected because only the weights for the backbone have been transferred, meaning the first predictions have been made at random for botch cases.

Figure 5.3b depicts the highest achieved accuracy for training on various subsets of the training data. Here we can observe a logarithmic increase in test accuracy with increasing size of the training examples per class. It is noteworthy that our method is able to achieve around 70% accuracy after only seeing a single example per class, while the baseline needs to see five times as many to achieve a similar accuracy. In addition to that, the accuracy our method can achieve after seeing Five samples per class is already substantially higher than what is achievable by the baseline, even



(a) (b)

Figure 5.4: Semi-supervised evaluation on full dataset (a) and subsets (b).

after seeing the complete dataset (Fig. 5.3a). Overall we can say that using our method results in a significant improvement for linear evaluation accuracy, regardless of the amount of training samples.

5.3 Semi-Supervised Evaluation

To further test the usefulness of the representations, we are making use of another evaluation method for image-classification. **Semi-Supervised evaluation** (also referred to as fine-tuning) is the second standard method for evaluating representations next to linear-evaluation [65, 28, 23]. It involves training the complete model, including the feature extractor, in comparison to linear-evaluation where it is effectively frozen. This results in an increased compute requirement but allows the image-representations that have been learned on the self-supervised task to be further tuned for the downstream task, which is what is commonly done in practice.

For the semi-supervised evaluation process, we are again using the backbone pre-trained for the linear evaluation and are following a similar setup. We are performing semi-supervised evaluation by again doing 100 runs of 3000 iterations using SGD with a batch size of 20. We have updated the learning rate to 0.05 after running a search and finding that it results in the highest accuracy for both models. The mean over all runs is reported in Figure 5.4.

In Figure 5.4a, we can see the testing accuracy for the first 200 iterations of training. Similar to 5.3a, we have a steeper slope for our method compared to the baseline. After 3000 iterations, we report a maximum accuracy of 91.4% for the baseline and 96.8% for our method.

Again in Figure 5.4b, we have the top accuracies for various subsets of the training data. Already at one sample per class, our method reports an accuracy on par with the baseline's accuracy after training on the full dataset. For the baseline, we can observe a behavior similar to the linear evaluation with the highest improvement going from 1 to 5 samples per class and then gradually less improvement with each following step. Our method is already reaching an accuracy of 91.7% at one sample per class, and then has only marginal improvements with each following step.

5.4 Object Detection/Instance Segmentation

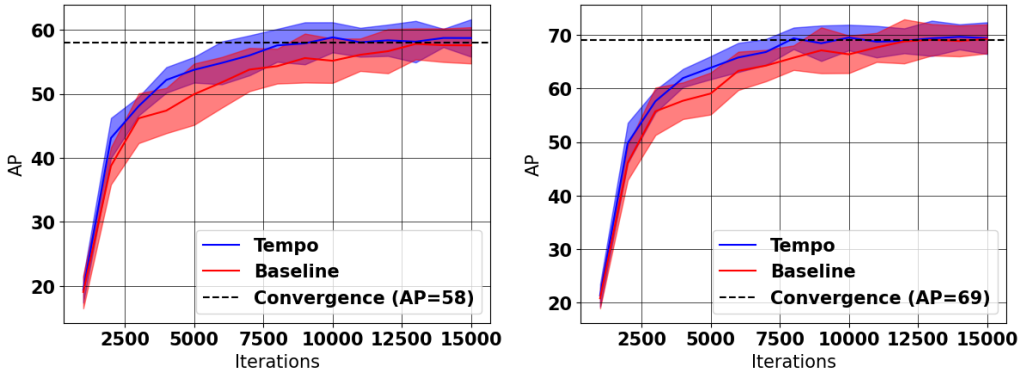
It is stated in [22] that a quality of good representations is that they generalize well to different tasks. To test this property, in this section, we are evaluating our method on the downstream tasks of object detection and instance segmentation:

- **Object Detection** is a computer vision task that involves classifying individual objects in an image, as well as localizing them by defining a bounding box around the object [27].
- **Instance Segmentation** also requires segmenting each occurrence of an object in addition to classifying and localizing it [27], meaning a precise segmentation mask needs to be defined around the object.

We are using the detectron2 [64] library to train a MaskRCNN [27] network that is simultaneously performing detection and segmentation. MaskRCNN is a neural network architecture for instance segmentation that is operating on representations produced by a CNN backbone. Pre-training this backbone and evaluating its downstream performance by fine-tuning the complete network is a common practice for the evaluation of image-representations [65, 28, 23]. To perform the evaluation, we are initializing the backbone of this network with weights learned by the respective methods.

For the baseline's supervised pre-training, we are again using weights learned on ImageNet-1k. MaskRCNN only supports ResNet-50 and larger models, which is why we have to train a new ResNet-50 Tempo backbone instead of reusing the model from the previous sections. To have a measure of the quality of the learned features prior to the experiments we select the backbone based on its linear-evaluation performance. We are measuring a maximum accuracy of 86.6% on linear evaluation for the ImageNet pre-trained backbone. The backbone trained with our algorithm achieves an accuracy of 95.7% after 50 epochs using the same hyperparameters as in the linear evaluation section. The accuracy of the detection and segmentation predictions will be evaluated using the Average Precision (AP) metric as it is defined for the COCO dataset [41] [1]. We are performing 5 runs of training for each method for 15000 iterations using detectron2's pre-defined "mask_rcnn_R_50_FPN_3x" configuration at a base learning rate of 25e-5 and a batch size of 5. The mean and the standard deviation are reported in Figure 5.5. For both detection and segmentation, while it can be observed that they are converging towards roughly the same value, we can see that our method has a higher slope with a faster convergence. We are measuring a maximum bounding-box AP of 58.83 for tempo and 57.80 for the baseline and a maximum mask AP of 69.62 for tempo and 69.24 for the baseline. Both are converging at a bounding-box AP of approximately 58, the baseline after training for ~ 13,000 iterations and our method after ~ 9,000 iterations (~ 30.7% faster). Similarly, both converge at a mask AP of approximately 69, the baseline after ~ 13,000 iterations while our method is already at 69.34 AP after ~ 8,000 iterations (~ 38.5% faster). The results indicate that while having no significant impact on the final performance, using our method can lead to an improvement in training speed.

[1] https://cocodataset.org/#detection-eval



(a) Bounding Box AP

(b) Segmentation Mask AP

Figure 5.5: Object detection and instance segmentation AP scores.

6 Explanation

Explainable Artificial Intelligence (XAI) is a sub-discipline of machine-learning that aims to provide explanatory feedback to machine-learning models, which in turn enables the enhancement of such models [47]. In a prior chapter, it was demonstrated experimentally that the features learned by our method can outperform the raw ImageNet features for the task of classifying ASL hand signs. In this current chapter, we intend to utilize an XAI algorithm, specifically Layer-wise Relevance Propagation (LRP) [2], to gain an understanding of the decision-making process of the network that leads to this improvement. "LRP operates by propagating the prediction $f(x)$ backwards in the neural network, by means of purposely designed local propagation rules" [47]. By doing so, we can obtain a measure of the contribution of each input-pixel to the decision, and visualize these contributions through a heatmap. This visualization highlights the features of the input image that contributed the most to the classification decision, as depicted in Figure 6.1.

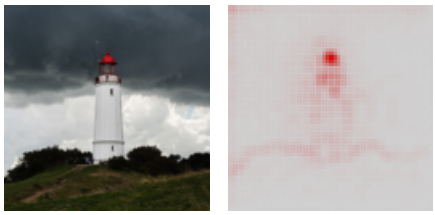


Figure 6.1: Input image on the left with the LRP heatmap on the right. We can see the dome highlighted as the main feature contributing to the the correct classification decision.

To conduct the test, we are again using the Resnet-34 backbone weights obtained in the evaluation section. We are freezing the backbone and only tuning the linear classification head on the subset with 20 samples per class. We have chosen not to finetune the backbone to ensure that we are evaluating the features as they have been learned by our self-supervised algorithm (without further supervised finetuning). Then we compute the pixel-wise relevance scores using the Zennit [1] library and plot the resulting heatmaps. To identify specific cases in which our methods improves on the baseline, we analyze the confusion matrices for the predictions on the test-set (see Figure 6.2).

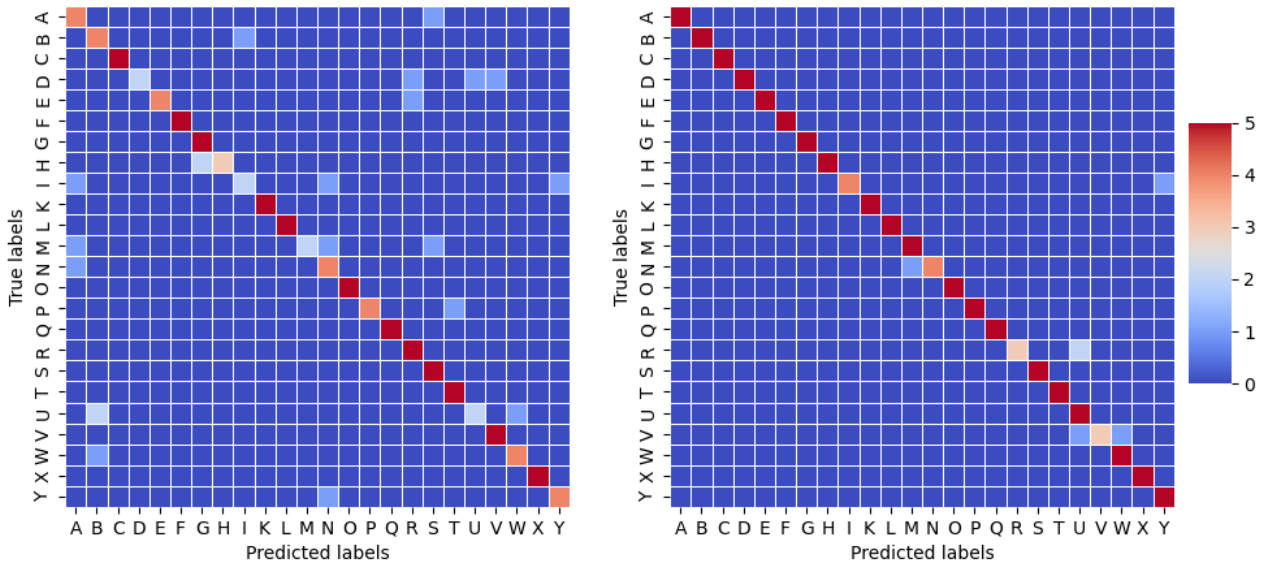


Figure 6.2: Confusion matrices for Baseline (left) and Tempo (right).

We can identify the letters D,I,M, and U as the biggest points of confusion for the baseline while they have been classified mostly correctly by our model. Upon further investigation, we can see that the letter D was misclassified as R,U, and V. Looking at the respective hand figures (see Figure 5.1) we observe that they are quite similar in their appearance regarding the forearm and the wrist region, with the differentiating factor being the figure formed by the stretched-out fingers and the upper palm region. Looking at the other examples, similar observations can be made. The heatmaps produced for the letter D are depicted in Figure 6.3.

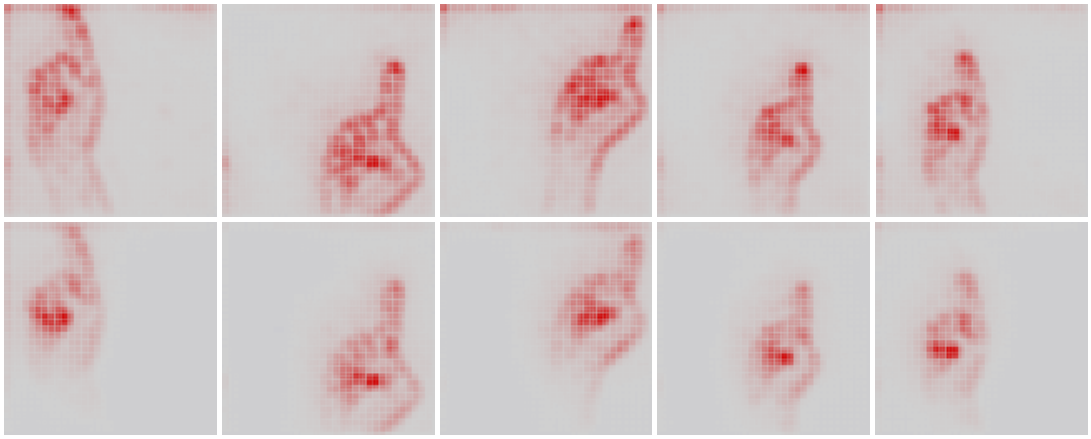


Figure 6.3: Heatmaps produced for the letter D, with the baseline heatmaps on the top and Tempo on the bottom.

It can be seen in Fig. 6.3 (also 6.5) that the model trained using our method attributes less relevance to the wrist region and the forearm, which are common features among most of the figures. Instead, the most relevance has been attributed to a single feature in the center of the palm just below where the thumb meets the middle finger and the ring finger. The improvement in accuracy that our model is achieving by relying on this feature indicates that our method is learning

features that are strongly representative of their class. Indeed, putting less emphasis on common features and a more consistent attribution of relevance to cleaner features are observations that can be made throughout most of the heatmaps (see 6.5 for more, and the supplementary material for the full list of heatmaps). We also want to note that in some cases the classification decision seems to have been negatively impacted (R and V). Nevertheless, the fact that the overall classification accuracy has been improved indicates that the features learned by our model are more useful for the downstream task.

6.1 Quantitative Evaluation

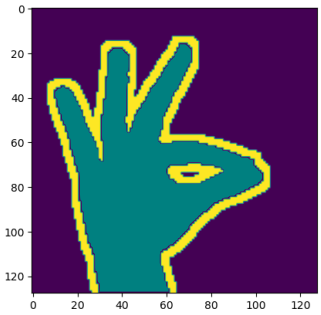


Figure 6.4: Mask defining RoI for one sample.

Another observation that can be made from the heatmaps is that the baseline, in comparison to our model, seems to suffer more strongly from spurious correlations [47], meaning its decisions are more often attributed to features that are unrelated to the object of interest. Figure 6.5 shows some examples in which this effect is particularly pronounced.

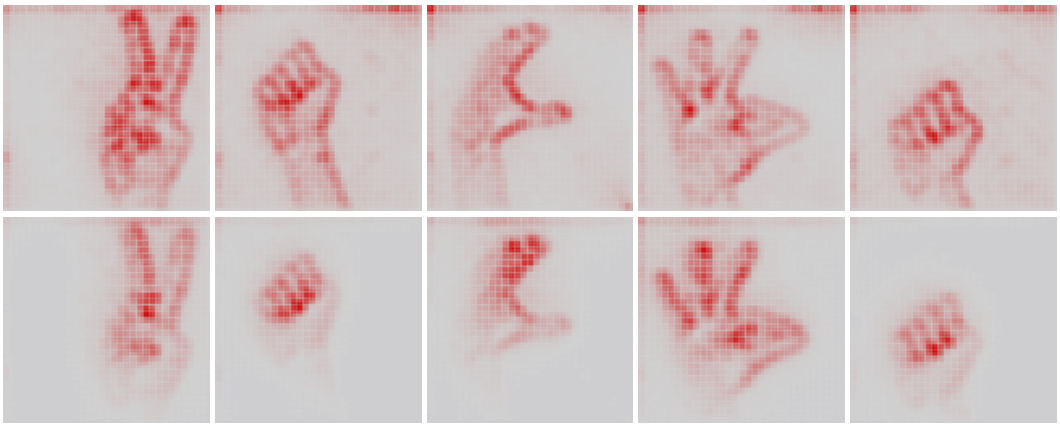


Figure 6.5: LRP heatmaps of selected samples showing less spurious correlations in the heatmaps created for models that have been trained using our method (bottom) compared to the baseline model (top).

To quantify the occurrence of these spurious correlations over the complete test-set, we are computing the Relative Relevance (RR) [36] of the object of interest (which, in our case, is the hand

figure). Inspired by a method proposed in [33], we define a region of interest (RoI) based on the segmentation masks we have created for the instance segmentation task. The RoI consists of the mask itself (green in Fig. 6.4) and a border around it (yellow in Fig. 6.4). The border is added because edges are a feature that CNNs commonly pick up. By including the border, we also attribute for pixels that lay outside the mask but are part of said edges. The RR score of an attribution map can then be calculated by the following equation:

$$\mathrm{RR}(R, M) = \frac{\sum_{i,j}(R \cdot M)_{ij}}{\sum_{i,j} R_{ij}} \tag{6.1}$$

where R is a matrix containing the pixel-wise attribution scores for the input image, and M is the corresponding bitmask with a 1 for entries inside the RoI and 0 otherwise. This score corresponds to the percentage of the relevance attributions that lay within the RoI. We are reporting a mean RR score of 66.17% for the baseline and 75.66% for our method over the complete testing set, which reaffirms the observations we have previously made.

6.2 Adversarial Attack

In the previous section, we have shown that the model trained using our method is less prone to attributions unrelated to the object of interest. We suspect that the baseline's features are being activated by the noise that is occurring in the image. Therefore we expect this effect to be amplified by introducing extra noise to them. To test this further, we create adversarial examples by introducing various degrees of additional noise to the test images to compare the models' robustness. We are creating an adversarial sample for an image by adding Gaussian noise to it that is sampled from the distribution $\mathcal{N}(0,\ 0.1)$ and then scaled by the "noise level" factor. In Figure 6.6, we can see that our model is almost unaffected by the noise and makes exclusively correct predictions up to a noise level of 0.5. At the same time, the baseline is already getting affected at a noise level of 0.1. Figure 6.7 shows a second example. This time we can see slightly more spurious correlations in the heatmaps of our model, with the predictions still being unaffected. The baselines' heatmaps are, again, very heavily affected, with only minimal attributions to the object itself at a noise level of 0.4 and 0.5. This pattern repeated itself for the rest of the tests we performed (heatmaps can be found in the supplementary material), leading us to the conclusion that the features learned by our method are less prone to be affected by noise.

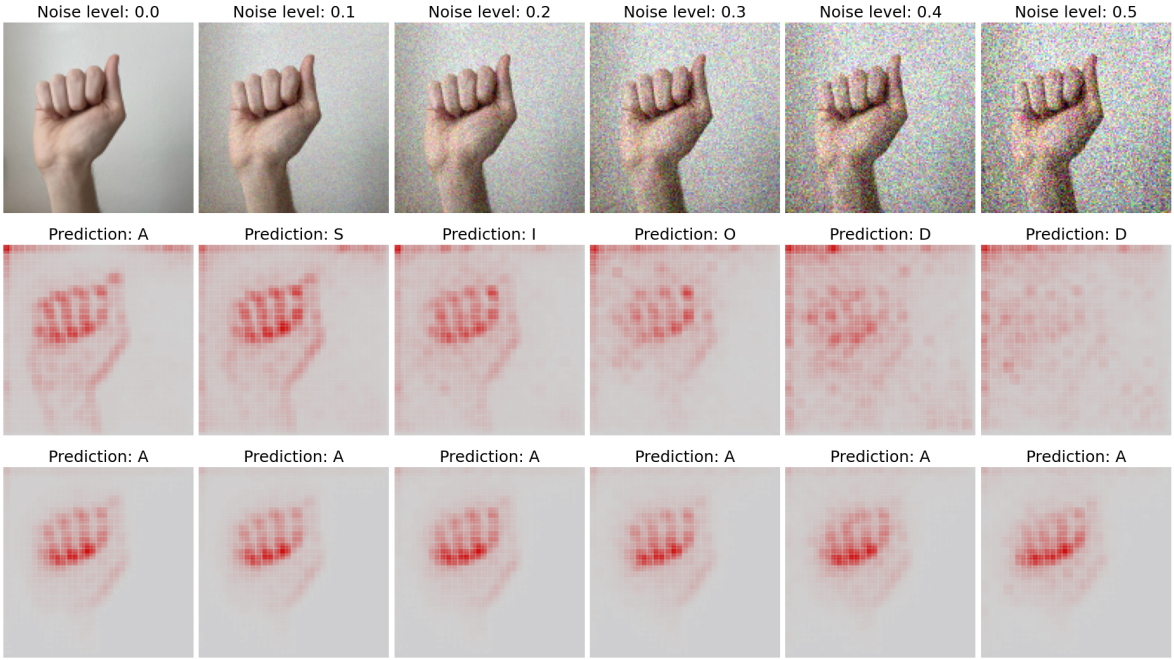


Figure 6.6: Example of adding noise to an input image depicting figure A at different noise levels with corresponding baseline heatmaps in the second row and tempo heatmaps in the third row.

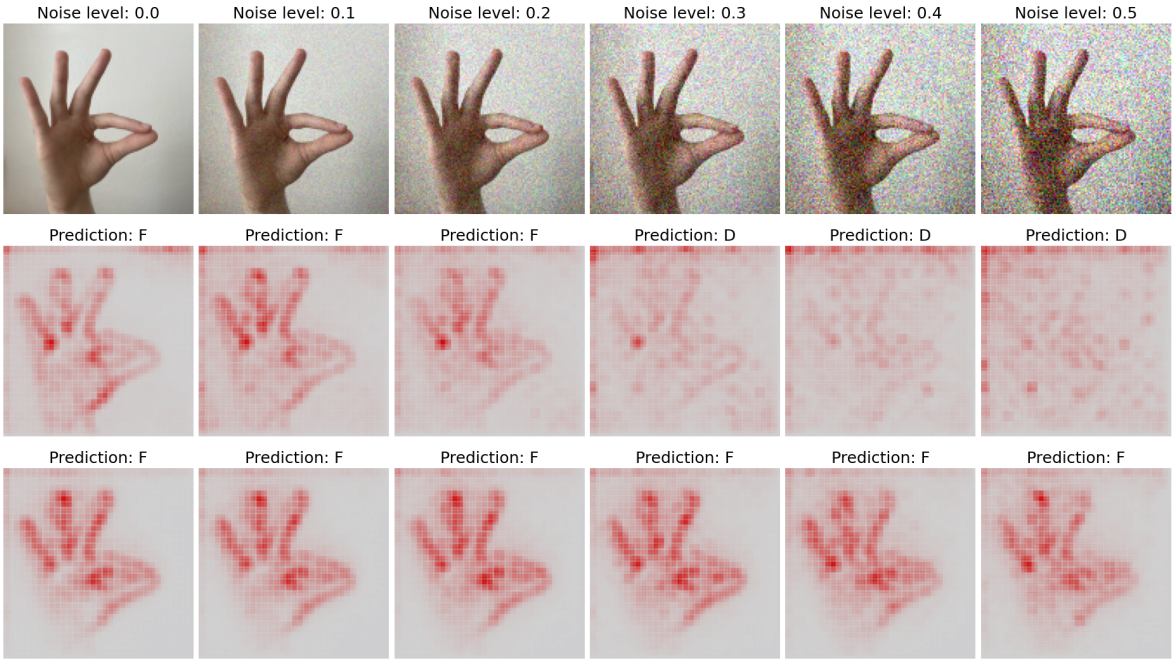


Figure 6.7: Example of adding noise to an input image depicting figure F at different noise levels with corresponding baseline heatmaps in the second row and tempo heatmaps in the third row.

7 Ablations

In the ablations chapter, we study the individual components' contributions to our algorithm. In the first part, we remove the proximity component, resulting in a setting similar to [46]. In the second section, we are training a backbone with randomly initialized weights to determine the role of ImageNet pre-training in our algorithm.

7.1 Proximity

Different from previous approaches like [46], our method is not limited to the immediate neighborhood for sampling positive pairs for its loss function. Instead, a proximity parameter that determines the maximum distance to the reference frame from which the second frame can be sampled needs to be defined. In the following, we show the effect of omitting the proximity component (by using a uniform distribution with $\tau = 1$, see equation 4.1):

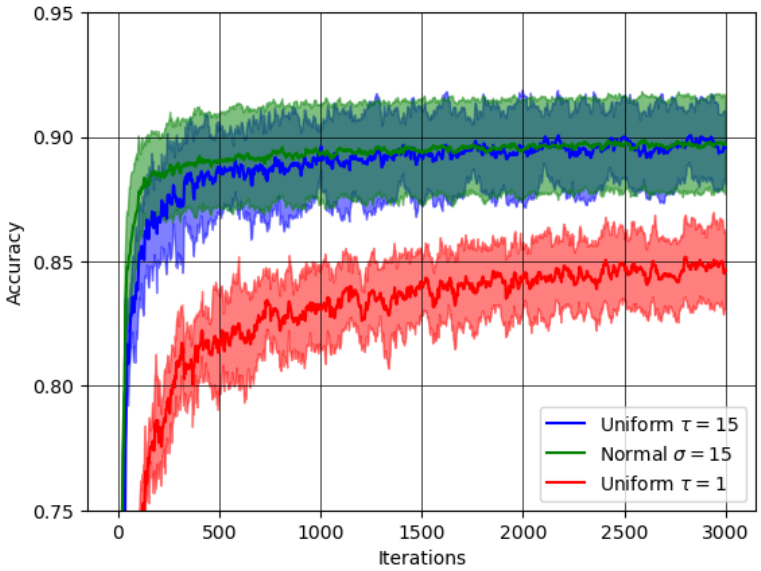


Figure 7.1: Effect of the proximity parameter on the linear-evaluation accuracy.

We are training five backbones for each proximity setting using our algorithm, each for ten epochs. For the training, we have increased the dimensionality of the embeddings to 4096, the batch size to 200, and decreased the learning rate to 1e-5 as we have found this will result in a slightly more stable training in combination with the AdamW [43] optimizer (compared to the setting from the evaluation section). The lambda value for the loss function has remained at $\lambda = 0.001$. Then, each backbone is evaluated linearly for ten runs, and the mean over the runs is computed (resulting in one value per backbone). After that, the mean and standard deviation for each proximity setting

is computed and reported in 7.1. It can be seen that by choosing a fitting value for the proximity parameter, the accuracy can be improved by a significant margin. In addition to that, both uniform and normal distributions seem to perform similarly well when the proximity parameter is chosen appropriately.

7.2 ImageNet Pretraining

Our algorithm relies on the usage of weights that have been pre-trained on the ImageNet dataset for initialization. We removed this component for the ablation studies by randomly initializing the backbone's weights. We are again increasing the dimensionality of the embeddings to 8192, the batch size to 1024, and training for 1000 epochs, keeping the remaining hyperparameters the same as in the previous section. The following figure shows a comparison between the linear evaluation accuracies of our algorithm trained from scratch and pre-trained on ImageNet, as well as the ImageNet-only baseline.

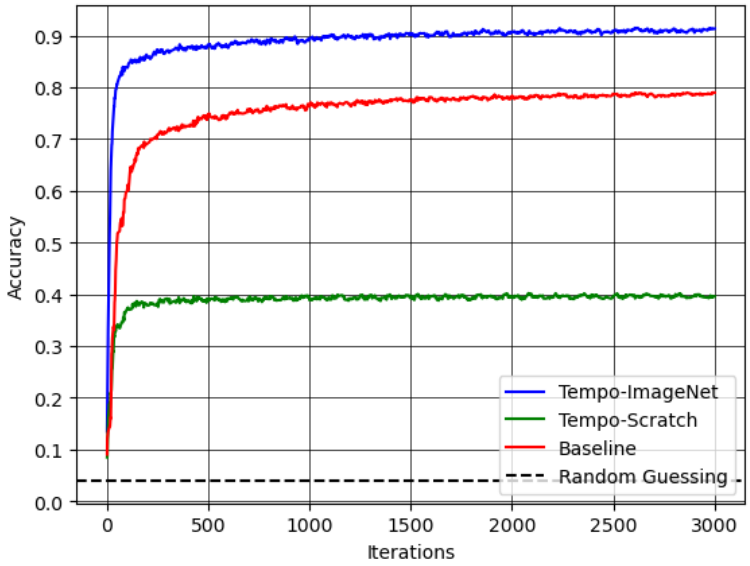


Figure 7.2: Comparison of linear-evaluation testing accuracy between training from scratch using our algorithm (green), using raw ImageNet features (baseline, red), and training with our algorithm with weights pre-trained on ImageNet (blue).

While training from scratch performs significantly better than random guessing, we can see a substantial decrease in maximum accuracy without ImagNet initialization, which is even well below the baseline's accuracy. This suggests that our algorithm, in its current form, is not able to learn competitive features but, when starting out from ImageNet features, is able to improve them significantly.

8 Conclusion

8.1 Discussion

In the context of this research paper, we have developed an algorithm that can extract image features from unlabeled video data to present an alternative to ImageNet pre-training for situations where labeled image data is rare, but video data of the data-generating process is available in abundance. We have evaluated this algorithm on the tasks of image classification, object detection, and instance segmentation and, in the case of image classification, were able to improve on the baseline method's accuracy significantly. Additionally, for the detection and segmentation tasks, we have recorded an improvement in training speed. We have observed an increase of around 12.5%, training on the complete data and up to 20.6% in the low-shot regime for linear evaluation. In fine-tuning, the improvement is around 3.7% for the complete data, while in the low-shot regime, we get an improvement of up to 38.2%. This improvement in classification accuracy supports our hypothesis that pre-training on data that has been created by the same data-generating process as the data used for the downstream task will produce features that are more useful to the downstream task than general-purpose features. This is especially supported by the fact that we get the greatest improvement in classification accuracy when we use the least amount of samples per class for training, which indicates that the knowledge gathered through pre-training has been highly beneficial to the downstream task. We have then gathered further evidence for this assumption through our explainability research, in which we have found that the decisions made using our algorithm are bound to features that are more relevant to the object of interest and are, in general, subject to less spurious correlations. Additionally, we have demonstrated that they are less prone to be affected by noise.

Through our work, we have shown that, if available, learning features from video with the use of self-supervised learning is a viable option that can bring an improvement over ImageNet pre-training in multiple computer vision tasks without creating any additional labeling overhead. We have demonstrated for our AMA dataset that by applying our proposed method, the need for labeled data can be significantly decreased while simultaneously improving the classification accuracy. We were able to achieve a similar level of accuracy in fine-tuning while only using 5% of the labeled examples compared to ImageNet fine-tuning.

We are seeing potential real-world applications for our method and similar approaches for processes that already use video recordings. One such process that comes to mind is Metal-Based Additive Manufacturing, in which video cameras for visible light and infrared imaging are used for monitoring [59], and CNN-based machine learning approaches are being used for classifying and detecting anomalies and other applications [56, 62]. However, it remains to be seen how well our approach translates to different domains and datasets.

8.2 Further Research

We have seen in section 7.1 that the method of sampling positive pairs can influence the performance of our algorithm. By increasing the maximum distance to the reference frame from which we allow to sample, we have been able to record an improvement in classification accuracy, however, there was no difference between sampling with a uniform distribution compared to sampling with a normal distribution (see Figure 7.1). Nevertheless, we believe that researching better methods for sampling positive pairs is a path that can enable learning higher quality features, which will ultimately lead to a better performance. The sampling strategies we have used so far have no way of avoiding "bad positive samples" that can occur when the region from which we sample the frames extends over two or more segments of the video (for example, when we transition from one hand figure to another). We suggest further research to be directed toward strategies that identify such transitions between segments in a video to reduce the occurrence of bad positive samples.

Supplementary Material

The supplementary material can be found in the digital submission provided with the thesis as well as in the accompanying repository: `https://github.com/myasincifci/bachelor_thesis_code`.

[1] Christopher J. Anders et al. “Software for Dataset-wide XAI: From Local Explanations to Global Insights with Zennit, CoRelAy, and ViRelAy”. In: *CoRR* abs/2106.13200 (2021).

[2] Sebastian Bach et al. “On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation”. In: *PloS one* 10.7 (2015), e0130140.

[3] Horace B Barlow et al. “Possible principles underlying the transformation of sensory messages”. In: *Sensory communication* 1.01 (1961), pp. 217–233.

[4] Yoshua Bengio and James Bergstra. “Slow, decorrelated features for pretraining complex cell-like networks”. In: *Advances in neural information processing systems* 22 (2009).

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation learning: A review and new perspectives”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[6] Jane Bromley et al. “Signature verification using a" siamese" time delay neural network”. In: *Advances in neural information processing systems* 6 (1993).

[7] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[8] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[9] Ting Chen et al. “Big self-supervised models are strong semi-supervised learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 22243–22255.

[10] Xinlei Chen and Kaiming He. “Exploring simple siamese representation learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 15750–15758.

[11] Xinlei Chen et al. “Improved baselines with momentum contrastive learning”. In: *arXiv preprint arXiv:2003.04297* (2020).

[12] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[13] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR’05)*. Vol. 1. Ieee. 2005, pp. 886–893.

[14] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[15] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[16] Vincent Dumoulin and Francesco Visin. “A guide to convolution arithmetic for deep learning”. In: *ArXiv e-prints* (Mar. 2016). eprint: 1603.07285.

[17] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[18] Isabel Funke et al. “Temporal coherence-based self-supervised learning for laparoscopic workflow analysis”. In: *OR 2.0 Context-Aware Operating Theaters, Computer Assisted Robotic Endoscopy, Clinical Image-Based Procedures, and Skin Image Analysis: First International Workshop, OR 2.0 2018, 5th International Workshop, CARE 2018, 7th International Workshop, CLIP 2018, Third International Workshop, ISIC 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16 and 20, 2018, Proceedings 5*. Springer. 2018, pp. 85–93.

[19] Zoubin Ghahramani. “Unsupervised learning”. In: *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures* (2004), pp. 72–112.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[21] Ross Goroshin et al. “Unsupervised learning of spatiotemporally coherent metrics”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4086–4093.

[22] Priya Goyal et al. “Scaling and benchmarking self-supervised visual representation learning”. In: *Proceedings of the ieee/cvf International Conference on computer vision*. 2019, pp. 6391–6400.

[23] Jean-Bastien Grill et al. “Bootstrap your own latent-a new approach to self-supervised learning”. In: *Advances in neural information processing systems* 33 (2020), pp. 21271–21284.

[24] Raia Hadsell, Sumit Chopra, and Yann LeCun. “Dimensionality reduction by learning an invariant mapping”. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*. Vol. 2. IEEE. 2006, pp. 1735–1742.

[25] Moritz Hardt and Benjamin Recht. *Patterns, predictions, and actions: Foundations of machine learning*. Princeton University Press, 2022.

[26] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[27] Kaiming He et al. “Mask r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[28] Kaiming He et al. “Momentum contrast for unsupervised visual representation learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 9729–9738.

[29] Sepp Hochreiter. “The vanishing gradient problem during learning recurrent neural nets and problem solutions”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116.

[30] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[31] Dinesh Jayaraman and Kristen Grauman. “Slow and steady feature analysis: higher order temporal coherence in video”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 3852–3861.

[32] Longlong Jing and Yingli Tian. “Self-supervised visual feature learning with deep neural networks: A survey”. In: *IEEE transactions on pattern analysis and machine intelligence* 43.11 (2020), pp. 4037–4058.

[33] Jacob Kauffmann et al. “From clustering to cluster explanations via neural networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[34] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. “Revisiting Self-Supervised Visual Representation Learning: Supplementary Material”. In: ().

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[36] Sebastian Lapuschkin et al. “Unmasking Clever Hans predictors and assessing what machines really learn”. In: *Nature communications* 10.1 (2019), p. 1096.

[37] Yann LeCun, Fu Jie Huang, and Leon Bottou. “Learning methods for generic object recognition with invariance to pose and lighting”. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Vol. 2. IEEE. 2004, pp. II–104.

[38] Yann LeCun and Ishan Misra. *Self-supervised learning: The dark matter of intelligence.* Mar. 2021. URL: https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/.

[39] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[40] Hsin-Ying Lee et al. “Unsupervised representation learning by sorting sequences”. In: *Proceedings of the IEEE international conference on computer vision.* 2017, pp. 667–676.

[41] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context”. In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13.* Springer. 2014, pp. 740–755.

[42] Tony Lindeberg. “Scale invariant feature transform”. In: (2012).

[43] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101* (2017).

[44] Ishan Misra and Laurens van der Maaten. “Self-supervised learning of pretext-invariant representations”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 2020, pp. 6707–6717.

[45] Ishan Misra, C Lawrence Zitnick, and Martial Hebert. “Shuffle and learn: unsupervised learning using temporal order verification”. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14.* Springer. 2016, pp. 527–544.

[46] Hossein Mobahi, Ronan Collobert, and Jason Weston. “Deep learning from temporal coherence in video”. In: *Proceedings of the 26th Annual International Conference on Machine Learning.* 2009, pp. 737–744.

[47] Grégoire Montavon et al. “Layer-wise relevance propagation: an overview”. In: *Explainable AI: interpreting, explaining and visualizing deep learning* (2019), pp. 193–209.

[48] Sheila J. Nayar. “Columbia Object Image Library (COIL100)”. In: 1996.

[49] Emilio Soria Olivas et al. *Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques: Algorithms, methods, and techniques.* IGI global, 2009.

[50] Veenu Rani et al. “Self-supervised Learning: A Succinct Review”. In: *Archives of Computational Methods in Engineering* (2023), pp. 1–15.

[51] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems* 28 (2015).

[52] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386.

[53] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[54] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[55] Shibani Santurkar et al. “How does batch normalization help optimization?” In: *Advances in neural information processing systems* 31 (2018).

[56] Luke Scime et al. “Layer-wise anomaly detection and classification for powder bed additive manufacturing processes: A machine-agnostic algorithm for real-time pixel-wise semantic segmentation”. In: *Additive Manufacturing* 36 (2020), p. 101453.

[57] Boris Sekachev et al. *opencv/cvat: v1.1.0*. Version v1.1.0. Aug. 2020. DOI: 10.5281/zenodo.4009388. URL: https://doi.org/10.5281/zenodo.4009388.

[58] Thalles Santos Silva. “A Few Words on Representation Learning”. In: *https://sthalles.github.io* (2021). URL: https://sthalles.github.io/a-few-words-on-representation-learning/.

[59] Gustavo Tapia and Alaa Elwany. “A review on process monitoring and control in metal-based additive manufacturing”. In: *Journal of Manufacturing Science and Engineering* 136.6 (2014).

[60] Hind Taud and JF Mas. “Multilayer perceptron (MLP)”. In: *Geomatic approaches for modeling land change scenarios* (2018), pp. 451–455.

[61] Richard A. Tennant. *textsThe American Sign Language handshape dictionary*. Clerc Books, Gallaudet University Press, 1998.

[62] Mahsa Valizadeh and Sarah Jeannette Wolff. “Convolutional Neural Network applications in additive manufacturing: A review”. In: *Advances in Industrial and Manufacturing Engineering* (2022), p. 100072.

[63] Laurenz Wiskott and Terrence J Sejnowski. “Slow feature analysis: Unsupervised learning of invariances”. In: *Neural computation* 14.4 (2002), pp. 715–770.

[64] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[65] Jure Zbontar et al. “Barlow twins: Self-supervised learning via redundancy reduction”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12310–12320.

[66] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer. 2014, pp. 818–833.

[67] Wen Zhang et al. “A survey on negative transfer”. In: *IEEE/CAA Journal of Automatica Sinica* (2022).

[68] Huiyu Zhou, Yuan Yuan, and Chunmei Shi. “Object tracking using SIFT features and mean shift”. In: *Computer vision and image understanding* 113.3 (2009), pp. 345–352.